

III - LE PROGRAMME SUR ARDUINO DUE

Ce document se trouve sur ce site : http://www.la-tour.info/uts/uts_page15.html#conduite

Si vous réaliser des améliorations matériels ou logicielles, merci de les poster sur le forum RMF pour en faire profiter le plus grand nombre.

Si vous publier cette réalisation avec ou sans améliorations, vous devez publier votre code source.

Ce logiciel est un logiciel libre. Exigence du concepteur : Ne pas modifier la ligne d'affichage "JLF xx/xx/xxxx" sur l'écran LCD.

On peut modifier ce programme et le diffuser. Dans ce cas, il faut préciser l'origine et donner accès aux sources modifiées.

Définition : Un logiciel libre est un logiciel distribué avec l'intégralité de ses programmes-sources, afin que l'ensemble des utilisateurs qui l'emploient, puissent l'enrichir et le redistribuer à leur tour.

Note : Un logiciel libre n'est pas nécessairement gratuit et les droits de la chaîne des auteurs sont préservés.

Équivalent étranger : free software, open source software.

(Source : Vocabulaire de l'informatique (liste de termes, expressions et définitions adoptés), NOR: CTNX0710138K, J.O n° 93 du 20 avril 2007 page 7078, texte n° 84)

logiciel libre

Par logiciel libre on entend un logiciel qui offre la liberté aux utilisateurs d'exécuter, de copier, de distribuer, d'étudier, de modifier et d'améliorer le logiciel. Plus précisément, elle fait référence à quatre types de liberté pour les utilisateurs du logiciel :

La liberté d'exécuter le programme, pour tous les usages (liberté 0).

La liberté d'étudier comment le programme fonctionne et de l'adapter à ses besoins (liberté 1). L'accès au code source est une condition requise.

La liberté de redistribuer des copies, (liberté 2).

La liberté d'améliorer le programme et de diffuser les améliorations au public pour en faire profiter toute la communauté (liberté 3). L'accès au code source est une condition requise.

Généralités

Ce montage peut être adapté à n'importe quel simulateur sur ordinateur, du moment que l'Arduino puisse recevoir et envoyer des informations à l'ordinateur.

La plaque de montage de l'Arduino DUE est compatible.

Le programme fonctionne avec le logiciel "Train Simulator Classic RailWorks de DTG", mais il est adaptable à d'autres environnements.

Les périphériques I2C fonctionnent à une fréquence standard de 100 kHz.

Si on ajoute plus d'entrées/sorties, ou si l'on veut aller plus vite, on peut essayer de passer en mode Fast à 400 kHz.

La connexion série à l'ordinateur est à **38400** bps. Si l'on constate des problèmes on peut passer à 19200 ou 9600 bps.

Mon logiciel se compose de trois éléments :

- | | |
|--------------------------------|------------------------------------------------------------------------|
| • PDC_Arduino_JLF.ino | Programme principal. Il est compatible avec toutes les configurations. |
| • Data_Recues_du_PC_TSCRJI.ino | Programme de réception des données. |
| • Data_Recues_du_PC_TSCRJI.ino | Programme d'envoi des données. |

Le code existe en plusieurs versions, adapté à chaque locomotive. Pour le moment, c'est décliné pour une locomotive.

- Les CC7200 de SimExpress, répertoire : **PDC_Arduino_JLF_CC72000_SE**

Pour adapter ce programme à d'autres locomotives ou à d'autres simulateurs, il faut modifier les programmes "Data_Recues_du_PC_TSCRJI_xx.ino" et "Data_Envoyees_vers_PC_TSCRJI_xx.ino" qui font l'interface avec l'ordinateur.

Le programme principal "PDC_Arduino_JLF_xx.ino" assure la couche matérielle.

Il met à jour les tableaux "i2c_inX_val[]" qui reflètent l'état des entrées.

Et quand l'on renseigne les tableaux "i2c_outX_val[]", il active les sorties en conséquence.

Il n'a pas besoin d'être modifié, et il est commun pour toutes les locomotives.

Dans le répertoire "...\\PDC_Arduino_JLF_xx", on doit avoir les fichiers :

- PDC_Arduino_JLF_xx.ino
- Data_Recues_du_PC_TSCRJI_xx.ino
- Data_Recues_du_PC_TSCRJI_xx.ino
- LiquidCrystal_I2C_W1.cpp
- LiquidCrystal_I2C_W1.h

Pour lancer l'environnement IDE Arduino, cliquer sur "PDC_Arduino_JLF_xx.ino".

Ajouter les librairies suivantes de Rob Tillaart et pour le joystick, par le gestionnaire de librairies :

- URL: https://github.com/RobTillaart/MCP23017_RT (V0.9.1)
- URL : https://github.com/RobTillaart/PCA9685_RT (V0.7.2)
- URL : <https://github.com/MHeironimus/ArduinoJoystickLibrary> (V2.1.1)

ou depuis les *.zip fournis. Menu : Croquis > Inclure une bibliothèque > Ajouter la bibliothèque .ZIP... >

- MCP23017_RT-master.zip
- PCA9685_RT-master.zip
- ArduinoJoystickLibrary-master.zip

Programmation de l'Arduino et dépannage

Avant la mise sous tension, vérifier au minimum, pas de court-circuit entre :

- Le + 5 Volts, le + 5 Volts de puissance et le +3,3 Volts.
- Le + 5 Volts et le + 5 Volts de puissance.
- Le + 5 Volts et le + 5 Volts de puissance et la masse.

Alimenter la plaque en 12 Volts. On doit avoir une consommation de moins de 100 mA.

Programmer l'Arduino depuis l'interface IDE.

Il ne faut pas brancher maintenant le port usb Natif qui émule un clavier, à l'ordinateur. Si on le branche, la programmation de la carte n'est pas possible.

Si il y a un problème de compilation ou erreur 'keyboard.h', installer la bibliothèque : Arduino SAM boards (32-bit arm cortex-m0). En branchant cette carte, on doit vous proposer de la télécharger. Mettre à jour les librairies déjà installées.

