Pythie – Aide au développement des cartes AVR328

Attention à la confusion Rx/Tx. Le signal qui arrive sur PD0 est appelé ici Rx
Les cartes DuiXx, nommées avec des noms de la mythologie grecque, sont programmées via un connecteur 5 pins avec les signaux Gnd, 5V, Tx, Rx, Rz liées à un adaptateur USB. Quand la prise USB n'est pas connectée, les 3 signaux sur ce connecteur peuvent aider au déverminage. Le module Gaia est le plus simple possible, voir www.didel.com/Gaia.pdf .
Gaia a donné son nom au connecteur 5 pins de programmation (connecteur et signaux Gaia).

Pythie ajoute une Led et poussoir sur le signal Rx, un poussoir et un interrupteur sur le signal reset, un connecteur d'alim pour un crayon logique, un connecteur au pas de 2.54 pour transmettre les signaux Rx, Tx au module Janus et un connecter pour un affichage Tell.
Détaillons toutes les possibilités (documentées avec des schémas et programmes de test dans www.didel.com/PythieJanus.pdf) :

Reset

Un poussoir et interrupteur sont câblés sur Tx et Rx. Un poussoir *reset* et un interrupteur permet de couper l'activité en cas de panique (au lieu de couper l'alimentation). Il faut parfois procéder en deux temps : presser le *reset*, et quand la réinitialisation est faite et que tout est calme, fermer l'interrupteur *reset* avant que le programme déraille, jusqu'à ce que le programme corrigé puisse être chargé.

PD0- Led

La pin PD0/Rx est reliée à une LED. La Led est allumée au démarrage et clignote au début des chargements.
On peut clignoter cette Led en désactivant le UART interne et en considérant PD0 comme une pin ordinaire.

PD1- Push

La pin PD1/Tx est reliée à un poussoir vers le Gnd. Une résistance de 220 Ohm limite le courant si on presse quand Tx est programmé en sortie. On peut lire ce poussoir en désactivant le UART interne et en considérant PD1 comme une pin ordinaire.
Le programme de test est **PythiePushLed.ino**

PD1-Tell

La pin PD1/Tx du 328 peut être utilisée pour mesurer à l'oscilloscope des durées d'exécution, synchroniser l'oscilloscope ou vérifier que l'on passe par certains points du programme. Le connecteur est compatible avec l'affichage Tell pour afficher 16 bits en 3ms. Le programme de test est **PythieTell.ino**

PD1-Tx Terminal sur tablette

PD1/Tx peut aussi être utilisé pour transmettre des données vers une tablette ou son propre écran. Les "Serial.print", absurdes, sont remplacés par la libraire TerSer.h qui a entre autre l'avantage d'imprimer les nombres avec leurs zéros non significatifs. Le programme de test est **PythieTerSer.ino**.

PD0-S0 sync PD1-Tx

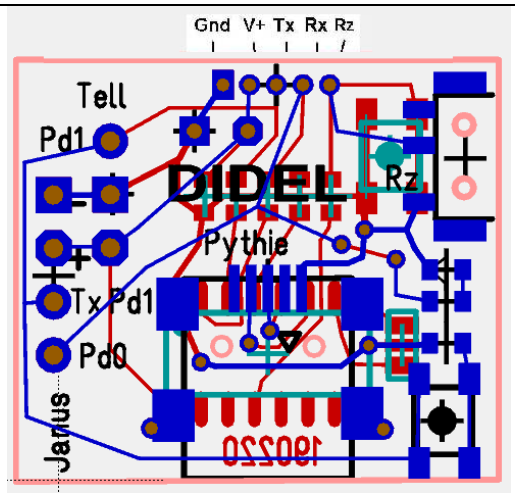
Tx est utilisé pour transmettre en série à 1 Mbits/s vers Janus et afficher des variables sur oscilloscope en synchronisant avec S0.

