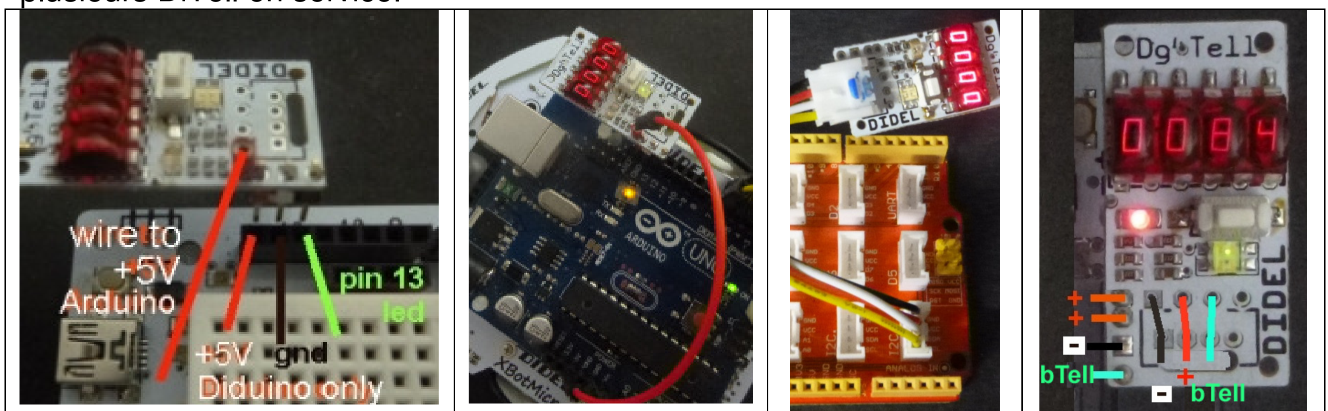
DiTell – affichage de nombres 16 bits

Utilisation de la pin 13 pour aider à la mise au point des programmes

Sérialiser de l'information sur une seule ligne impose des contraintes. DiTell est facile à utiliser si on comprend son fonctionnement. Il rend alors de grands services, et est utilisable sur tous les microcontrôleurs programmés en C.

Utilisation

DiTell est optimisé pour être positionné sur le connecteur B d'une carte compatible Arduino. Il utilise la pin Aref, rarement utilisée, qui doit être câblée au +5V (standard sur les cartes Diduino). Un connecteur Grove peut être installé à la place des 3 pins. Le signal bTell peut provenir d'une pin quelconque en modifiant les déclarations de câblage et on peut avoir plusieurs DiTell en service.



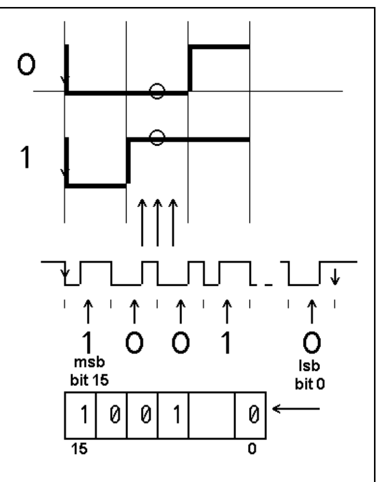
DiTell fonctionne de 3 à 5V et consomme moins de 40 mA.

Une action sur le poussoir converti le nombre en décimal, voir les différents modes sous http://www.didel.com/diduino/DiTell.pdf

TellI2C est programmé pour recevoir le mot de 16 bits par I2C; il affiche également des textes. ***PyTell*** pour Raspberry affiche en mode segment seulement et est associé à une librairie Python.

Principe

Pour transférer des bits en série sur une ligne, on utilise des impulsions de longueur variable. Le bus domotique "1-wire" de Dallas utilise le même principe avec des timings moins serrés. Si l'impulsion est courte, c'est un 1, si elle est 2 fois plus longue, c'est un zero. Le récepteur doit connaître cette durée `tt` de l'impulsion courte et il échantillonne le signal à1,5 fois cette durée, après avoir vu le début de l'impulsion. Si la marge d'erreur est respectée, le transfert sera fiable. L'erreur vient de l'émetteur, qui peut avoir un "jitter" et du récepteur qui ne réagit pas instantanément quand le signal passe à zéro.
Les bits du mot de 16 bits sont transmis poids fort en premier (msb=most significant bit) et sont poussés à la réception dans une variable (décalage à gauche).