Une fois alimentée en 12 Volts, si l'écran LCD n'affiche pas le menu d'accueil :

Appuyer sur le bouton Reset.

Régler le potentiomètre de réglage de contraste sous l'écran LCD. En position usine, rien ne s'affichera.

Vérifier que l'alimentation 12 Volts est bien branchée. Il faut que le voltmètre de la carte indique 12 Volts.

Vérifier les connecteurs à barrettes, et mettre un coup d'aérosol à contacts. Parfois les tiges des barrettes mâles glissent et ne font plus contact.

Vérifier si il n'y pas eut une inversion des signaux SDA/SCL.

Mettre le 12 Volts, avant la tension sur les prises usb.

Vérifier le câblage et l'insertion des prises.

Si les boutons du boîtier de commande ne réagissent pas essayer le menu : (2) Information Système > Touche du LCD. J'ai déjà eu plusieurs cartes Arduino DUE avec les entrées analogiques en panne.

Poser vos questions sur le forum RMF : <https://www.rmfmagazine.com/phpBB/viewforum.php?f=21&sid=2b6958827b62a3a43f687cc722793b9b>

L'installation des programmes sur l'ordinateur Windows.

Une fois la carte Arduino programmée, on branchera les deux ports de l'Arduino DUE à l'ordinateur.

1 / Le port "Programming Atmega" assure la liaison série avec de l'ordinateur.

Quand on branche la carte Arduino, sous Windows, on la retrouvera sous le nom COMxx (Ex : COM41).

L'ordinateur enverra des données par ce port COMxx à l'Arduino DUE.

La vitesse de cette liaison série avec l'Arduino est de 38400 bps.

2 / Le port "Native usb Sam3x" assure l'émulation d'un clavier supplémentaire, pour envoyer des codes de touche au simulateur, comme le fait le clavier principal.

Le logiciel "Train Simulator Classic RailWorks de DTG" est installé sur l'ordinateur.

On doit configurer ce programme avec les commandes en mode "**Expert**", depuis le menu "Paramètres".

On installera la locomotive **CC72000 de SimExpress**.

On peut installer une autre locomotive, mais il faudra reprendre des codes de touche, dans le programme de l'Arduino DUE. Par exemple, la touche du clavier commandant de sablage peut être différente.

On installe le programme "TSClasic Raildriver and Joystick Interface V3.3.0.7". Ce programme va chercher les informations dans le programme "Train Simulator Classic", pour les exporter :

<https://forums.dovetailgames.com/threads/ts-classic-raildriver-and-joystick-interface.72488/>

Faire une sauvegarde des fichiers originaux, puis copier mes fichiers en les renommant :

- **AssignedPorts** - Fichier de référence pour **CC72000_SE.txt** sous le nom **AssignedPorts.txt** dans le répertoire : TSClasic Raildriver and Joystick Interface/KeysMaps/Ports/

- **ControlNames** - Fichier de référence pour **CC72000_SE.txt** sous le nom **ControlNames.txt** dans le répertoire : TSClassic Raildriver and Joystick Interface/Settings/

Le fichier **ControlNames.txt** contient toutes les variables que le programme d'interface va chercher dans "Train Simulator Classic RailWorks de DTG", pour l'envoyer à l'Arduino DUE.

- Le premier champ est un nom libre, pour classer les informations.
- Le deuxième champ est un nom libre pour classer les informations, en plus court. Ne pas dépasser pas 3 caractères, car ce champ est envoyé à l'Arduino, et cela prend du temps sur la liaison série à 38400 Bps.
- Le troisième champ est le plus important. C'est lui qui est la référence entre l'ordinateur et l'Arduino. Ce champ n'est pas libre. Il est imposé.

On retrouve le libellé de ce champ dans le fichier "**CC72000.txt**" fourni avec la locomotive, ou quand le simulateur et l'interface ont fonctionnés, dans le fichier : TSClassic Raildriver and Joystick Interface/debug.txt.

1 / Extrait du fichier **ControlNames.txt** :

```
LEVIER=LEV=Chauffage_Train
VOYANTS=VOY=VY_Batt
MANO=MNO=RE_Anim
PWM=PWM=AMP_Moteur
```

2 / Extrait du fichier **AssignedPorts.txt** avec les mêmes variables, avec l'ajout du port série utilisé :

```
BaudRate=38400
LEVIER=LEV=Chauffage_Train=COM41
VOYANTS=VOY=VY_Batt=COM41
MANO=MNO=RE_Anim=COM41
PWM=PWM=AMP_Moteur=COM41
```

Dans le fichier **AssignedPorts.txt**, il faudra modifier le port **COM41** avec le n° de port attribué sur votre ordinateur.

Lancer le programme : TSClassic Raildriver and Joystick Interface

Dans le menu : Settings > Mode > Cocher la case ☒ **Advanced**.

Faire une extraction des données, menu : Railworks Data Extractor > Extract All Files (*C'est long*).

Dans le menu : Output Data > Update Serial Data, on retrouve les données du fichier "**AssignedPorts.txt**".

Vérifier la vitesse de transmission = 38400.

Dans le menu : Settings Edit Control Names, on retrouve le contenu du fichier "**ControlNames.txt**".

Ne pas faire d'erreur en les modifiant manuellement ces fichiers .txt. En cas d'erreur, le programme d'interface supprime des lignes dans ces fichiers. En tout cas, garder une copie.

On peut aussi remplir ces fichiers avec le programme d'interface, mais c'est plus long.

Si l'on ajoute une locomotive, faire une extraction des données, menu : Railworks Data Extractor > Extract Single Folder.