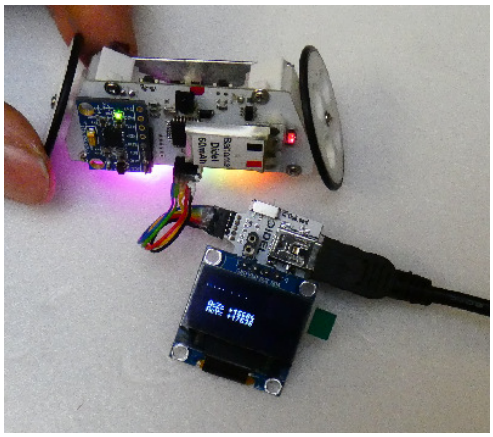
PD0-S0 sync PD1-S1

Pour dépanner à l'oscilloscope, PD0 et PD1 sont programmés en sorties et utilisés pour synchroniser l'oscilloscope et mesure des durées. programme Arduino. Le programme de test est **PythiePushLed**.

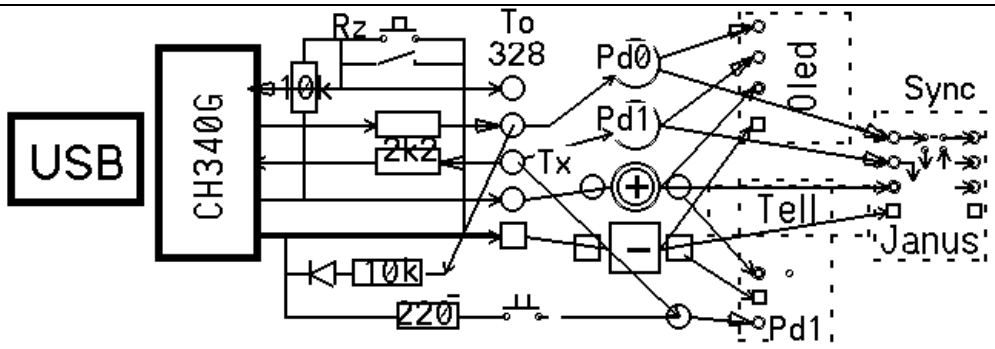
PD0=Scl PD1=Sda pour Oled

Un Oled peut être pratique dans certains cas. Il affiche plusieurs variables et des graphiques simples en 50ms.
Le programme de test est **PythieOled.ino**



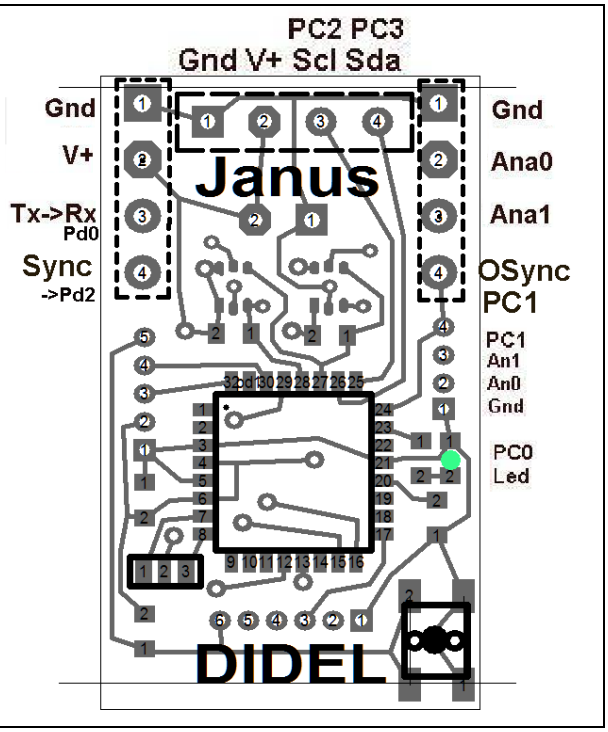


Ce schéma-bloc donne le schéma et les options de câblage.