Il ne suffit pas de compter les bits, car on peut en perdre suite à un parasite. Les mots transmis sont séparés par un espace et c'est la détection de cet espace qui remet à zéro le compteur de bits du récepteur.

La réception par DiTell est délicate à programmer. Le processeur (qui balaye l'affichage comme tâche principale) est interrompu à chaque transition de l'entrée.

Occupons-nous seulement de l'envoi de l'information.

Bloquant ou par interruption ?
Ce que l'on veut, c'est faire afficher une variable que l'on veut surveiller.
Quand? Sur demande ou en continu?
Sur demande, on appelle une fonction de transfert et le programme est "bloqué" pendant l'envoi des 16 bits, pour une durée de 3 ms..
L'interruption permet un transfert "transparent", en interrompant le programme toutes les 58 µs, mais moins de 5 µs chaque fois . Traitons séparément ces deux cas.

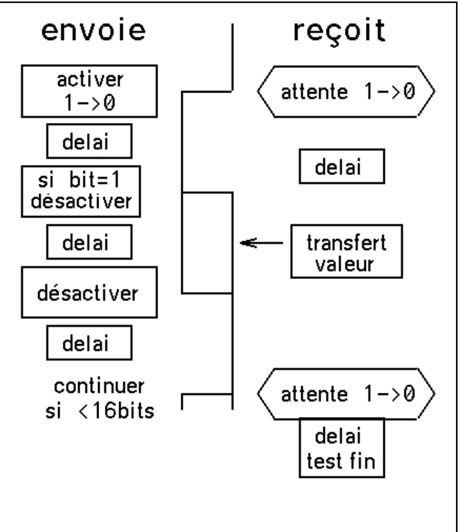
1ere partie Utilisation bloquante

Programme de test Arduino

La fonction d'envoi répète 16 fois le timing que l'on vient de voir. Pour une raison expliquée plus loin, la durée `tt` est de 58 microsecondes. Le programme "naif" que l'on peut écrire pour tester l'envoi suit l'organigramme ci-contre:

```
… déclaration
void loop {  // envoie la variable v16
  temp = v16;
  for (byte i=0;i<16;i++) {
      digitalWrite (13,0); delayMicroseconds (58);
      if ((temp & (1<<15))==0) { digitalWrite (13,1); }
      delayMicroseconds (58);
      digitalWrite (13,1); delayMicroseconds (58);
      temp <<=1;
  }
   delayMicroseconds (600);   // garanti la resynchronisation
   v16++; delay (100);  // on compte
}
```

Programme complet est sous http://www.didel.com/diduino/ DiTellFacile.zip



Ce programme fonctionne, mais la durée observée à l'oscilloscope est de 63 à 70 us au lieu de 58 et la taille est de 1180 bytes. Les fonctions Arduino prennent de la place et du temps. Des interruptions cachées ajoutent un jitter de 7 microsecondes. On est donc plus précis en remplaçant le 58 par 50 (trouvez le minimum et le maximum).

Ce programme a été testé sur un AVR328 à 16 MHz. Il doit être portable sur toutes les carte compatible Arduino et doit fonctionner avec toutes les librairies Arduino, à condition de désactiver les interruptions pendant le transfert Tell (voir plus loin).

Programmons en C

Les "facilités" Arduino empêchent l'utilisation de processeurs qui ont peu de pattes et peu de mémoires (Tiny25, 24). Ils ont les mêmes problèmes de mise au point, donc DiTell qui n'exige qu'une patte de libre et 100 bytes est très intéressant. Ecrire en C portable sur n'importe quel microcontrôleur est efficace et simple comme on va le voir.
Reprenons le programme "naif" avec les contraintes suivantes:

1) Il faut une fonction que l'on peut appeler quand on veut afficher
2) On peut être amené à utiliser une autre pin, à utiliser 2 affichages. La fonction ne doit pas contenir de référence liée au câblage. C'est le rôle de définitions initiales.
3) Il ne faut pas utiliser les fonctions Arduino, ce sera plus efficace et on pourra passer le programme sur n'importe quel processeur programmé en C.