Une fois installé et configuré, l'ordinateur envoie ce genre de trames à l'Arduino :

```
<RPM : VirtualRPM : .670><Speed : SpeedometerKPH : .010> <Acceleration : Acceleration : .352> <RPM : VirtualRPM : .700> <Speed : SpeedometerKPH : .011>
<Acceleration : Acceleration : -.034> <RPM : VirtualRPM : .705> <Speed : SpeedometerKPH : .012> <Acceleration : Acceleration : .165> <RPM : VirtualRPM :
.701> <Speed : SpeedometerKPH : .011> <Acceleration : Acceleration : .046> <RPM : VirtualRPM : .698> <Speed : SpeedometerKPH : .011> <Acceleration :
Acceleration : -.172> <RPM : VirtualRPM : .690> <Speed : SpeedometerKPH : .011> <Acceleration : Acceleration : .002> <RPM : VirtualRPM : .701> <Speed :
SpeedometerKPH : .011> <Acceleration : Acceleration : -.390> <RPM : VirtualRPM : .707> <Speed : SpeedometerKPH : .011> <Acceleration : Acceleration :
.000> <RPM : VirtualRPM : .707> <Speed : SpeedometerKPH : .012> <Acceleration : Acceleration : .000> <RPM : VirtualRPM : .706> <Speed : SpeedometerKPH :
.012>
```

Le programme 'Data_Envoyees_vers_PC_TSCRJI_Y.ino' traite les entrées suivantes :

Les entrées marquées X ne sont pas utilisées, et restent disponibles.

Les entrées marquées ??, ne sont pas traitées, en absence de code de touche à envoyer au simulateur.

Il faut éviter d'utiliser les entrées n° 0 et n° 15, car il existe un risque de dysfonctionnement des MCP23017.

Si l'on a besoin de ces entrées, les brancher sur des équipements qui ne change pas souvent d'état.

ARDUINO - ENTREES - CC72000 SimExpress Le 11/05/2025

Variables de retour d'info

HVFEP Isol F.E.P. - Interrupteur - momentan� inverse la position - '^f'	0	IN1	inter_FEP_control
Sablage - Bouton poussoir pouvant �tre maintenu - 'x'	1		Pas besoin
BP Desserrage Grand D�bit - Bouton poussoir pouvant �tre maintenu - ':'	2		Pas besoin
BP Urgence Rouge - - momentan� inverse la position - 'BackSpace'	3		BP_URG_control
FD - Frein direct Grand levier - Interrupteur en avant - Une impulsion - ';	4	IN1	FD_control AV= - 0,5
FD - Frein direct Grand levier - Interrupteur en arri�re - Une impulsion - ', '	5		O= 0 AR= 0.50
FA - Frein de train Petit levier - Bouton poussoir pouvant �tre maintenu, en avant - 'c'	6		Pas besoin
FA - Frein de train Petit levier - Bouton poussoir pouvant �tre maintenu, en arri�re - 'w'	7		Pas besoin
Manipulateur de traction - (-) - Interrupteur - 'q/d'	8	IN1	CC6500_manip_1_control
Manipulateur de traction - STOP - Interrupteur - 'q/d'	9		(O) = 0 (-) = 0,5
Manipulateur de traction - (+) - Interrupteur - 'q/d'	10		STOP = 0,75 (+) = 1,0
Manipulateur de traction - PR - Interrupteur - 'q/d'	11		PR = 1,5
Inverseur de traction, MPJ - Interrupteur en avant - Une impulsion (Si d�blocage) - 'z'	12	IN1	CC6500_inverseur_control
Inverseur de traction, MPJ - Interrupteur en arri�re - Une impulsion (Si d�blocage) - 's'	13		AV=1,0 O=0,0 AR= - 1,0
Cerclo - VACMA - Veille active VA - Bouton poussoir pouvant �tre maintenu - 'Espace'	14		Pas besoin
BP Arm FEP - BP_FEP_control - Bouton poussoir pouvant �tre maintenu - '^e' (Lance l'�diteur !)	15		Pas besoin
Pas de contact sur le bouton de d�blocage sur le Cerclo			
La vitesse des essuie-glaces est un potentiom�tre branch� sur l'entr�e AN0			
Boitier de commande :			
(Noir) = FAM - Frein � main - Shift+Entr�e			
(Blanc) = Mise en route batterie + ZECL - '^B' + '^E'			
(Bleu) = Isol KVB - 'Numpad moins'			
R�armement QT- Interrupteur - Bouton poussoir pouvant �tre maintenu - 'u'	0	IN 2	X
Compresseur Auto - Interrupteur - momentan� inverse la position - 'i'	1		Pas besoin
Compresseur Direct - Interrupteur - momentan� inverse la position - 'l'	2		inter_compresseur_auto_control
	3		CC6500_Z_compresseur_direct_control
Inter chauffage Contr�le - Interrupteur - momentan� inverse la position - 'y'	4	IN 2	inter_chauffage_control
Z s�rie control acc�l�ration diesel - 'Y'	5		CC6500_Z_serie_control
Eclairage Cabine - Interrupteur - momentan� inverse la position - 'k'	6		inter_cabine_control
Eclairage Fiche Horaire - Interrupteur - momentan� inverse la position - '.'	7		X
Eclairage Pupitre - Interrupteur - momentan� inverse la position - 'l'	8	IN 2	inter_lum_pup_control
Fanal Gauche - Interrupteur - momentan� inverse la position - 'h'	9		inter_fanal_G_control
Fanal Droit - Interrupteur - momentan� inverse la position - 'H'	10		inter_fanal_D_control
Projecteur - Interrupteur - momentan� inverse la position - 'g'	11		inter_projecteurs_control
Essai VACMA - Interrupteur - momentan� inverse la position (Ne bouge pas � l'�cran) - 'R'	12	IN 2	inter_essai_VA_control
	13		X
BL - Boite � leviers - Interrupteur - momentan� inverse la position - 'b'	14		cle_BL_control
	15		X
X			
Acquittement signaux - Bouton poussoir pouvant �tre maintenu - 'Entr�e pav� num'	0	IN 3	X
BP Inter SAL- Interrupteur - momentan� inverse la position - '^H'	1		Pas besoin
BP Arr�t Moteur - Bouton poussoir pouvant �tre maintenu - 'o'	2		Z_alerte_control
	3		Pas besoin
Essuie-glaces - Interrupteur - momentan� inverse la position - 'v'	4	IN 3	Wipers
Cl� de frein - Interrupteurs 4 positions - Neutre - 'n/N'	5		cle_frein_control
En service - 'n/N'	6		Retrait = 0 Isolation = 0,33
Isolement - 'n/N'	7		En service=0,7 Neutre =1
Avertisseur - Bouton poussoir pouvant �tre maintenu - 'page up'	8	IN 3	Pas besoin
Avertisseur - Bouton poussoir pouvant �tre maintenu - 'page down'	9		Pas besoin
ZLFM - Projecteur Frontal - Interrupteur - momentan� inverse la position - '^h'	10		??
ZLFRG - Feu Rouge Gauche - Interrupteur - momentan� inverse la position - 'f'	11		inter_FRG_control
ZLFRG - Feu Rouge Droit - Interrupteur - momentan� inverse la position - 'F'	12	IN 3	inter_FRD_control
BP ALVA ISOL VA- ZVA - isolement VA - Bouton poussoir pouvant �tre maintenu - 'r'	13		Pas besoin
Inter Surcharge - Interrupteur - Calcul en local Arduino	14		X
	15		X

