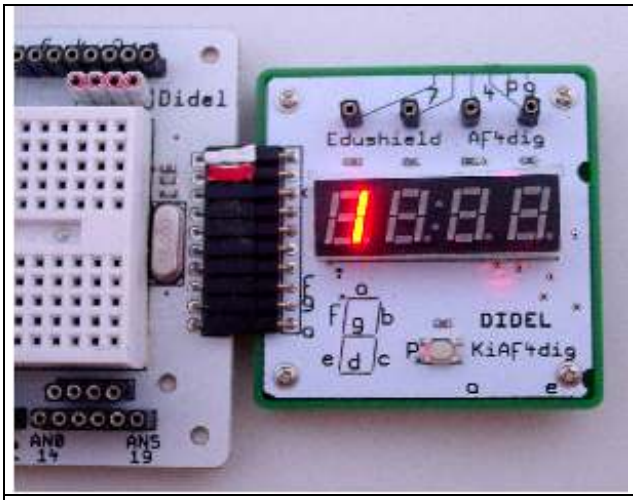
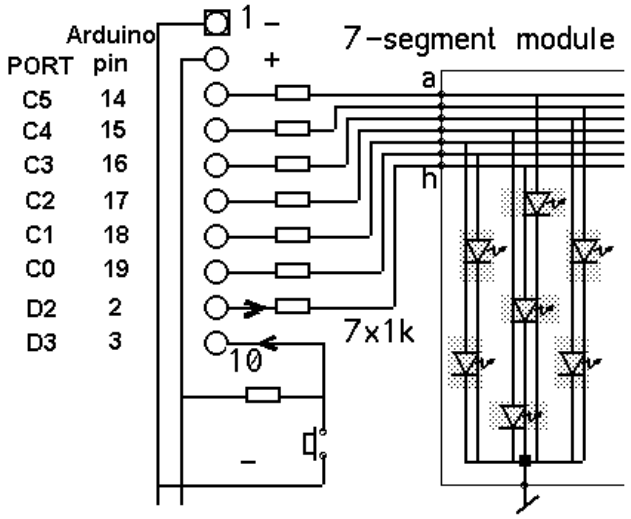
Diduino - expériences avec le Kidule Affichage 4 digits

Le but est de comprendre comment on représente des chiffres, puis des nombres sur un affichage à 7 segments. Les programmes sont en www.didel.com/kidules/CKiAf4.zip
Dans une première étape, tout se passe comme si le module n'avait qu'un seul digit commandé par le connecteur Kidule. 7 bits commandent les segments, L'état 1 (HIGH) allume (avec d'autres affichages, un 0 (LOW) allume). Le 8e bit (bit 7) permet de lire le poussoir.



Affectation des bits 0-7
Arduino pins: 14, 15, 16, 17, 18, 19, 2, 3
Arduino ports: PORTC 0 .. 5, PORTD 2.3
les bits 0 à 6 sont en sortie, actifs à 1



Le bit 7 est en entrée, actif à 0 si on presse.
Sur l'Edushield Arduino, la pin 4 allume le :

Exercice 1 Pour allumer les deux segments, on peut le faire "à la Arduino"

```
// Segments.ino
#define  SegA 14
#define  SegB 15
#define  SegC 16
. . . . .
#define  ON HIGH
#define  OFF,LOW
```

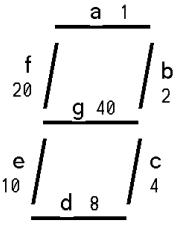
```
void setup () {
  pinMode (SegA,OUTPUT);
  pinMode (SegB,OUTPUT);
  pinMode (SegC,OUTPUT);
. . . .
}
```

```
void loop () {
   digitalWrite (SegA,ON);
   delay (500);
   digitalWrite (SegA,OFF);
   delay (500);
   . . . . .
{
```

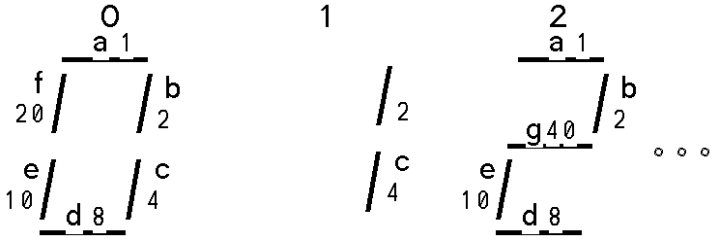
Chargez le programme ***Segments.ino*** et ajoutez ou enlevez des instructions
Ecrivez le programme **AllSeg.ino** qui allume les segments a b c d e f g.
Pour cela sauvez Segments.ino en AllSeg.ino et modifiez (aussi le titre).