Fonction

Créer une fonction Tell (): n'a pas besoin d'être réexpliqué. Tell (v16); affiche la variable globale v16. Dans la fonction, on déclare une variable locale cc pour le compteur de bytest et une variable dd pour décaler et tester les bits l'un après l'autre.

Délai programmé

Un délai de 58 microsecondes se fait par une boucle d'attente. Un oscilloscope permet de calibrer car on ne connait pas les instructions générées par le compilateur (sans oscilloscope, on appelle 100000 fois la boucle et on mesure le temps avec sa montre).
Deux lignes de C génèrent notre attente de 58 µs

```
byte qz=95;
while (qz--){}  // ~58 µs avec un AVR à 16 MHz
```

Direction de la pin (pinMode)
Les pins Arduino sont sur un bit d'un port. La pins 13 est sur le PORTB et le regisre DDRB associé définit si la ligne est en entrée ou sortie. On définit le no de pin, et écrit l'instruction d'initialisation pour le set-up.

```
#define bTell 5 // pin 13 Arduino PORTB pin 5
#define bTellOut bitSet(DDRB,bTell)
```

Ceci permet d'utiliser par la suite l'instruction `bTellOut;` dans le setup

Etat de la pin (digitalWrite)
Le bit bTell du port choisi est actif à zéro et on définit

```
#define TellOn  bitClear (PORTB,bTell)
#define TellOff  bitSet  (PORTB,bTell)
```

On pourrait écrire à la place `PORTB &= ~(1<<bTell)` et `PORTB |= (1<<bTell)`.

Boucle `for`
Un while est plus rapide et utilise moins d'instruction que le for. On va donc écrire la boucle répétée 16 fois de la façon équivalente suivante:

```
byte cc=0;
while (cc++ < 16) {
   . . .
}
```

Jitter d'interruption
Pour avoir le delay() et delayMicroseconds(), Arduino met et route une interruption qui crée un jitter acceptable de 5 µs. Le PWM, les servos, des librairies, mettent en service d'autres interruptions plus longues qui peuvent créer des affichages erronés. Il faut couper les interruption pour la durée de 3 µs du transfert Tell.

```
cli;   // coupe les interruption (clear interrupt)
   . . .
sei;   // rétabli les interruption (set interrupt)
```

Nouveau programme

```
//TellFonction.ino
#define bTell 5 // pin 13 Arduino PORTB pin 5
#define TellOn  bitClear (PORTB,bTell)
#define TellOff  bitSet  (PORTB,bTell)
#define bTellOut bitSet  (DDRB,bTell)

volatile byte qz;
volatile int qqz;
#define Delt qz=95; while (qz--) {}  // 60us à 16 MHz
#define Deltt  qqz=600; while (qqz--) {}  // 600us à 16 MHz

void Tell (int dd) {
  byte cc=0;
  //  cli();  //remove jitter interrupt if required (>5us)
  while (cc++ < 16) {
    TellOn; Delt;
    if (!(dd&0x8000)) { TellOff;}
    Delt;
    TellOff; Delt;
    dd <<=1;
  }
  //  sei();  //restore interrupts
  Deltt;
}
void setup() {
  bTellOut;
  TellOff; // high
}
int cnt = 0x1234;

void loop() {
  Tell(cnt++);
  delay(100);
}
```

Comparaison
Le programme "naif" utilise 1180 bytes.
Le programme ci-dessus 750 bytes et 160 bytes peuvent être économisé si on n'utilise pas la fonction Arduino delay(). Arduino utilise 450 bytes pour un programme vide.
La fonction Tell(); seule ne prend que 92 bytes (voir http://www.didel.com/diduino/DiTell.pdf).