Janus

En connectant le module Janus on a la possibilité de visualiser des signaux qui évoluent rapidement.
Janus joue le rôle d'un oscilloscope à mémoire pour afficher en analogique deux variables échantillonnées à 40 kHz au plus.
A défaut d'un oscilloscope, l'écran Oled de Janus peut être utilisé, avec l'avantage par rapport à l'écran sur Pythie, qu'il suffit de 20 µs pour mettre à jour deux variables. Voir www.didel.com/Janus.pdf .
Janus peut avoir d'autres utilisations indépendantes avec son écran Oled, deux entrées analogiques et deux sorties analogiques.
Le connecteur de programmation donne accès à SPI (PB3,4,5).



En comparaisons au traceur série Arduino (Graph), la perturbation temporelle est nettement plus faible et l'affichage est cohérent.

Programmes de test et librairies

Il faut bien comprendre le setup pour agir directement sur Pd0 (Rx) et Pd1(Tx).
Dans UART.h, on voit que pour activer Rx et Tx, il faut les 2 lignes

```
UCSR0B |=0x10;//  le bit 4 active Rx
UCSR0B |=0x08;//  le bit 3 active Tx
```

Ces bits sont activés par le chargeur. Il faut les désactiver dans le setup du programme.

```
UCSR0B =0;

UCSR0B &=~0x10;// force le bit 4 à 0 et désactive Rx
UCSR0B |=0x08;//  met à un le bit 3 active Tx
```

On peut ensuite configurer Pd0 et Pd1 en entrée ou sortie, comme on veut.

Par exemple pour avoir Tell sur Pd1 on désactive Rx, on active Pd0 en sortie et on peut garder Tx=Pd1 pour envoyer de l'info sur un terminal ou Janus branché sur la pin3 du connecteur Janus.

Pour y voir clair, il faut des définitions claires. Pd0 est la pin arduino 0, le bit 0 du port D.
On désactive
Tell doit être initialisé en sortie, et utilise les instructions TellOn (
`PythiePushLed.ino`
3 clignotement Tx à chaque pression sur le poussoir Rx
TestPythieS0S1.ino et PythieS0S1.h
Pas fait

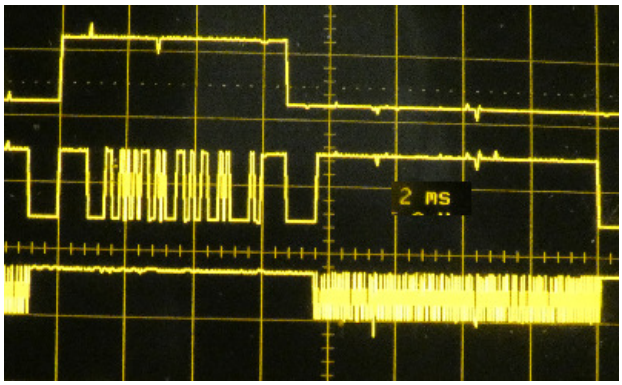
PythieTell.ino *utilise PyTell.h*
Teste PyTell.h adapté pour connecteur Tell
PythieTerSer.ino
A faire – exemple appel TerSer
PythieOled.ino utilise *Oled.h (variante à définir)*
Affichage sur SSD1301, USB déconnecté
PythieTerOled.ino utilise *Oled.h et TerOled.h*
A faire - Affichage sur SSD1301, USB déconnecté