Vous voyez qu'allumer plusieurs segments nécessite beaucoup d'instructions. Raisonner avec le mot binaire 8 bits qui représente chaque chiffre est plus efficace.

binaire	hexa	décimal
0b00000001	0x01	1
0b00000010	0x02	2
0b00000100	0x04	4
0b00001000	0x08	8
0b00010000	0x10	16
0b00100000	0x20	32
0b01000000	0x40	64
0b10000000	0x80	128
rang 7 6 5 4 3 2 1 0		



noms	g	f	e	d	c	b	a
bits	6	5	4	3	2	1	0
poids	40	20	10	8	4	2	1



0b00111111 0x3F
0b00000110 0x06
0b01011011 0x5B

On peut alors donner un nom à chaque chiffre et lui faire correspondre sa valeur en hexa.
Pour transférer sur le connecteur 8 bits du kidule, il faut un peu d'acrobatie car le processeur n'a pas un port 8 bits complet. Si vous ne connaissez pas les fonctions logiques, assez délicates à bien comprendre, attendez d'avoir fait des progrès en C. La procédure `KiWrite ();` s'utilise comme `delay();` Au lieu de mettre une durée, on met une valeur 8 bits qui est une variable qui est distribuée sur les bons bits des ports C et D. Elle est expliquée dans le Kidule dé
.
De même pour le setup. On dit en une fois aux registres de direction DDRC et DDRD si les lignes que l'on utilise sont en entrée ou en sortie, plutôt que de dire pour chaque pin quelle est sa direction, et chaque fois le compilateur génère plusieurs instructions pour corriger DDRC ou DDRD.

```
//AfDigits.ino
#define Zero 0x3F
#define Un 0x06
#define Deux 0x5B
. . . .
void KiWrite (int kk) {
   PORTC = kk;
   PORTD |= kk>>4;   // force les 1
   PORTD &= kk>>4;   // force les 0
}
```

```
void setup () {
   DDRC  = 0b00111111;   // segm a..f
   DDRD |= 0b11110100;   // seg  g  out
   DDRD &= 0b11110111;   // poussoir in
}
```


```
void loop () {
   KiWrite (Zero);
   delay (500);
   KiWrite (Un);
   delay (500);
   KiWrite (Deux);
   delay (500);
   . . . . .
}
```

Exercice 2 Chargez le programme ***AfDigits.ino***, renommez-le en AfAllDigits.ino et complétez la table pour afficher d'autres chiffres.
Si vous êtes pressé, chargez ***Compter09.ino*** mais ne vous arrêtez pas là! Complétez ce programme (compter0F.ino) pour afficher les chiffres hexa de 0 à F.

Plus efficace: une table indexée

Plutôt que des #define, mettons les valeurs binaires des segments dans un tableau de variables, ce que le C note simplement `byte Digit [10] ;`
Le processeur va réserver 10 cases successives, numérotées de 0 à 9, de 1 octet (8 bits) chacune.
Pour remplir cette table il suffit d'énumérer les valeurs binaires (notées en hexa) entre accolades:
`byte digit[10]={0x3f,0x06,0x5B,0x4F,0x66,0x6D,0x7D,0x07,0x7F,0x67};`

