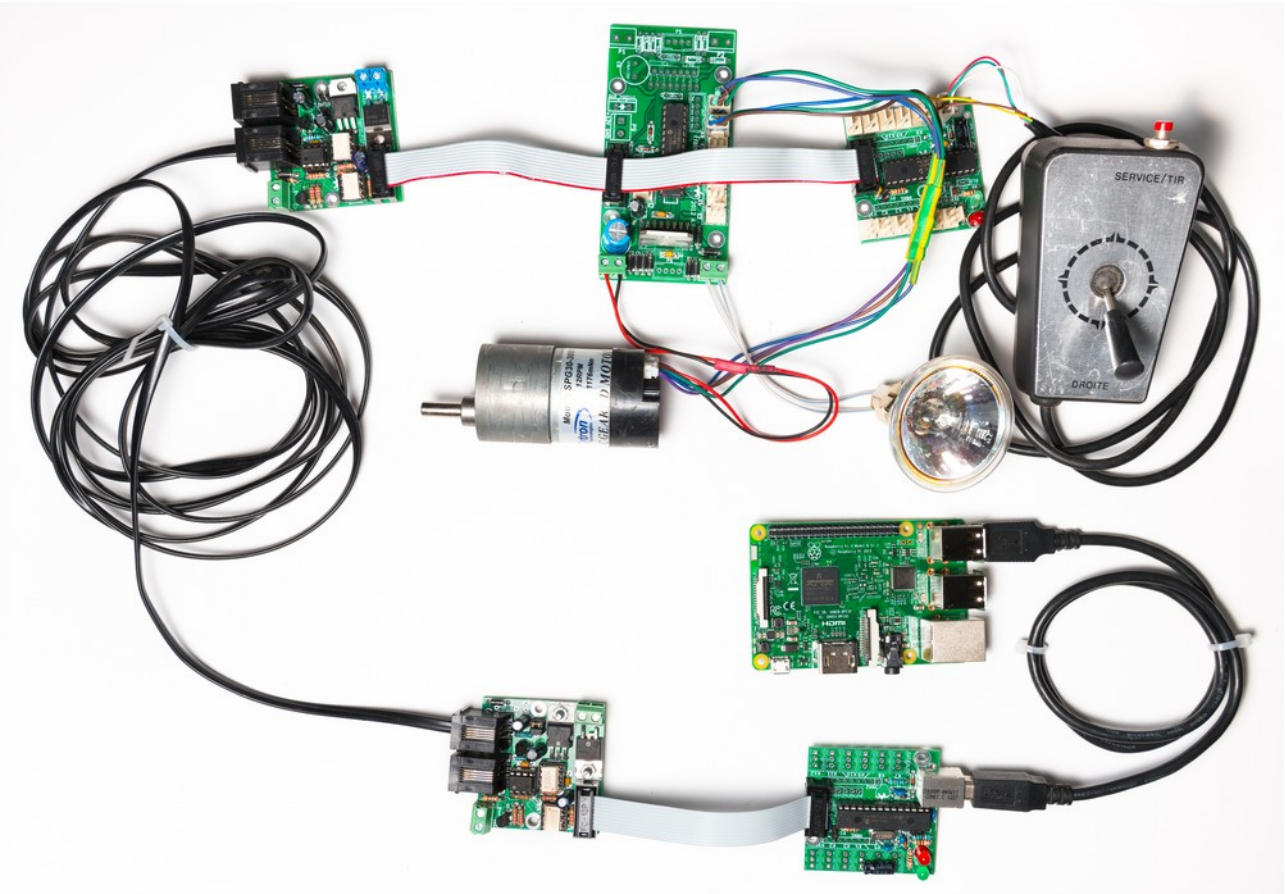
fraise

FRAmework
for Interfacing
Software & Electronics



metalu.net
Chantier créatif numérique

Table des matières

I. Présentation générale

Fraise est un environnement *open source* développé au sein de **metalu.net**, composé de cartes électroniques programmables à faible coût et d'un ensemble logiciel. Cet outil vise à simplifier la réalisation d'installations robotiques, la construction d'interfaces de contrôle (pour la musique, la lumière, le Vjing…) ou tout autre assemblage de logiciel, de capteurs et d'actionneurs.