Eviter d'utiliser les entrées n° 0 et n° 15

'r' = Touche [r]

'^r' = Touche [Ctrl]+[r]

?? = Pas câblé, car pas de touche affectée

Variable d'entrée = Pas besoin : Sur un contact type bouton poussoir, le levier revient automatiquement à 0. Donc pas besoin de connaître sa position dans le simulateur.

Le programme 'Data_Recues_du_PC_TSCRJI_Y.ino 'traite les sorties suivantes :

Les sorties marquées X ne sont pas utilisées, et restent disponibles.

ARDUINO - SORTIES - CC72000 SimExpress Le 11/05/2025

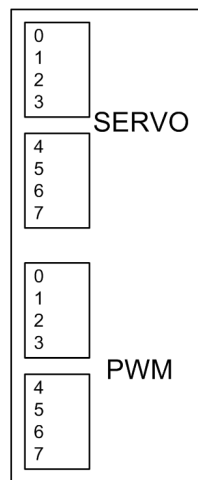
Variables de retour d'info

Manomètre RE / Gros mano à gauche / Aiguille blanche - Max 5,5
 Manomètre CP / Gros mano à gauche / Aiguille rouge - Max 9,5
 Manomètre CG / Mano au Centre - Max 5,5
 Manomètre CF1 / Mano à Droite / 1 trait - Max 4,0

Manomètre CF2 / Mano à Droite / 2 traits - Max 4,0
 X
 X
 X

VA Batterie
 Ampèremètre Moteur de Traction 1
 Ampèremètre Moteur de Traction 1
 U redressée

Compteur de vitesse - Tachro
 X
 X
 X



RE_control
 CP_control (Valeur + 4 Bars)
 CG_control
 CF_control
 CF_control (idem CF1)
 X
 X
 X

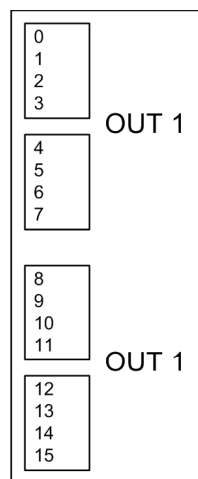
CC72000_levier_batt_control = 2
 Regulator (Val = 1 pour 2 KA)
 Regulator (Val = 1 pour 2 KA)
 ??

CC_aigu_tachro_1_control ou ..2..
 X
 X
 X

Picto - Disjoncteur - QT
 Picto - Batterie - BAT
 Picto - Patinage - PAT
 Picto - Vreg - Régulation chauffage train - CB
 Picto - Qmin - SMG - Servomoteur à zéro
 Picto - Qmax : LT SSMH - Servomoteur au MAX
 Picto - (PR) - Interdiction passage de cran supérieur
 Picto - Frein électro pneumatique - FEP

Picto - P.H 1.1
 Picto - P.H 1.2
 Picto - P.H 2.1
 Picto - P.H 2.2

SAL - Alerte
 Surcharge
 X
 X



LS_QT_control
 LS_BAT_control
 LS_PAT_control
 LS_Vreg_control

LS_GR_control
 ??
 CC6500_manip_1_control = 1.5
 LS_FEP_control

LS_PH1_control
 LS_PH2_control
 LS_PH3_control
 LS_PH4_control

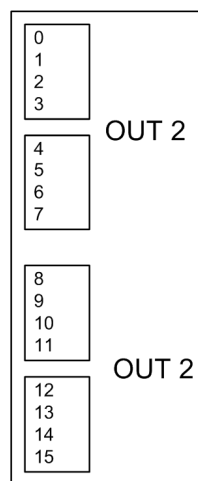
SAL_control
 Calcul Local Arduino
 X
 X

Alarme sonore RA
 Alarme sonore MA
 Alarme sonore PORTES
 X

X
 X
 X
 X

X
 X
 X
 X

X
 X
 X
 X



son_RA_control
 son_MA_control
 Sons_porte_ouverte_control
 X

X
 X
 X
 X

X
 X
 X
 X

X
 X
 X
 X

Sorties PCA9685, courant maxi :
 - 25 mA entre une sortie et le + 5 Volts
 - 10 mA entre une sortie et la masse.

Servomoteur - Impulsion de 700 µs à 2300 µs
 PWM - Fréquence = 62 Hz

Le programme 'Data_Envoyée_vers_le_PC_TSCRJI_xx.ino'

Quand une des entrées sur les cartes d'entrée 1, 2 ou 3 change d'état, le programme la traite.

Il envoie alors un code de touche au simulateur sur l'ordinateur. Par exemple les lettres 'Q' et 'D' pour le Cerclo.

Il faut convertir le code des touches d'un clavier azerty vers un clavier qwerty.

En branchant le port "Native usb Sam3x" de l'Arduino sur l'ordinateur, il émule automatiquement un clavier. Il n'y a rien d'autre à faire.