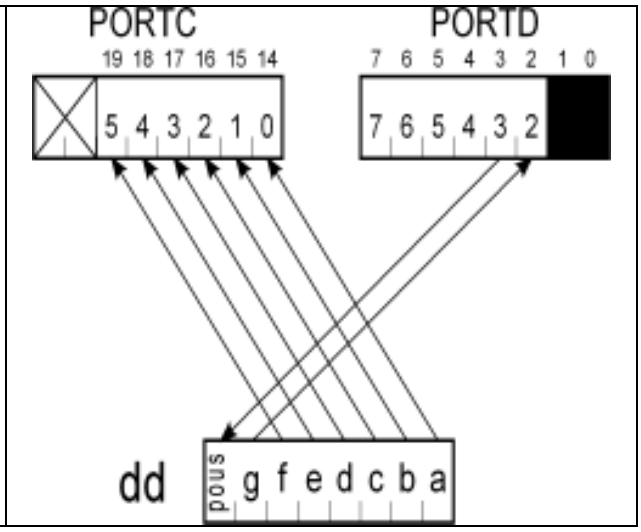
Pour lire la valeur associée au chiffre 3, qui est dans la case 3, on écrit `digit [3];` et pour afficher ce 3 sur le port Kidule on écrit `WriteKi (digit [3]) ;` Joli n'est-ce pas!
Le programme qui compte de 3 en 3 avec ces instructions s'appelle ***ComptePar3.ino.***
Cela permet de voir comment on calcule avec l'index en restant dans les valeurs qui nous intéressent Naturellement, il faut maintenant compléter la table pour avoir les chiffres hexa, et remarquer que l'index pour le F est 15 (programme ComptePar3Hex.ino).

Approche Arduino

Expliquée dans le mooc "Microcontrôleurs", l'approche Arduino est élégante, mais lente et utilise beaucoup de mémoire, ce qui n'est en général pas gênant.

L'image des segments est dans une variable et on utilisera le même tableau de digits.
La fonction CopyDigit () place les bits qui correspondent aux segments sur les bonnes pins, qui peuvent être dans un ordre quelconque. La correspondance avec les bits des ports est faite par un procédure compliquée.

On teste un segment après l'autre et on active ou désactive la pin correspondant au segment câblé. C'est la procédure CopyDigit. Pour afficher le chiffre 3, on appelle CopyDigit (3);



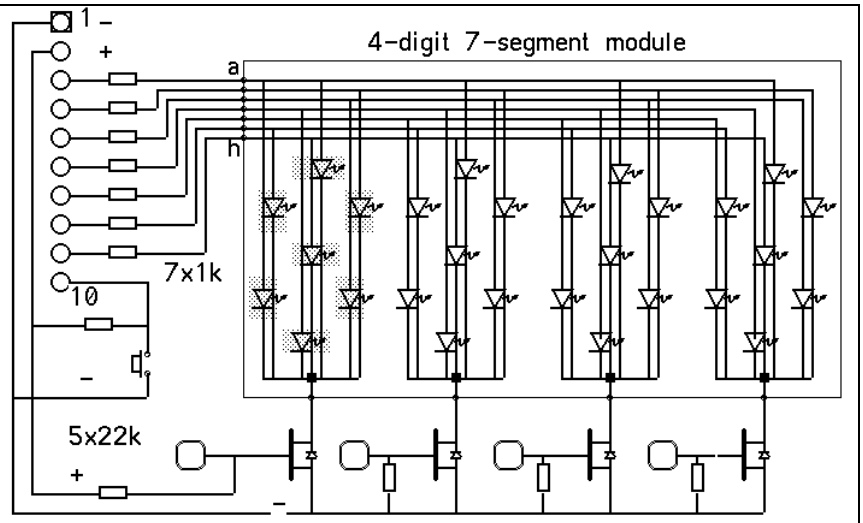
```
void CopyDigit (byte dd) {
if (dd & 1<<0) bitSet (PORTC,0) ; else bitClear (PORTC,0) ; //a
if (dd & 1<<1) bitSet (PORTC,1) ; else bitClear (PORTC,1) ; //b
if (dd & 1<<2) bitSet (PORTC,2) ; else bitClear (PORTC,2) ;
if (dd & 1<<3) bitSet (PORTC,3) ; else bitClear (PORTC,3) ;
if (dd & 1<<4) bitSet (PORTC,4) ; else bitClear (PORTC,4) ;
if (dd & 1<<5) bitSet (PORTC,5) ; else bitClear (PORTC,5) ;
if (dd & 1<<6) bitSet (PORTD,2) ; else bitClear (PORTD,2) ; //g
}
```

Avec une boucle for, on peut alors parcourir la table des chiffres pour les afficher une seconde chacun.
Modifiez AfAcompte.ino pour décompter.