Fraise trouve sa place dans de nombreuses créations d'artistes, dont certaines sont documentées sur le site de metalu.net.

Fraise possède de nombreux avantages, elle permet notamment :

- une meilleure symbiose entre logiciel (l'application exécutée par l'ordinateur) et matériel (les cartes électroniques interfaçant capteurs et actionneurs) ;
- de simplifier le câblage d'une machine complexe (éviter les « paquets de fils ») ;
- de ne pas être limité par la longueur des câbles (libre dimension de l'installation) ;
- de pouvoir facilement rajouter une fonction matérielle à n'importe quel endroit de l'installation ;
- d'accélérer le prototypage et le développement.

Fraise dispose de plusieurs caractéristiques qui lui sont propres :

- la connexion de plusieurs dizaines de cartes (jusqu'à 126) sur le même port USB d'un ordinateur ;
- une longueur de câbles d'interconnexion pouvant atteindre 400 mètres cumulés ;
- un environnement de développement, de test et d'exécution entièrement intégré dans **Pure Data** ;
- une bibliothèque de fonctions avancées, intégrant par exemple l'acquisition de mesures de capteurs, le contrôle de moteurs (asservissement en vitesse ou position, limitation de vitesse/accélération/freinage), la gradation de lumière…

II. Principes de base

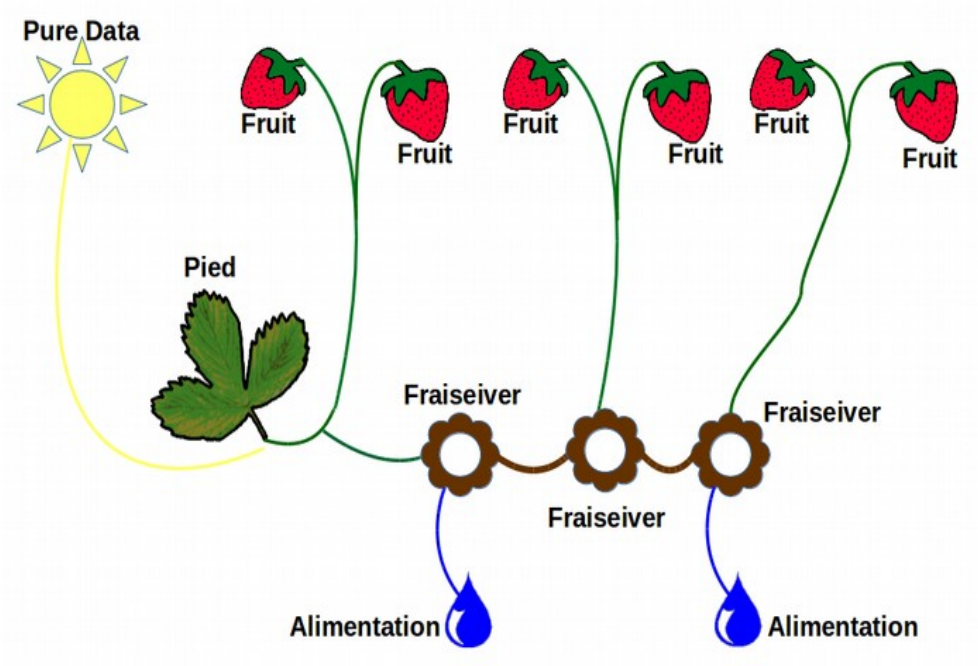


Illustration 1: Dessin de l'organisation de Fraise (par analogie au fraisier)

La technologie Fraise repose sur l'interconnexion de cartes programmables nommées **fruits** par l'intermédiaire d'un **bus**, et le pilotage de ces cartes par le biais d'avatars les représentant sous forme de **patchs** Pure Data dans l'ordinateur, lui-même connecté au bus par un **pied** USB.

1. Le bus Fraise

Deux niveau du bus sont définis, un niveau de **courte distance** utilisant du câble « nappe », et un niveau **longue distance** (protocole RS485) qui utilise du câble téléphonique associé aux connecteurs RJ11 (appelés aussi 6P4C).