Applications avec Janus documentées sous Janus.pdf

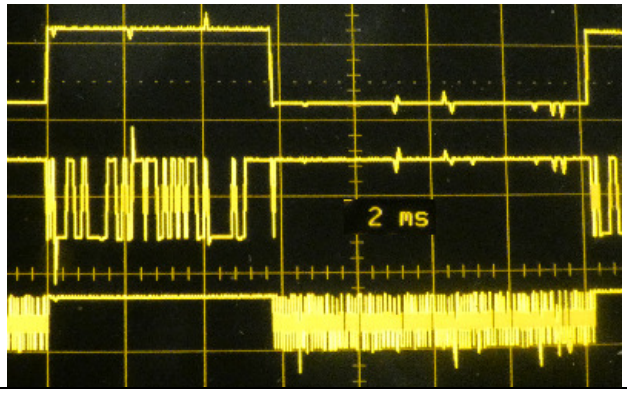
Comment utiliser ces programmes et librairies *– a vérifier jd 191008*
Un programme est en général écrit en oubliant que l'on aura des problèmes
Fonctions déclarées dans les librairies
S1On; S1Off; S1Toggle; pin Tx – ne pas charger Serial.begin.
S2On; S2Off; S2Toggle;
PyTell(v16);
Les appels au terminal série doivent être parenthésés si on veut utiliser S1On/Off. On ne peut pas utiliser S1 comme signal de test dans les zones ou on affiche.
Note: Les appels sont les mêmes que avec OledPix, mais il a fallu changer les noms.
La librairie Ternum.h demande un SetupTerNum(); et n'a pas besoin du TerOn; TerOff;
TerOn; Autorise les transferts série, 9600 bit/s
TerOff; Désactive et et autorise S1On/Off
TerCar(v8); Envoie un byte, usuellement en Ascii ('a' ou 0x41 par exemple).
TerBin8(v8); *TerSer seulement:* TerBin16(v16);
TerHex8(v8); TerHex16(v16); TerHex32(v32);
TerDec8(v8); TerDec16(v16); TerDec32(v32); 11 digits???
TerText("bla");
CR comment faire??
Exemple;

```
//TestPegase.ino
#include "Pegase.h"
#include "OledPix.h"
#include "Pythie.h"
void setup() {
  SetupPegase();
  SetupOledPix();
  SetupPythie();
int var;
void loop() {
  S1On;  // Tx
*  DelMs(1);
  LiCol(2,20); Hex16(var);
//  PyTell(var);
*  DelMs(1);
  S1Off;
     DelMs(1);
   S2On; // Rx
*  DelMs(1);
   TerOn;
*   DelMs(1);
  TerHex16(var);
   TerOff; // 3ms
   S2Off;
  var++;
*  DelMs(1);
}
```

Avec délais



Sans délais



Debug sur RxTx

```
// PythieDebug.h  S1 et Tell sur Tx Pythie  S2 sur Rx
// #include "PythieDebug.h"  SetupPythieDebug();
#define bTell 1 // Tx
#define TellOn  bitClear (PORTD,bTell)
```

```
#define TellOff  bitSet  (PORTD,bTell)
#define TelldirOut bitSet  (DDRD,bTell)
#define bS1 1 // Tx
#define bS2 0 // Rx
#define S1On  bitSet (DDRD,bS1);  bitSet (PORTD,bS1)
#define S1Off bitSet (DDRD,bS1); bitClear (PORTD,bS1)
#define S1Toggle PORTD ^= (1<<bS1)
#define S2On  bitSet (DDRD,bS2);  bitSet (PORTD,bS2)
#define S2Off bitSet (DDRD,bS2);  bitClear (PORTD,bS2)
#define S2Toggle PORTD ^= (1<<bS2)

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 SetupPythieDebug() {
  DDRD  |= 0b00000011;
  PORTD |= 0b00000011;
}

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;
}

/*
 * Pour mettre un Oled sur le connecteur, créer OledPiBb10
 * #define Ddr  DDRD
#define Port PORTD
#define Pin  PIND
#define bCk 0  // Oled128x
#define bDa 1  // pin
#define Ck1  bitClear (Ddr,bCk)
#define Ck0  bitSet (Ddr,bCk)
#define Da1  bitClear (Ddr,bDa)
#define Da0  bitSet (Ddr,bDa)

void SetupOledPixBb01() { //Avec OledPix
  Ddr  |=  (1<<bCk|1<<bDa) ;
  Port &= ~(1<<bCk|1<<bDa) ;
}
 */
```