Attention, quand on manipule le pupitre et que "Train Simulator Classic" n'est pas lancé, des codes de touches vont être envoyés à l'ordinateur, comme si on les tapait sur le clavier.

Quand une entrée a bougé, on récupère la valeur de l'entrée $X = \text{bitRead}(\text{i2c_in1_val}, X) = 0$ ou 1 . ($1 = \text{ON}$).

On envoie un code de touche au simulateur.

La routine **tch_env**(Normal, Majuscule ou Control, Code de la touche, Durée d'appui en msec) assure l'envoi en tâche de fond des codes des touches.

Dans ce programme, on traite les 16 entrées de la carte in1, de la carte in2 puis de la carte in3.

On traite les boutons poussoir, en envoyant un code de touche "PRESS" à la fermeture du contact, puis un code de touche "RELEASE" à l'ouverture du contact.

Dans ce cas, on utilise les codes "PRESS" et "RELEASE" à la place d'un délai en msec.

```
if (x == 1) { if(bitRead(i2c_in1_val, x) == 1) tch_env(NORMAL, 'x', PRESS); else tch_env(NORMAL, 'x', RELEASE); } //  
x      Bouton poussoir Sable.  
  
else if (x == 2) { if(bitRead(i2c_in1_val, x) == 1) tch_env(NORMAL, '.', PRESS); else tch_env(NORMAL, '.', RELEASE); } //  
. pour / Desserrage Grand Débit.  
  
else if (x == 6) { if(bitRead(i2c_in1_val, x) == 1) tch_env(NORMAL, 'c', PRESS); else tch_env(NORMAL, 'c', RELEASE); } //  
c      Petit levier Frein du train en avant.
```

On traite les interrupteurs en envoyant un code de touche, à chaque changement d'état de l'entrée.

Dans ce cas, on garde en mémoire l'état de l'interrupteur dans le champ : **i2c_in1_val_ref**.

Le programme de réception des données, va mettre à jour cet état depuis le simulateur.

S'il y a un écart, on enverra de nouveau un code de touche.

Comme cela, l'état du simulateur est toujours conforme à celui du pupitre.

```
if (x == 3) { tch_env(NORMAL, KEY_BACKSPACE, duree_touche_100);  
bitWrite(i2c_in1_val_ref, x, bitRead(i2c_in1_val, x)); } // 'KEY_BACKSPACE' Frein d'urgence.  
  
else if ((x == 4) && (bitRead(i2c_in1_val, x) == 1)) { tch_env(NORMAL, ',', duree_touche_200); bitWrite(i2c_in1_val_ref, x,  
bitRead(i2c_in1_val, x)); } // , pour . Grand levier Frein direct en avant. 200 msec.  
  
else if ((x == 5) && (bitRead(i2c_in1_val, x) == 1)) { tch_env(NORMAL, 'm', duree_touche_100); bitWrite(i2c_in1_val_ref, x,  
bitRead(i2c_in1_val, x)); } // m pour ? Grand levier Frein direct en arrière.  
  
else if ((x == 12) && (bitRead(i2c_in1_val, x) == 1)) { tch_env(NORMAL, 'w', duree_touche_300); bitWrite(i2c_in1_val_ref, x,  
bitRead(i2c_in1_val, x)); } // w pour z Inverseur de traction, en avant. 300 msec.
```

Pour les interrupteurs à plusieurs positions, il manque un contact pour savoir si on est en position centrale.

Dans ce cas, si les interrupteurs sont tous ouverts, on remet automatiquement en place le levier.

```
if ((x == 4) || (x == 5)) && (bitRead(i2c_in1_val, 4) == 0) && (bitRead(i2c_in1_val, 5) == 0) {  
  
    if (bitRead(i2c_in1_val_ref, 4) == 1) { tch_env(NORMAL, 'm', duree_touche_100); } // m pour ? Grand levier Frein  
direct en arrière. 100 msec.
```

```

    else if (bitRead(i2c_in1_val_ref, 5) == 1) { tch_env(NORMAL, ',', duree_touche_200); } // , pour . Grand levier Frein
    direct en avant. 200 msec.

    bitWrite(i2c_in1_val_ref, 4, 0); bitWrite(i2c_in1_val_ref, 5, 0);
}

```

Si il y a des potentiomètres, comme pour la lecture du Cerclo, on les traite en envoyant des séquences de durée calibrées de touches.

C'est subtil à programmer, car il faut définir par des essais successifs, la durée à attribuer aux touches pour obtenir un mouvement de 0 à 100%.

Suivant les appareils, on peut choisir d'utiliser 8, 16 ou 32 pas.

On envoie une touche pendant 25 msec, pour un faible déplacement, à 200 pour un grand déplacement.

```

if (millis() > lect_can_milli) { // Pour ne pas passer 25000 fois par seconde dans ce code. On économise du
cpu par boucle.

    lect_can_milli = millis() + 100; // On passe ici toutes les 100 msec.

    regul_val = analogRead(pin_regul_broche); // 0 à 1023.

    if ((regul_val - regul_val_1023 > 10) || (regul_val - regul_val_1023 < -10)) { // Un peu d'hystérésis, pour que la valeur
ne bouge pas sans arrêt.

        regul_val_1023 = regul_val;
    }

    regul_val = regul_val_1023 >> 5; // >>5 : 0 à 31 = 32 Pas de régulation.

    if (regul_val != regul_val_ref) {

        if (regul_val == 31) { tch_env(NORMAL, 'a', 200); regul_val_ref = regul_val_ref + 7; } // a pour q
Traction : (+) <-- On arrive en butée.

        else if (regul_val == 0) { tch_env(NORMAL, 'd', 200); regul_val_ref = regul_val_ref - 7; } // d
Traction : --> (-) On arrive en butée.

        else if (regul_val >= regul_val_ref + 4) { tch_env(NORMAL, 'a', 100); regul_val_ref = regul_val_ref + 4; } // a pour q
Traction : (+) <--

        else if (regul_val <= regul_val_ref - 4) { tch_env(NORMAL, 'd', 100); regul_val_ref = regul_val_ref - 4; } // d
Traction : --> (-)

        else if (regul_val > regul_val_ref) { tch_env(NORMAL, 'a', 25); regul_val_ref++; } // a pour q
Traction : (+) <--

        else if (regul_val < regul_val_ref) { tch_env(NORMAL, 'd', 25); regul_val_ref--; } // d
Traction : --> (-)

        if (regul_val_ref > 31) regul_val_ref = 31;

        else if (regul_val_ref < 0) regul_val_ref = 0;

        ic2_in_comp_ref_milli = ic2_in_comp_ref_milli + duree_touche_200; // Si le cerclo a bougé sur le pupitre, on attendra un
peu plus avant de vérifier les valeurs du simu.
    }
}