Une carte spéciale appelée **Fraiseiver** autorise le passage d'un niveau à l'autre. Un ensemble local de fruits et éventuellement d'un Fraiseiver, reliés ensemble par une « nappe », est surnommé une **grappe** ; les grappes, par l'intermédiaire des Fraiseivers, sont reliées les unes aux autres par des câbles téléphoniques.

La communication est du type série asynchrone half-duplex sur 9 bits, à la vitesse de 250 kbit/s.

Le bus longue distance 4 fils (connecteurs RJ11) transmet les signaux symétriques A et B du RS485, ainsi que la masse et une alimentation 12V ; sa longueur maximum cumulée est d'environ **400 mètres**. Il peut connecter jusque 32 grappes.

La nappe 10 fils transporte deux niveau d'alimentations +5V et +12V, la masse et 3 fils permettant la communication (RX, TX, et DRIVE) ; sa longueur ne peut dépasser quelques mètres. Le nombre de cartes contenues dans une grappe ne doit pas excéder la dizaine.

L'architecture de Fraise autorise une grande souplesse en ce qui concerne les alimentations électriques, permettant de s'adapter à tous les types de besoins en **consommation de courant** tout en privilégiant la sécurité et la fiabilité de l'installation (par l'**isolation galvanique** des différentes parties). Cet aspect est développé dans le chapitre V.

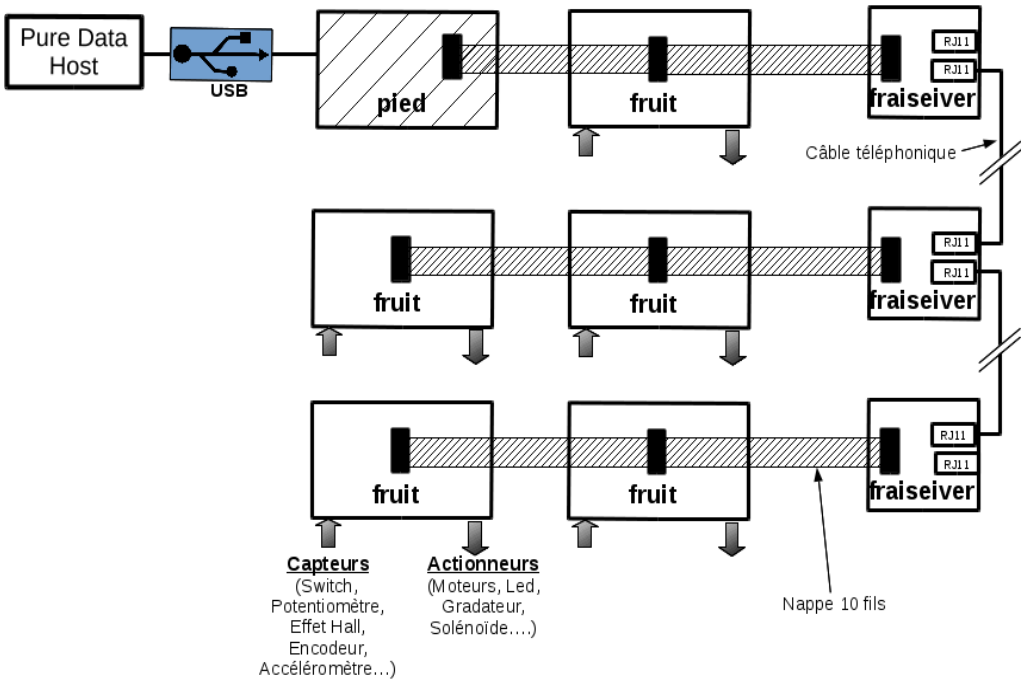


Illustration 2: Topologie du réseau fraise

2. Le logiciel : firmware + patchs

L'objet de Fraise est le dialogue entre deux niveaux de programmation : le **firmware,** qui est le programme du fruit, et le **patch**, programme écrit en Pure Data dans l'ordinateur qui organise la communication entre les fruits et les ressources de l'ordinateur (son, image, automatisation).