Terminal série

```
//TestTerSerNum.ino  Terminal série et aff nombres
// 3034 avec oled  846 sans
#include "Pegase.h"
#include "OledPix.h"
#include "Tell.h"
#include "TerSer.h"
#include "TerNum.h"
#include "PythieDebug.h"
void setup() {
  SetupPegase();
  SetupOledPix();
  SetupTell();
  SetupTerSer();
  SetupPythieDebug();
 // LiCol(0,0);  Sprite(smile);  // top
 // LiCol(1,20); BigText("TestTerSer");

}
```

Petit tutorial pour bien comprendre le poussoir et la Led sur Pythie (et Witty)
La Led est sur la pin 0, le poussoir sur la pin 1. Pythie port D, Pégase port C

Il faut définit les actions des base

`#define bLed 0 // LED/Push sur Rx/PD0`
`#define  LedOn    bitSet (PORTD,bLed)`
`#define  LedOff  bitClear (PORTD,bLed)`
`#define  LedToggle (PORTD^=(1<<bLed))`
	En Arduino, on airait
	`#define pinLed 0 // LED/Push sur Rx/PD0`
	`#define  LedOn    pinMode`
	`#define  LedOff`
	`#define  LedToggle`

`#define bPush 1  // actif à zero` `#define  IsPushOn   (!(PIND&(1<<bPush)))`	`#define pinPush 1  // actif à zero` `#define  IsPushOn`

Un premier test doit vérifier ces définitions

```
//Test0.ino
.. lesdéfinitions
void setup() {
   bitSet  (DDRD,bLed);
   bitClear(DDRD,bPush);
}
void loop() {   // echo
   if(IsPushOn){LedOn;}
   else {LedOff; }
}
```

On peut maintenant utiliser la fonction `Cli(nfois,per);` qui clignote `n` fois avec la période `per.`
LedToggle est très pratique, mais il faut se souvenir qu'il faut deux demi-périodes pour une impulsion.

```
#define DelMs delay (le DelMs() de LibX donne les mêmes délais que le delay() Arduino)
void Cli(byte nn, int dd) {
  for(byte i=0;i<2*nn;i++){LedToggle; DelMs(dd);}
}
```

On veut maintenant que chaque fois que l'on presse, on lance 3 clignotements. De plus, pour savoir que c'est le moment de presser, on clignote rapidement.
Il faut donc, tant que l'on ne presse pas, clignoter rapidement, et lancer les 3 clignotements quand on relâche.
Cela devient intéressant si on se pose la question. Et si je represse rapidement, j'aurai 6 impulsions? Cela pourrait se programmer, mais il faut bien maîtriser, parce que le processeur ne sait faire qu'une chose à la fois..
Pendant les 3 clignotement, on ne lit pas le poussoir. Que veut-on si on a déjà repressé? Relancer 3 impulsions? Attendre que l'on ait relâché? Le programme correspond à cette 2e situation. A noter que pour clignoter en continu,

```
void loop() {
     while (!IsPushOn) {LedToggle; DelMs(50);}
     Cli (3,300);
     while (IsPushOn) ;
}
```

Ce programme a un problème que l'on ne remarque pas parce que les impulsions ont des périodes très différentes. Quand on relâche le poussoir, la Led est allumée une fois sur deux. Ensuite on demande d'allumer une première fois. Cette première fois n'est pas isolée; on la reconnaît parce qu'elle est plus longue.
Donc. Pour programmer correctement du point de vue interface homme-machine, il faut éteindre la Led et attendre un peu.

```
void loop() {
     while (!IsPushOn) {LedToggle; DelMs(50);}
     LedOff; DelMs(500);
     Cli (3,300);
     while (IsPushOn) ;
     DelMs(500);
}
```

Avec ces délais, on peut estimer que le `while (IsPushOn); n'est plus indispensable.`

La fonction GetPush()
Elle fait un peu la même chose, mais on presse plusieurs fois et la fonction rend ce nombre de fois. Pour que cela soit utilisable par la totalité des personnes "normales", éventuellement après quelques essais-erreur, il faut savoir ou sont les temps d'attente qu'il a fallu mettre dans le programme. Ces constantes doivent être déclarées et commentées en début de programme.