```

Si l'on a branché un boîtier KVB, on récupère l'information d'un bouton appuyé sur le KVB, pour l'envoyer au simulateur.

```

if (Serial3.available() > 0) {

    char car_lu = Serial3.read();

    if (car_lu == 'A') { tch_env(KEY_LEFT_SHIFT+KEY_LEFT_CTRL, 'v', duree_touche_200); } // Appui sur le bouton [VAL] du
KVB. 200 msec obligatoire

    if (car_lu == 'E') { tch_env(NORMAL, KEY_KP_ENTER, duree_touche_200); } // Appui sur le bouton [SF] du
KVB. 200 msec obligatoire
}

```

On choisit les touches à envoyer lorsque l'on appuie sur les boutons du boîtier de commande.


```

if (int_0_val == 1 ) { // Si l'interrupteur "ENVOI" est sur 'ON' = 0, on inhibe l'envoi des touches au PC.

    if ((bp_1_cpt == 0xFFFF) && (bp_1_val == 0)) { tch_env(KEY_LEFT_CTRL, 'n', duree_touche_100); bp_1_val = 1; } // ^n :
    Sectionneur batterie.

    if ((bp_2_cpt == 0xFFFF) && (bp_2_val == 0)) { tch_env(KEY_LEFT_SHIFT, 'e', duree_touche_100); bp_2_val = 1; } // MajE :
    Commutateur éclairage.

    if ((bp_3_cpt == 0xFFFF) && (bp_3_val == 0)) { tch_env(KEY_LEFT_CTRL, 'n', duree_touche_100); bp_3_val = 1; } // n :
    Frein à main.

```

Le programme 'Data_Recues_du_PC_TSCRJI_xx.ino'

Il reçoit les trames de l'ordinateur sous la forme : <Speed : SpeedometerKPH : 150.0>

Une fois la trame complète reçue, le programme renseigne les trois chaînes de caractères suivantes :

- buffer_rx[0] = Nom du groupe de données (Texte).
- buffer_rx[1] = Nom de la donnée (Texte).
- buffer_rx[2] = Valeur au format texte (Nombre entier ou avec virgule).

Le champ buffer_rx[0] ne me sert pas, c'est pour cela que je mets que 3 caractères, pour réduire le temps de transmission de la trame.

Le champ buffer_rx[1] est le 3^{ème} champ du fichier "AssignedPorts.txt".

Exemple : Chauffage_Train, VY_Batt, RE_Anim, AMP_Moteur.

Le champ buffer_rx[2] est la valeur de la variable. C'est du texte ou des nombres transmis au format texte. Ça peut être des entiers ou des nombres flottants.

1 / SERVOMOTEURS

Pour les sorties de type SERVO, on renseigne un élément du tableau "i2c_srv_val[]", avec une valeur en µsec. Ça donne une valeur comprise entre **700** µsec et **2300** µsec. Attention, ces valeurs extrêmes dépendent du modèle de servomoteurs installés.

Les valeurs d'un servo standard sont comprises entre **1000** µsec et **2000** µsec, pour 90° de rotation.

Les valeurs d'un servo plus performant comme le 7350MG-D, entre **700** µsec et **2300** µsec pour 150°.

Il ne faut pas dépasser les valeurs admissibles par le servomoteur utilisé, sinon le moteur risque de forcer et de griller.

Pour les servomoteurs "7350MG-D", la plage utilisable va de 600 à 2200 µs. Par contre la plage où le servomoteur a une course linéaire, va de 700 à 2200 µs.

On choisit la course du servomoteur pour un manomètre, en prenant pour l'indication 0 Bar, 700 µs ou 2200 µs suivant la rotation du servomoteur. Ainsi, la majorité de la course du servomoteur sera linéaire.

Exemple dans le code, pour un servomoteur d'un manomètre.

Ce manomètre a un cadran de 0 à 10 Bars, mais la valeur maximum affichée sera de 5,5 Bars.

Dans le cas d'un servomoteur de type 7350MG-D, lors de la modification du manomètre, on fait en sorte d'utiliser la plage la plus grande possible (700 µs à 2300 µs) pour parcourir tout le cadran.

Le servomoteur installé, indique 0 Bars pour 2200 µs et 10 Bars pour 600 µs.

Lors du remontage du manomètre, il faut envoyer une impulsion de 2200 µs et ensuite positionner l'aiguille sur son axe.

```
/ < :RE_control:5.50>                                ==> Sortie servo 0 - Aiguille BLANCHE RE, , Affichage
5,5 Bars maximum sur cadran 10 Bars.
if (!strcmp(buffer_rx[1],"RE_control")) {                // Affichage 5.5 Bars maximum.
    buffer_rx_fvaleur = atof(buffer_rx[2]);
    buffer_rx_fvaleur = 155.0*buffer_rx_fvaleur;         // 5,5 Bars maximum reçu : Le cadran indique 0 Bar
pour 2200 µs et 5,5 Bars pour 1380 µs : Coefficient = (2200-1380)/(5,5) = 149
    buffer_rx_valeur = 2200 - int(buffer_rx_fvaleur);    // Valeur minimum = 2200 µsec.
    buffer_rx_valeur = min(buffer_rx_valeur, 2200);     // Valeur < 2200 µsec. Valeurs extrêmes admissibles
pour ce type de servomoteur.
    i2c_srv_val[0] = max(buffer_rx_valeur, 700);        // Valeur > 700 µsec. Renseigner "i2c_pwm_init".
    bitWrite(i2c_srv_maj, 0, 1)
```

Explication des lignes de code :

Variable flottante (à virgule) pour les calculs : **buffer_rx_fvaleur**

Variable entière (sans virgule) pour les calculs : **buffer_rx_valeur**

Il faut borner la variable en sortie, pour rester dans les limites **700** et **2200**, avec les fonctions **min()** et **max()**.