Le firmware, programme de bas niveau permettant d'accéder aux capteurs et actionneurs branchés sur le fruit, est écrit en langage C en utilisant un simple éditeur de texte :

```
#define BOARD Versa1
#include <fruit.h>

void setup(void) {
    fruitInit();
    pinModeDigitalOut(LED); // set the LED pin mode to digital out
    digitalClear(LED);      // clear the LED
}

void loop() {
    fraiseService();    // listen to Fraise events
}

void fraiseReceiveChar() // receive text
{
    //switch LED on/off:
    if(fraiseGetChar()=='L') digitalWrite(LED, fraiseGetChar() != '0');
}
```

Illustration 3: Exemple de firmware

Pure Data exécute le patch de l'application (celui que l'utilisateur écrit pour réaliser son projet) intégrant les patchs du pied et des fruits. Le patch d'un fruit prend en charge la compilation de son firmware et son transfert vers le fruit.

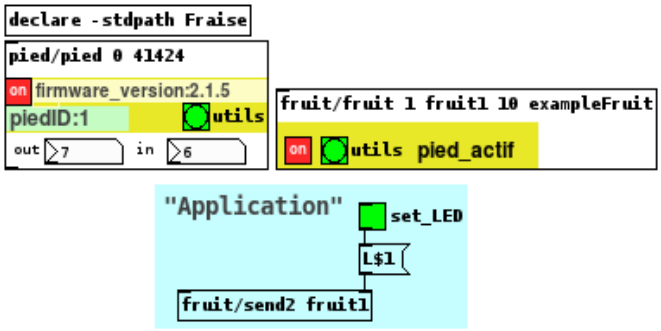


Illustration 4: Exemple de patch

Illustration 5: Fenêtre "utils" du fruit, permettant la compilation et le chargement du firmware

Pure Data intègre ainsi dans un même environnement :

- le pilotage du bus Fraise
- le développement du firmware
- le développement de l'application
- l'exécution de l'application
- le pilotage du firmware

Sans devoir arrêter puis relancer l'application, on peut agir sur la façon dont les informations sont envoyées et traitées par les deux parties : le fruit et l'ordinateur.

Enfin, étant donné que les processus de compilation et de transfert du firmware ainsi que le pilotage du réseau sont pratiquement entièrement écrits en Pure Data, le portage du système vers les différents systèmes d'exploitation (Linux, Mac, Win…) est largement facilité.

L'installation de Fraise est documentée ici :

https://github.com/MetaluNet/Fraise

III. Liaisons électriques

1. Pied et Fruit

Pied et fruit sont reliés au bus courte distance d'une façon identique, que l'on retrouve dans le schéma suivant :

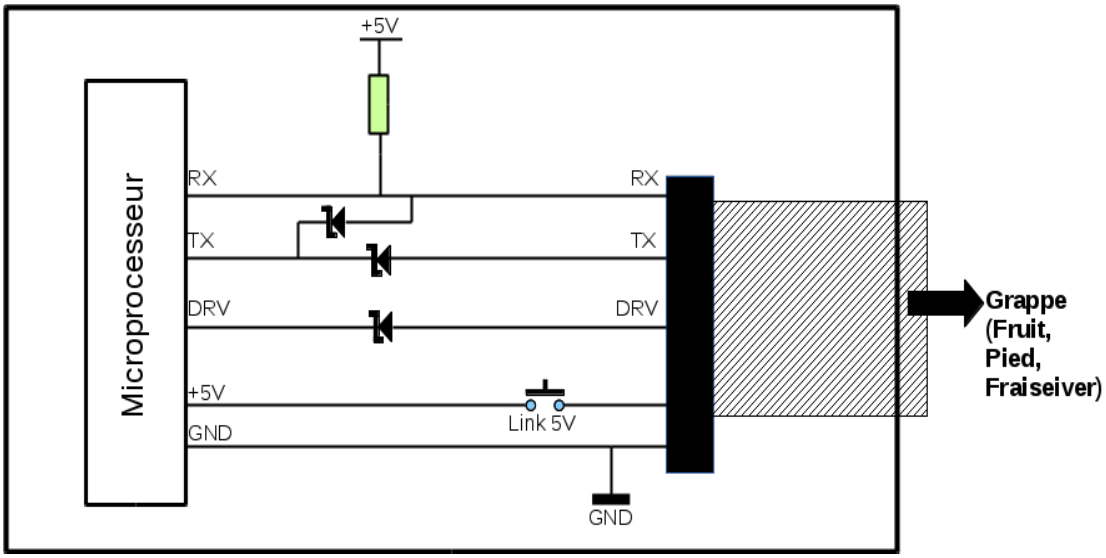


Illustration 6: Liaison électrique fruit/pied au bus courte distance (nappe)