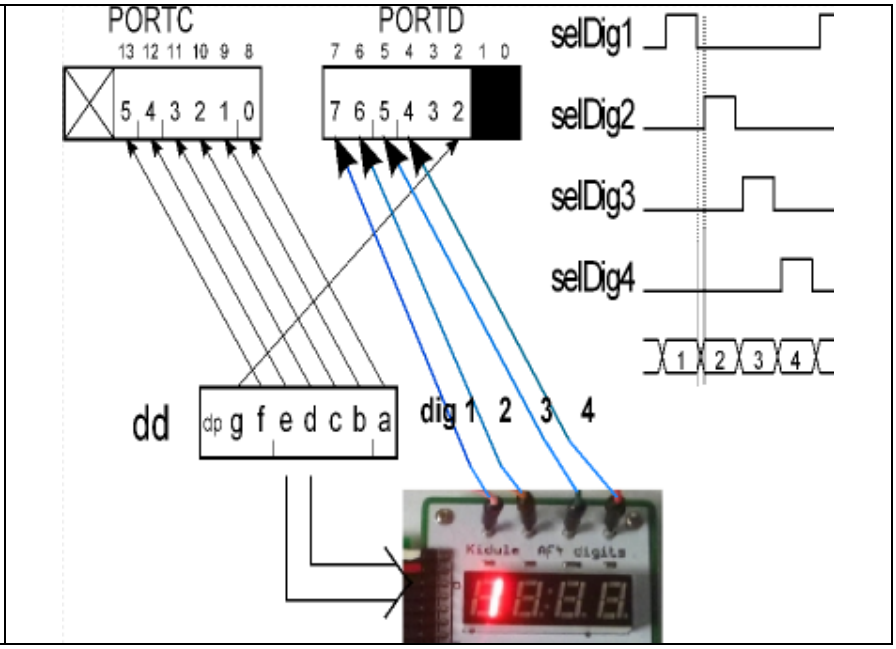
```
// AfAcompte.ino  Affiche 0 1 2 .. 9
… définitions, setup, CopyDigit
void loop () {
   for (byte i=0 ; i<10 ; i++ ) {
     CopyDigit ( digit [ i ] ) ;
     delay (1000) ;
   }
}
```

Balayer Le kidule a 4 digits, et 4 pins pour sélectionner l'allumage.

A cause du câblage de 4 résistances, le premier digit est allumé si on ne fait rien. Tirez un fil entre la première pin et du 0V sur une pin maquée -.
Le digit s'éteint. Tirez un fil vers le 5V marqué + et le digit choisi s'allume. Les segments de ces digits sont reliés. Pour afficher un nombre, il faut allumer un digit après l'autre. L'information doit venir de sorties supplémentaires d'Arduino, par exemple les pins 4,5,6,7, déjà initialisées en sorties dans les setup précédents.



Le problème est donc de sélectionner une position après l'autre et afficher le chiffre correspondant. On peut balayer dans un sens ou dans l'autre.
Si on veut que l'affichage n'ait pas de glitch (apparition d'un mauvais chiffre pendant quelques microsecondes), il faut désélectionner un digit, mettre l'information suivante dans le registre dd et sélectionner le digit suivant.



Afficher 1234

Utilisons des bitSet et bitClear plus efficaces que des digitalWrite (); pour sélectionner les 4 digits et afficher 1 2 3 4. C'es de plus compatible avec tous les compilateurs C.
Le programme fait ceci:

activer la pin 7, ce qui allume le digit de gauche
allumer le digit en envoyant le code du 1, attendre 10ms
désactiver la pin 7, activer la pin 6
allumer le 2e digit en envoyant le code du 2, attendre 10ms
désactiver la pin 6, activer la pin 5
. . . la boucle se termine en désactivant la pin 4.

La boucle dure 20 ms, ce qui est compatible avec la persistance rétinienne. Bougez l'affichage, cela fait apparaître le séquencement. Changer la valeur du délai (variables durAff) et observez l'effet. Essayez avec des microsecondes. Pourquoi est-ce que l'intensité diminue?