Ces limites dépendent du modèle de servomoteur. Si l'on utilise un servomoteur standard, prendre 1000 µs et 2000 µs.

On affecte le résultat à l'indice **[0]** du tableau de valeur : **i2c_srv_val[0]**

On averti le programme pour mettre à jour cette sortie numéro **0** : **bitWrite(i2c_srv_maj, 0, 1);**

Si le servomoteur tourne dans l'autre sens.

Si le servomoteur tourne à l'envers, il faut afficher 0 Bar quand l'impulsion est égale à 700 µs.

La ligne de calcul modifiée est la suivante :

```
buffer_rx_valeur = 700 + int(buffer_rx_fvaleur); // Valeur minimum = 700 µsec.
```

Position des servomoteurs au démarrage du programme

Dans tous les cas, il faut définir la position des servomoteurs au démarrage du programme.

Dans le code, il faut renseigner la ligne :

```
const int i2c_srv_init[] = {700, 700, 700, 2200, 2200, 2200, 2300, 2200};
```

Pour les 8 sorties SERVO. Il est indiqué la position en µsec des 8 servomoteurs en µsec.

2 / SORTIES PWM

Pour les sorties de type PWM, on renseigne un tableau "i2c_pwm_val[]", avec une valeur comprise entre **0** et **4095**.

La valeur "0" mettra la sortie à 0 Volt.

La valeur "4095" mettra la sortie à 5 Volts.

On a besoin d'une impulsion normale, (0 = 0 Volt), pour les Galva amplifié, Carte Tri, Compte-tours, Tachro, utiliser la formule Valeur = Coefficient * Valeur_reçue.

On a besoin d'une impulsion inversée (0 = 5 Volts), pour les Galva non amplifié relié au (+5V).

Exemple dans le code, pour une sortie pwm.

```
// < :aiguille_UL_C_control:16.00> ==> Sortie pwm 0 Tension ligne. 0 à 16. 15.00 reçu = 1,5KV affiché. Configuration sortie n° 0 = Impulsion inversée.
else if(!strcmp(buffer_rx[1],"aiguille_UL_C_control")){// Volts transmis en flottant.
    buffer_rx_fvaleur = atof(buffer_rx[2]); // Conversion du dernier champ : Ascii-> Flottant. Pas de contrôle ! Si Nok valeur = 0 !
    buffer_rx_fvaleur = 256*buffer_rx_fvaleur; // 16 au maximum => buffer_rx_fvaleur = 4096 au maximum (Coéf = 4096/16). Une seule multiplication en
    flottant, pour économiser le cpu.
    buffer_rx_valeur = int(buffer_rx_fvaleur); //
    buffer_rx_valeur = min(buffer_rx_valeur, 4095); // Valeur <= 4095.
    buffer_rx_valeur = max(buffer_rx_valeur, 0); // Valeur positive.
// i2c_pwm_val[0] = buffer_rx_valeur; // Si impulsion Normale, Galva amplifié, Carte Tri, Compte-tours, Tachro. Ne pas oublier de renseigner
    "i2c_pwm_inv_const". Bit(x) = 0.
    i2c_pwm_val[0] = 4095 - buffer_rx_valeur; // Si impulsion Inversée. Galva non amplifié relié au (+). Ne pas oublier de renseigner "i2c_pwm_inv_const".
    Bit(x) = 1.
    bitWrite(i2c_pwm_maj, 0, 1);
```

3 / SORTIES TOUT OU RIEN

Exemple pour une sortie tout ou rien.

Attention pour des interrupteurs, souvent la valeur reçue passe par plusieurs valeurs intermédiaires. 0, 0.12, 0.35, 0.70, 1.00. C'est pour cela que je teste les valeurs avec la routine : **buffer2_compare_float_maj_out**(out1, Numéro de la sortie).

Cette routine compare la valeur reçue à 0,02 et 0,98, pour savoir si c'est un 0 ou un 1.

Si l'on est sûr que la valeur envoyée est 0 ou 1, on utilisera la routine **buffer2_compare_int_maj_out**().

Sinon, on utilisera la routine : **buffer2_compare_float_maj_out**().

```
else if (!strcmp(buffer_rx[1], "LS_DJ_control")) buffer2_compare_int_maj_out(out1, 0); // < :LS_DJ_control:1>
Disjoncteur
```

Si l'on veut recevoir l'état d'un interrupteur, on utilise la routine **buffer2_compare_float_maj_sim**().

Cette routine met à jour la valeur de référence de l'entrée x, et déclenche une comparaison avec l'entrée physique.

```
else if (!strcmp(buffer_rx[1], "inter_DJ_control")) buffer2_compare_float_maj_sim(in2, 1); // < :inter_DJ_control:1>
Info en retour sur le Disjoncteur. La valeur passe progressivement de 0.0 à 1.00.
```