Les trois signaux de la communication sont :

- RX : réception des données
- TX : transmission des données
- DRV : drive, prenant la main sur le bus pour activer la transmission

Une résistance *pull-up* permet de garder un état haut stable sur le signal RX en l'absence de réception.

Une diode permet de recopier la sortie de TX vers le RX (afin que les fruits et le pied réunis dans une même grappe puissent communiquer).

Les deux autres diodes (de TX et de DRIVE) permettent de réaliser une communication sur un seul fil sans risque de conflit de niveaux électriques (bus à « collecteur ouvert »).

Le fruit peut, en option, tirer son alimentation 5V de la nappe, grâce au cavalier link5V ; cette liaison peut également servir au pied pour envoyer le 5V provenant de l'USB sur la grappe locale (voir chapitre V).

2. Fraiseiver

Le Fraiseiver est la carte dédiée à l'extension du réseau fraise ; il permet de réaliser les liaisons longue distance et l'isolation galvanique entre les grappes. Il est l'interface entre les réseaux de courte distance et de longue distance.

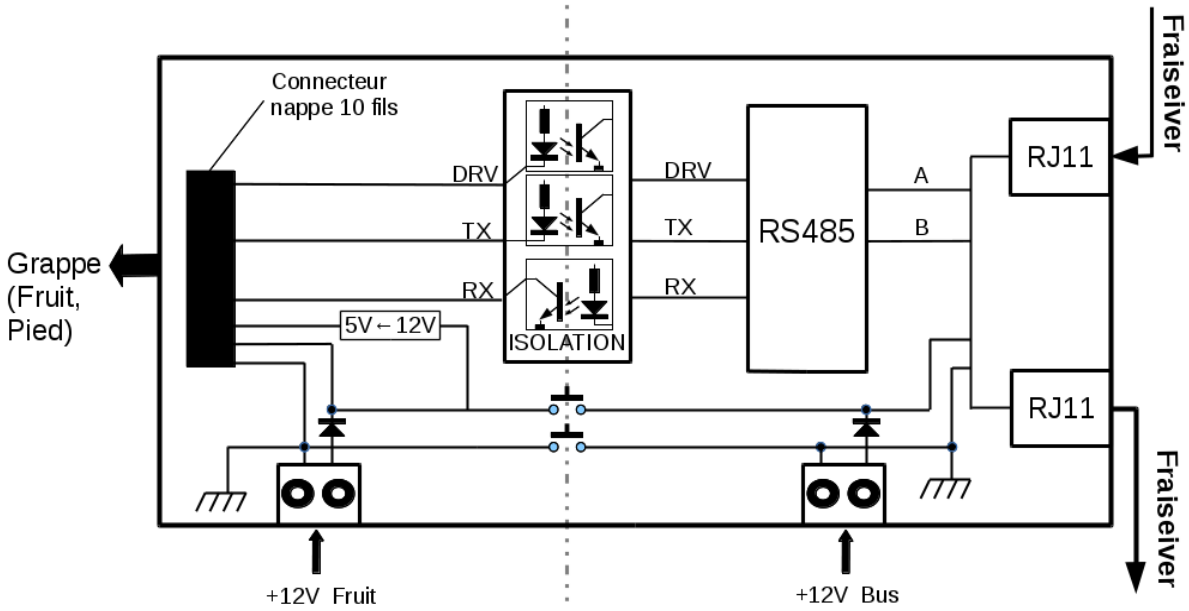


Illustration 7: Fraiseiver : liaison électrique

Le Fraiseiver est donc composé de deux parties principales, qui sont isolées électriquement : d'un coté la communication avec le fruit (nappe 10 fils), et de l'autre la liaison RS485 (connecteurs RJ11) qui relie les Fraiseivers entre eux.