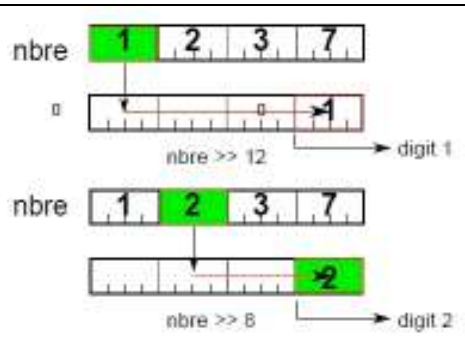
Remarque: Le microcontrôleur numérote toujours à partir de 0. Mais si on doit donner des noms aux digits, nos reflexes scolaires utilisent 1,2,3,4 !

```
// Af4val1234.ino
void setup ()
{
  DDRD |= 0b11111100 ;  //
  DDRC |= 0b00111111;   // segments a..f
  PORTD |= 0b00010000;
  PORTC |= 0b00000000;  // seg éteints
}

byte digit[10]={0x3f,0x06,0x5B,0x4F,0x66,0x6D,0x7D,0x07,0x7F,0x67};

void CopyDigit (byte dd) {
if (dd & 1<<0) bitSet (PORTC,0) ; else bitClear (PORTC,0) ; //a
if (dd & 1<<1) bitSet (PORTC,1) ; else bitClear (PORTC,1) ; //b
if (dd & 1<<2) bitSet (PORTC,2) ; else bitClear (PORTC,2) ;
if (dd & 1<<3) bitSet (PORTC,3) ; else bitClear (PORTC,3) ;
if (dd & 1<<4) bitSet (PORTC,4) ; else bitClear (PORTC,4) ;
if (dd & 1<<5) bitSet (PORTC,5) ; else bitClear (PORTC,5) ;
if (dd & 1<<6) bitSet (PORTD,2) ; else bitClear (PORTD,2) ; //g
}
#define SelDig1On bitSet (PORTD,7) ;  // actif
#define SelDig1Off bitClear (PORTD,7) ;
#define SelDig2On bitSet (PORTD,6) ;
#define SelDig2Off bitClear (PORTD,6) ;
#define SelDig3On bitSet (PORTD,5) ;
#define SelDig3Off bitClear (PORTD,5) ;
#define SelDig4On bitSet (PORTD,4) ;
#define SelDig4Off bitClear (PORTD,4) ;

int  durAf = 5 ;  // temps pour chaque digit
void loop() {  // affiche 1234
   CopyDigit (digit [ 1 ] ) ;
   SelDig1On;  delay (durAf) ;
   SelDig1Off;
    CopyDigit (digit [ 2 ] ) ;
    SelDig2On;  delay (durAf) ;
    SelDig2Off;
   CopyDigit (digit [ 3 ] ) ;
   SelDig3On; delay (durAf) ;
   SelDig3Off;
    CopyDigit (digit [ 4 ] ) ;
    SelDig4On;  delay (durAf) ;
    SelDig4Off;
}
```

Cette solution n'est pas très élégante, mais on peut voire pire avec des pages de "digitalWrite" !

Affiche un nombre BCD ou hexa

On veut maintenant afficher une variable qui contient un nombre binaire de 16 bits, affiché en hexadécimal. Si c'est du BCD, chaque groupe de 4 bits est inférieur ou égal à 9. Pour afficher le premier digit, il fait extraire les 4 premiers bits du nombre. On décale de 12 positions à droite, ce qui injecte des zéros dans les 12 premiers bits et génère le nombre de 0 à F pour la fonction CopyDigit
Pour afficher le digit suivant, on doit décaler de 8, etc.
Modifiez le programme précédent pour afficher une variable initialisée par exemple à 1234.