Dernière étape: Tell.h
On va utiliser la fonction Tell dans plusieurs programme, Il faut en faire une librairie qui rendra le programme plus lisible. Tout ce qui réalise une fonction claire et bien comprise est encapsulée comme une librairie personnelle, avec un #include "nom.h", à ne pas confondre avec les librairies Arduino qui s'écrivent avecun #include <nom.h>
Le fichier Tell.h contient les 3 composants à mettre en œuvre dans toute librairie personnelle:
- les déclarations qui disent comment c'est câblé
- la fonction a effectuer dans le setup pour initialisr les ports et prédéfinir éventuellement des variables
- la ou les fonctions de librairie

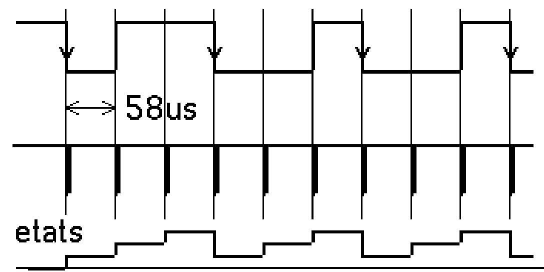
Donc Tell.h a l'allure suivante (les instructions bien connues ne sont pas répétées)
```
#include <Arduino.h>  // byte n'est pas reconnu autrement

#define bTell 5 // pin 13 Arduino PORTB pin 5
#define TellOn  bitClear (PORTB,bTell)
#define TellOff  bitSet  (PORTB,bTell)
#define bTellOut bitSet  (DDRB,bTell)

volatile byte qz;
volatile int qqz;
#define Delt qz=95; while (qz--) {}  // 60us à 16 MHz
#define Deltt  qqz=600; while (qqz--) {}  // 600us à 16 MHz

void SetupTell() {
  bTellOut;
  TellOff; // high
}

void Tell (int dd) {
  byte cc=0;
. . .
  Deltt;
}
```
Le programme principal exprime maintenant seulement ce que l'on veut faire:
```
//TellInclude.ino
#include "Tell.h"
void setup(){
  SetupTell();
}
int cnt = 0x1234;
void loop() {
  Tell(cnt++);
  delay(100);
}
```

Nouveau programme avec Tell.h
Vous écrivez un nouveau programme avec des `Tell(var);` quand cela vous arrange.
Le programme commence par
```
//titre.ino
#include "Tell.h"
. . . autres #include
void setup(){
  SetupTell();
  . . . autres initialisations
}
Vos définition, fonctions, setup et la boucle
```

Le problème est d'avoir Tell dans le dossier titre qui contient Titre.ino et doit aussi contenir Tell.ino (on ne peut pas le mettre à un endroit central comme pour les librairies < >).
Sous "Croquis" sélectionner "Ajouter un fichier", trouver un `Tell.h` dans un autre programme ou dans le dossier de vos librairies, et il fait partie maintenant de votre programme.

2e partie Utilisation transparente par interruption 58 µs

Couper le transfert en tranches

Le transfer Tell implique une décision de modification toutes les 58 µs. Cette décision, qui n'utilise que 2-3 instructions peut se prendre dans une interruption qui toutes les 58 microsecondes, coupe le programme principal et avance dans la séquence des états de transfert.	

Le Delt du programme bloquant est remplacé par un retour au programme principal et rappel après 58 us.
Une machine d'état DoTell() est appelée, qui affiche en permanence une variable 16 bits appelée `highLow`.
La mise en service de l'interruption est plus compliquées, la fonction DoTell décomposée en états est plus délicate à comprendre, mais ceci se met en place à l'initialisation et est ensuite facile à utiliser: **au lieu de `DiTell(v16);` on écrit `highLow=v16;`**

Machine d'état

Le transfert dure 3ms et est déclenché par un flag qu'il suffit d'activer toutes les 20 à 50ms. De plus, l'état de la pin bTell est sauvé et rétabli. Il est possible de clignoter la Led 13 tout en transférant, mais la led est moins intense puisque coupée sans cesse de quelques microsecondes..

```
enum {AtS1,DebS1,StartS1,BitS1,ContS1,StopS1} eS1 = AtS1;
  volatile unsigned int hL;
  volatile byte cc, etatLed;
void DoTell () {   // appel toutes les 58 us
  switch (eS1) {
    case AtS1:  // on attend le flag de départ
      if (flagS1) { hL=highLow;  eS1 = DebS1;}
      break;
   case DebS1: // on sauv LED, pin 13 à 1
      if (!S1In) {etatLed=0; S1Off; }  else {etatLed=1;}
      cc=Nbits;  eS1 = StartS1;
      break;
   case StartS1:  //envoie bit
      S1On; eS1=BitS1;
      break;
   case BitS1: // coupe si bit=0
     if (!(hL&0x8000)) S1Off;
     eS1 = ContS1;
     break;
   case ContS1:
      S1Off; hL <<=1;
      if (cc--) { eS1 = StartS1; }
      else  { eS1 = StopS1; }
      break;
    case StopS1:
      if (!etatLed) { S1On; }
      flagS1=0; eS1 = AtS1;
      break;
  } // end switch
}
```