Deux entrées d'alimentation, isolées entre elles (et optionnellement reliées par deux cavaliers), sont également montrées ; leur usage sera détaillé dans le chapitre V.

Par ailleurs, le Fraiseiver fournit au fruit une alimentation 12V et 5V.

3. Evolution

Une évolution prochaine du bus de courte distance prévoit d’utiliser le même conducteur pour les 2 signaux RX et TX. Cette évolution rendra superflue la diode qui, dans le fruit, relie RX à TX. Elle permettra également, dans certains cas, de réduire le nombre de fils du bus courte distance à 4 : GND, 12V, RX/TX, DRV. Cependant le fil 5V reste nécessaire dans le cas d’un fruit alimenté par l’USB (voir chapitre V paragraphe 1).

Une autre évolution prévoit de supprimer la diode du fruit qui relie le signal DRIVE(DRV) à la sortie correspondante du microcontrôleur, puisque cette fonction (interdire à la sortie de fournir du courant) peut-être facilement programmée dans la bibliothèque firmware Fraise.

IV. Conventions de câblage des connecteurs

Fraise standardise la câble de plusieurs câbles et connecteurs : la nappe 10 fils servant à relier les éléments d'une grappe, et le câble téléphonique reliant entre eux les Fraiseivers.

Le brochage des ports d'entrée/sortie élémentaires des fruits (connecteurs « IOP ») est également défini.

1. Nappe 10 fils

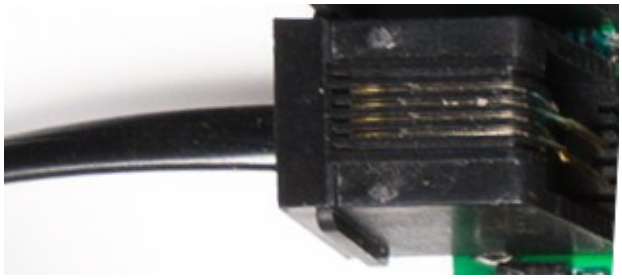
Le câble est une nappe (ou limande) d'espacement 1,27 mm. Le connecteur utilisé est le HE10 à 10 contacts. La carte est munie du connecteur mâle, et le connecteur femelle est serti sur la nappe, ce qui permet à celle-ci d'en recevoir plusieurs aux positions souhaitées.

1	DRV
2	+12V
3	TX
4	+12V
5	RX
6	+12V
7	+5V
8	GND
9	GND
10	GND

2. Câble téléphonique / connecteur RJ11

Le câble est de type plat à 4 conducteurs. Le connecteur est le RJ11 (appelé aussi RJ14 ou 6P4C), femelle sur la carte et mâle en bout de câble.

1	GND
2	RS485-A
3	+12V
4	RS485-B (A inversé)



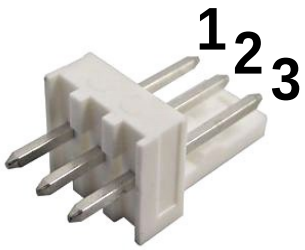
3. Entrée/sortie « IOP »

Chaque « patte » (ou *pin*) du microcontrôleur du fruit peut être connectée à un port d'entrée-sortie élémentaire, dit IOP (i/o port) ; sa fonction dépend de la façon dont il est configuré par le firmware.

Le connecteur choisi est le type KK à 3 contacts d'espacement 2,54mm entre les contacts. Il fournit, en plus du signal du port lui-même (sur 1 fil), les bornes d'alimentation GND (0V) et +5V.

Le connecteur mâle (muni d'un détrompeur) est soudé sur la carte, le femelle serti en bout de câbles.

1	Signal d'entré/sortie
2	+5V
3	GND



V. Les différents modes d'alimentation

Fraise permet une grande souplesse au niveau des raccordements électriques, s'adaptant aux besoins de l'application réalisée. Les cartes demandant peu de puissance peuvent être alimentées directement par l'USB, ou par le bus Fraise longue distance ; une puissance plus important nécessite au contraire une alimentation extérieure au bus, et si possible isolée de celui-ci par sécurité.