Pour les interrupteurs à plusieurs positions, il faut traiter la valeur reçue pour vérifier la bonne position des interrupteurs.

```
else if (!strcmp(buffer_rx[1], "FD_control")) { // Grand levier de frein direct à trois positions.

    buffer_rx_fvaleur = atof(buffer_rx[2]); // Obligé de passer par un nombre flottant.

    if (buffer_rx_fvaleur > 0.38) { bitWrite(i2c_in1_val_sim, 4, 0); bitWrite(i2c_in1_val_sim, 5, 1);

        bitWrite(i2c_in1_maj_sim, 4, 1); bitWrite(i2c_in1_maj_sim, 5, 1); // La comparaison et demande de mise à jour, se fera
        automatiquement toutes les xxx msec.
    }

    else if (buffer_rx_fvaleur < -0.23) { bitWrite(i2c_in1_val_sim, 4, 1); bitWrite(i2c_in1_val_sim, 5, 0);
        bitWrite(i2c_in1_maj_sim, 4, 1); bitWrite(i2c_in1_maj_sim, 5, 1); // La comparaison et demande de mise à jour, se fera
        automatiquement toutes les xxx msec.
    }

    else if ((buffer_rx_fvaleur > 0.08) && (buffer_rx_fvaleur < 0.12)) { bitWrite(i2c_in1_val_sim, 4, 0);
        bitWrite(i2c_in1_val_sim, 5, 0);

        bitWrite(i2c_in1_maj_sim, 4, 1); bitWrite(i2c_in1_maj_sim, 5, 1); // La comparaison et demande de mise à jour, se fera
        automatiquement toutes les xxx msec.
    }
}
```

Pour le KVB, une fois reçu une commande du simulateur, on la renvoie au KVB.

Le caractère placé dans **KVB_car_vers_KVB** sera automatiquement envoyé au KVB.

```
if(!strcmp(buffer_rx[0], "KVB")) {
    buffer_rx_valeur = atoi(buffer_rx[2]); // Valeur = 0 ou 1.

    if (!strcmp(buffer_rx[1], "KVB_BP_VAL_lumiere_control")) {if(buffer_rx_valeur == 1) KVB_car_vers_KVB = 'G'; else
    KVB_car_vers_KVB = 'g'; }

    else if(!strcmp(buffer_rx[1], "KVB_BP_CAR_lumiere_control")) {if(buffer_rx_valeur == 1) KVB_car_vers_KVB = 'I'; else
    KVB_car_vers_KVB = 'i'; }

    else if(!strcmp(buffer_rx[1], "KVB_BP_VIO_lumiere_control")) {if(buffer_rx_valeur == 1) KVB_car_vers_KVB = 'H'; else
    KVB_car_vers_KVB = 'h'; }
```

Test de la carte Arduino DUE

Brancher un servomoteur sur la sortie servo n° 0.

Brancher un voltmètre sur la sortie pwm n° 0 et le +5 V.

Brancher une led sur la sortie out1 n° 0 et le + 5V.

La carte Arduino DUE est déjà programmée.

1 / Mettre le montage sous tension.

Le menu d'accueil s'affiche sur l'écran LCD du boîtier de commande :

JLF - 01/05/2025

MENU: TURNER BTN

Si le menu d'accueil ne s'affiche pas correctement, appuyer sur le bouton RAZ de l'Arduino.

Régler le potentiomètre de réglage de contraste sous l'écran LCD. En position usine, rien ne s'affichera.

Il faut mettre le 12 Volts, avant la tension sur les prises usb.

Si rien ne s'affiche, appuyer sur le bouton 'Reset'.

Vérifier le câblage et l'insertion des prises.

Si les boutons du boîtier de commande ne réagissent pas essayer le menu : (4) Information Système > Touche du LCD.

J'ai déjà eu plusieurs cartes Arduino DUE avec les entrées analogiques en panne.

Brancher les deux ports de l'Arduino à l'ordinateur.

Ouvrir le bloc-notes.

Cliquer dans la fenêtre du bloc-notes.

Mettre des plots des entrées in1, in2 ou in3 à la masse.

On doit avoir des lettres qui s'affichent dans le bloc, Exemples in3_4 => 'p', in3_6 => 'y'.

Ca correspond au tableau des touches envoyées par l'Arduino.

2 / Avec le boîtier de configuration, configurer la sortie pwm n° 0 en mode inverse.

3 / Double-cliquer sur "PDC_Arduino_JLF.ino".

Dans l'interface IDE Arduino, choisir la bonne carte et le bon port série.

Lancer le moniteur série, menu : Outils > Moniteur série

Mettre la bonne vitesse = **38400 Bps**.

Sur la ligne du moniteur série, taper :

< :RE_Anim:0.00> (valider chaque commande avec la touche [Entrée])

< :RE_Anim:3.00>

< :RE_Anim:6.00>

Le servomoteur doit se positionner sur trois positions différentes.

Sur la ligne du moniteur série, taper :

< :AMP_Moteur:0> (valider chaque commande avec la touche [Entrée])

< :AMP_Moteur:1000>

< :AMP_Moteur:2000>

La sortie pwm doit passer de 0 à 2,5, puis 5 Volts

Sur la ligne du moniteur série, taper :

< :VY_QT1:0>

< :VY_QT1:1>

La sortie led soit s'allumer ou s'éteindre.

Attention

1 / Si l'on modifie les programmes d'entrées ou de sortie, ne pas confondre les variables :

i2c_in1_maj, i2c_in1_val, i2c_in1_val_ref, i2c_in1_val_sim, i2c_in1_maj_sim

i2c_in1_maj : Demande de traitement de l'entrée physique.

i2c_in1_val : Valeur de l'entrée physique.

i2c_in1_val_ref : Valeur mémorisée de l'entrée, une fois le code de touche envoyé au simulateur.

i2c_in1_maj_sim : Demande de vérification de l'entrée, car la valeur du simulateur vient d'être envoyée.

2 / Si l'on modifie les programmes d'entrées ou de sortie, ne pas confondre les variables :

i2c_in1_val, i2c_in2_val, i2c_in3_val

3 / Si l'on modifie les programmes d'entrées ou de sortie, ne pas confondre les variables :

i2c_in1_val, i2c_out1_val

Si vous voulez plus de 32 sorties pour des voyants, il faudra ajouter une carte de sortie supplémentaire, à base d'Arduino NANO.

Cette carte se trouve sur le site : http://www.la-tour.info/uts/uts_page15.html#conduite

Elle sera branchée sur le connecteur RX2/TX2 de la carte principale.

Dans ce cas, il faudra ajouter du code pour piloter la liaison série n°2. On peut s'inspirer du code utilisé pour la liaison série n°3 pour le KVB.

A+