Interruption par le Timer2

```
// Inter.h tous les inter
volatile byte cnt0,cnt1,cnt2;
volatile byte e6ms;
ISR (TIMER2_OVF_vect) {
 TCNT2 = -55;  //  60 us calibre
 DoTell();
}

void SetupInter() {  // initialisation
   cli();
```

```
    TCCR2A = 0; //default
    TCCR2B = 0b00000010;  // clk/8
    TIMSK2 = 0b00000001;  // TOIE2
    sei();
  }
```

Programme de test

Sur l'écran Arduino, on voit les 4 onglets .ino et .h

```
// TellLibX.ino  998 bytes
// variables globales
// highLow défini dans DoTell.h

#include "Robot.h"
#include "DoTell.h"
#include "Inter.h"

void setup() {  // initialisation
  SetupRobot();
  SetupTell();
  SetupInter();
}
void loop() {
  highLow++;
  delay(100);
}
---------------------------
// Robot.h
#include <Arduino.h>  //<avrio.h>
#define bLed 5 // bit 5 PORTB (pin 13)
#define  LedOn  bitSet   (PORTB,bLed)
#define  LedOff bitClear (PORTB,bLed)
#define  LedToggle PORTB ^= (1<<bLed)

void SetupRobot() {  // initialisation
PORTD 23 in 4567 out
  DDRD |=  0b11110000 ;
  DDRD &=  0b11110011 ;
  DDRB = 1<<bLed ; // Led13 et DiTell
}
---------------------------
// Inter.h
volatile byte cnt1, e6ms;
ISR (TIMER2_OVF_vect) {
 TCNT2 = 141;  //  58 us (256 = 125us
   DoTell();
   if (cnt1++ > 175) { cnt1=0;  //
toutes les 175x58= 10ms
     flagS1=1;   // lance le transfer
   }
}

void SetupInter() {  // initialisation
  cli();
  TCCR2A = 0;
  TCCR2B = 0b00000010;  // clk/8
  TIMSK2 = 0b00000001;  // TOIE2
  sei();
}
```

```
// DoTell.h
volatile int highLow;
byte flagS1;  // envoi toutes les 20ms

#define bTell 5 // PORTB pin 13
#define TellOff    bitSet (PORTB,bTell)
#define TellOn   bitClear (PORTB,bTell)
#define TellIn  (PORTB & (1<<bTell)) // avec
Tell il ne faut pas lire la pin
#define TelldirOut  bitSet (DDRB,bTell)

void SetupTell() {
  TelldirOut; TellOff;
}
  volatile unsigned int hL;
  volatile byte cc, etatLed;
#define Nbits 16
enum {AtS1,DebS1,StartS1,BitS1,ContS1,StopS1}
eS1 = AtS1;
void DoTell () {   // appel toutes les 60 us
  switch (eS1) {
    case AtS1:  // on attend le flag de
départ
      if (flagS1) {  hL=highLow;  eS1 =
DebS1;}
      break;
   case DebS1: // on sauv LED, pin 13 à 1
      if (!TellIn) {etatLed=0; TellOff; }
else {etatLed=1;}
      cc=Nbits;  eS1 = StartS1;
      break;
   case StartS1:  //envoie bit
      TellOn; eS1=BitS1;
      break;
   case BitS1: // coupe si bit=0
     if (!(hL&0x8000)) TellOff;
     eS1 = ContS1;
     break;
   case ContS1:
      TellOff; hL <<=1;
      if (cc--) { eS1 = StartS1; }
      else  { eS1 = StopS1; }
      break;
    case StopS1:
      if (!etatLed) { TellOn; }
      flagS1=0; eS1 = AtS1;
      break;
  } // end switch
}
```