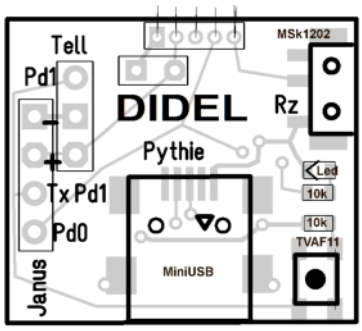
```
GetPush est expliqué sous
GetPush utilise  LedOn; LedOff; IsPusOn; et DelMs; comme vu précédemment.
Le programme de test est
// TestGetPush.ino
.. définitions LedOn etc
.. fonctions DelMs() et Cli(nn,dd)
```

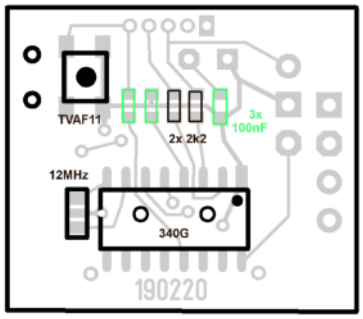
```
#include "GetPush.h"
void setup() {
.. direction pour les pins Led et Push
}
byte nPush;
void loop() { // blink and wait for a push
  nPush=GetPush();
  Cli(nPush,400);
}
```

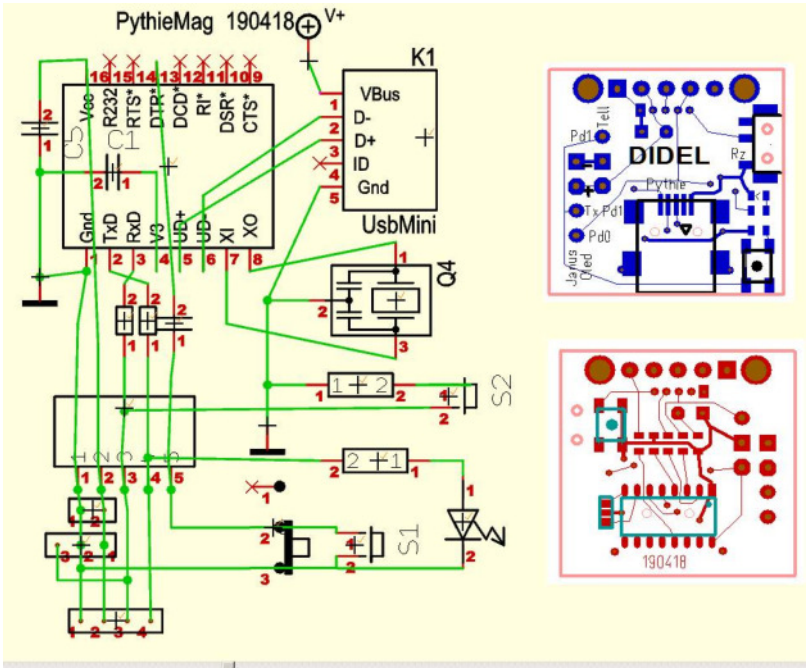
Avec ce programme, on vérifie que la fonction GetPush() s'est terminé en clignotant le bon nombre de fois. On vérifie encore que le paramètre rendu est correct.
Cligno(..) ne sera plus nécessaire. On utilise en général nPush dans un Switch case pour choisir une démo, sélectionner parmi des options et paramètres.

Jdn 190705

Complément provisoire







Nouveau en cours de documentation :
Shield ArduiPy avec connecteur pour le module Pythie, offrant quelques possibilités supplémentaires.

200206