Pour afficher le nombre en appelant une boucle qui se fait 4 fois, ce qui permettrait de passer à un affichage 6 ou 8 digits en changeant un paramètre et en évitant les définitions SelDigxOn/Off. Un truc pour éviter un décalage de 12,8,4,0 est de décaler le nombre dans l'autre sens de 4 après chaque affichage.

```
void AfNombre (int nbre) {
    for (byte i=0; i<4; i++) {
       copyDigit (digit [nbre>>12] ;
       SelDigit (i) ;
       nombre <<=4 ;
    }
 }
```

Il nous faut aussi pour cette boucle une fonction de sélection qui dépende de i. Avec le câblage choisi, on voit que si i=0 on doit écrire un mot 1000xxxx dans le portD. si i=1, 0100xxxx, etc. Il ne faut pas modifier les bits xxxx.
On désélectionne en mettant les 4 sélections à zéro; inutile de trier.

```
void SelDigit (byte ii)  {
     PORTD |= 1<< (7-ii) ;
     delay (durAf);
     PORTD &= 0x0F);
  }
```

Pour afficher 1234, il suffit d'écrire `AfNombre (1234); (voir`
Un programme qui compte contient l'instruction `AfNombre (nbre);` et un délai pour que cela ne compte pas trop vite (charger `Af4Compte.ino`) . Ce délai doit compter les cycles d'affichage, et pas être le delay() qui bloque tout action pendant son exécution.
Avec les `#include` expliqués avec le Kidule Feux, le programme AfCompteInc.ino devient une base pour tester des programmes qui ont besoin d'un affichage 4 digits.

```
// Af4Compte  sans include
void setup () {
  DDRD &= 0b11110111 ;   // poussoir in
  DDRD |= 0b00000100 ;   // segment h
  DDRC |= 0b00111111;    // segments a..f
  pinMode (7,1); // probl Arduino
}
void KiWrite (int kk) {
   PORTC = kk;
   PORTD |= kk>>4;   // force les 1
   PORTD &= kk>>4;   // force les 0
 }
```

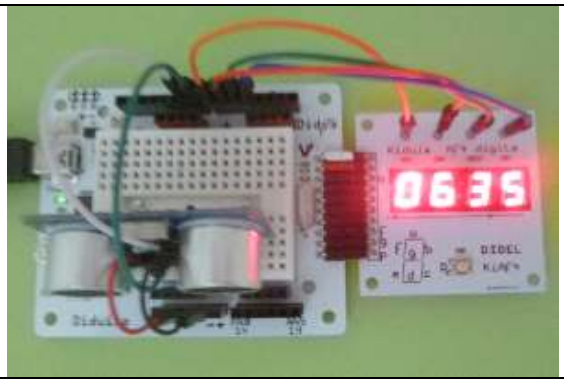
Note: le problème Arduino n'est pas systématique. Il semble qu'un timer soit initialisé par derrière et perturbe parfois le fonctionnement de cette pin 7.

```
byte digit[16]={0x3f,0x06,0x5B,0x4F,0x66,0x6D,0x7D,0x07,
        0x7F,0x67,0x77,0x7E,0x39,0x5E,0x79,0x71};
byte cnt = 0 ;

int durAf = 5 ;
void SelDigit (byte ii)  {
       PORTD |= 1<<(7-ii) ;
       delay (durAf) ;
       PORTD &= 0x0F ;   // désélectionne
 }
void AfNombre (int nbre) {
    for (byte i=0; i<4; i++) {
       KiWrite (digit [nbre>>12]) ;
       nbre <<=4 ;
       SelDigit (i);
    }
}
int nombre = 0x0000;
int vitComptage = 20 ;
void loop () {
   AfNombre (nombre) ;
   vitComptage--;
   if (vitComptage==0) {
      vitComptage = 20;
      nombre++;
   }
}
```

On peut faire une libraire avec ce qu'il faut pour afficher: voir Af4CompteInc.ino