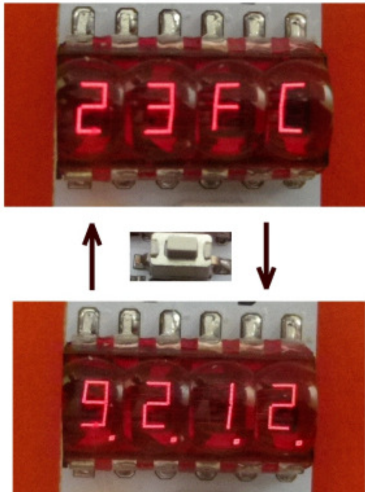
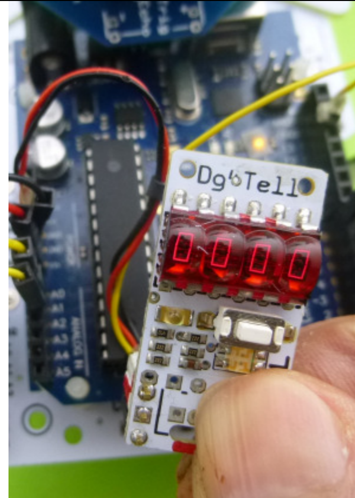
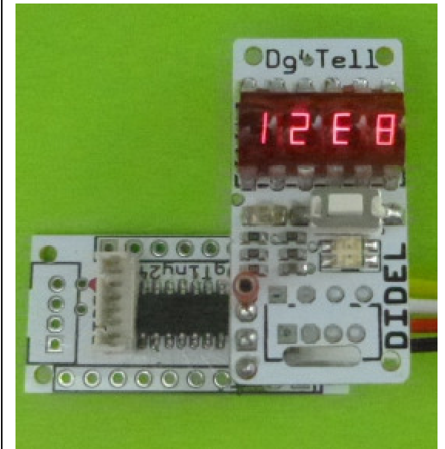
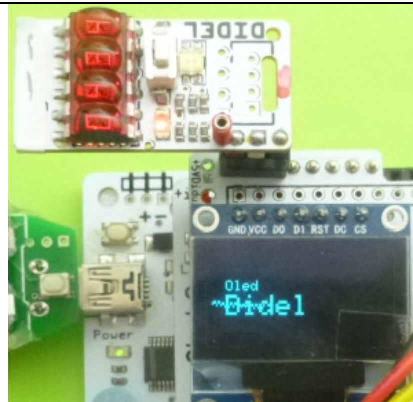
Librairie LibX

La librairie LibX, facilement portable sur tout microcontrôleur programmable en C est documentée sous www.didel.com/xbot/LibX.pdf

Elle offre un choix de librairies similaires à DoTell.h pour lire des capteurs (ultrason, IR, analogue, encodeur) et commander des actionneurs (PFM, Servo, PasAPas).

Le capteur ultason a fixé la période de 58 us: 1 cm de distance correspond à une unité de comptage dans la machine d'état.

Chaque tâche ajoutée dans la routine d'interruption prend 4-8 microsecondes toutes les 58 us. Si on arrive à 30 us, le programme principal "voit" un processeur à 8 MHz au lieu de 16, sans longues coupures comme avec les interruptions traditionnelles.

Galerie		
		
Le poussoir passe pare les modes hexa et décimal	Ditel se branche avec 3 fils	DiTell a un connecteur prévu sur le Digrove Tiny24
		
Ditell avec affichage normal.	Ditell a sa place sur l'adaptateur pour Oled 64x96	TellI2C affiche des textes

Autre modèles
DgTell (TellI2C) avec connecteur Grove compatible I2C/SMbus. 8 commandes pour les différents modes d’affichage hexa, décimal, Ascii, segment.
http://www.didel.com/digrove/DgTelli2c.pdf
PyTell sous-ensemble de DgTell pour librairie Python (mode segment).
Sur les 3 modèles, l’affichage miniature, plus disponible, sera remplacé par un affichage qui débordera du circuit imprimé.