Application: lire un capteur ultrason

Le capteur Ultrason est facile à utiliser si on accepte de ne rien faire d'autre pendant la lecture de la distance. Pout tous les détails, voir www.didel.com/xbot/DistSonar.pdf
Comme le montre les définitions, le SR04 ou SR05 est câblé sur les pins 8 et 9.
L'affichage se fait sur le terminal et sur le Kidule, ce qui permet de voir l'information du capteur quand le robot bouge.



```
//Af4Sonar.ino
#define Trig 8
#define Echo 9
void setup ()  {
  DDRD &= 0b11110111 ;  // poussoir en entrée
  DDRD |= 0b11110100 ;  // seg horiz g
  DDRC |= 0b00111111;  // seg a-f
  pinMode(Trig, OUTPUT);
  pinMode(Echo, INPUT);
  Serial.begin(9600);
}
void KiWrite (byte kk) {
   PORTC = kk;
   PORTD |= kk>>4;  // force les 1
   PORTD &= kk>>4;  // force les 0
 }

byte
digit[]={0x3f,0x06,0x5B,0x4F,0x66,0x6D,0x7D,0x07,
        0x7F,0x67,0x77,0x7C,0x39,0x5E,0x79,0x71};

int durAf = 5 ;
unsigned int nombre;
void SelDigit (byte ii)  {
       PORTD |= 1<<(7-ii) ;
       delay (durAf) ;
       PORTD &= 0x0F ;  // désélectionne
 }
```

```
void AfNombre (unsigned int nbre) {
    for (byte i=0; i<4; i++) {
       KiWrite (digit [nbre>>12]) ;
       nbre <<=4 ;
       SelDigit (i);
    }
}
int GetSonar () {
  int temp;
  digitalWrite(Trig,HIGH);
  delayMicroseconds(10);
  digitalWrite(Trig,LOW);
  temp= pulseIn(Echo,HIGH);
  return temp;
}

int dist;
byte durAff = 40 ; // duree aff entre sonar  env
1s 200/5
void loop () {
   dist=GetSonar();
   Serial.print(dist);
   Serial.print(" HEX ");Serial.println(dist,HEX);
   durAff = 0;
   while  (durAff<20) { // affiche 0.1 sec
     AfNombre (dist);
     durAff++;
   }
}
```

Avec des librairies, le programme est plus clair; inutile de s'encombrer dans chaque test de capteur avec des bouts de programmes bien connus!

```
//Af4SonarInc.ino
#include "LibAf4.h"
#define Trig 8
#define Echo 9
#include "LibSonarBloc.h"

void setup ()  {
  SetupAf4 ();
  SetupSonar ();
  pinMode(Trig, OUTPUT);
  pinMode(Echo, INPUT);
  Serial.begin(9600);
}

int dist;
byte durAff = 40 ;
void loop () {
  dist=GetSonar();
  Serial.print(dist);
  Serial.print(" HEX ");
  Serial.println(dist,HEX);
  durAff = 0;
  // affiche 0.1 sec + sonar
  while  (durAff<20) {
     AfNombre (dist);
     durAff++;
  }
}
```

```
//LibAf4.h
#include <Arduino.h>
void SetupAf4 () {
  DDRD  = 0b11110111 ;  // poussoir en entrée
  DDRD |= 0b00000100 ;  // led centre sortie
  DDRC |= 0b00111111;  // Leds 0-5
}
void KiWrite (int kk) {
   PORTC = kk;
   PORTD |= kk>>4;  // force les 1
   PORTD &= kk>>4;  // force les 0
 }
byte digit[]=
{0x3f,0x06,0x5B,0x4F,0x66,0x6D,0x7D,0x07,
 0x7F,0x67,0x77,0x7C,0x39,0x5E,0x79,0x71};
int durAf = 5 ;
void SelDigit (byte ii)  {
       PORTD |= 1<<(7-ii) ;
       delay (durAf) ;
       PORTD &= 0x0F ;  // désélectionne
 }
void AfNombre (unsigned int nbre) {
    for (byte i=0; i<4; i++) {
       KiWrite (digit [nbre>>12]) ;
       nbre <<=4 ;
       SelDigit (i);
    }
}
```

```
//LibSonarBloc.h  bloquant
50-200ms

void SetupSonar () {
  pinMode(Trig, OUTPUT);
  pinMode(Echo, INPUT);
}

int GetSonar () {
  int temp;
  digitalWrite(Trig,HIGH);
  delayMicroseconds(10);
  digitalWrite(Trig,LOW);
  temp= pulseIn(Echo,HIGH);
  return temp;
}
```

Poussoir

Le poussoir sur le kidule Af4Dig est actif à zéro. Il a des rebonds de contact.
Relisez www.didel.com/diduino/Composants02.pdf pour en savoir plus.