1. Auto alimentation via USB

Dans ce cas le fruit est alimenté directement par l'ordinateur via l'USB ; cela implique que le courant qu'il consomme soit faible (moins de 400mA), et qu'il se trouve à proximité du pied, et donc de l'ordinateur.

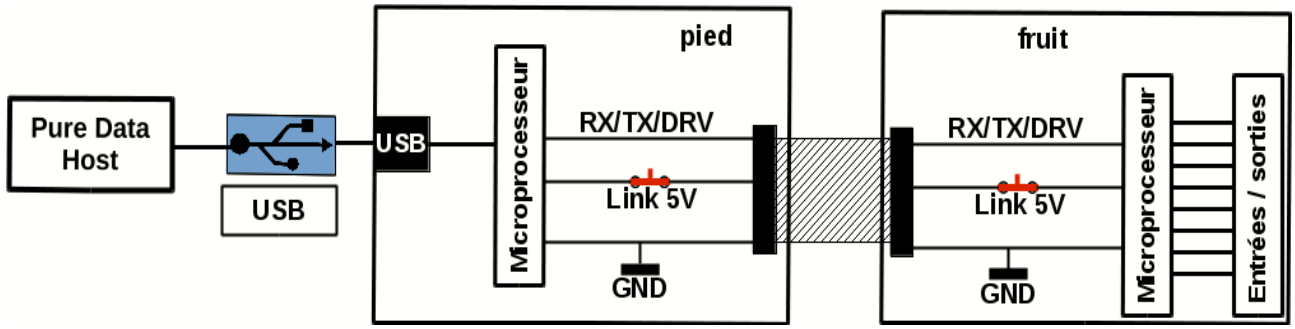


Illustration 8: Alimentation par l'USB

2. Alimentation masse commune USB

Ici le fruit nécessite plus de puissance (par exemple parce qu'il commande un moteur ou un éclairage) , et une alimentation 12V est ajoutée.

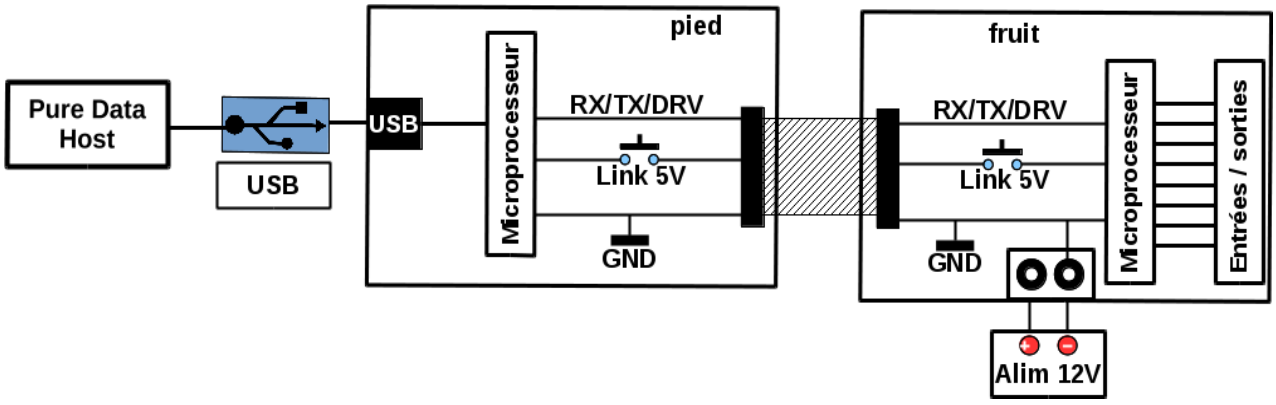


Illustration 9: Alimentation masse commune USB

3. Alimentation par le bus Fraise

Lorsque l'on doit connecter un fruit éloigné de l'ordinateur, il est nécessaire d'utiliser le bus longue distance, et donc au moins 2 Fraiseivers. Ce bus doit être alimenté (12V) : au moins l'un des Fraiseivers sera alimenté du côté « bus ». Cette alimentation peut également être utilisée par un fruit si sa consommation est modérée (moins de 1A).

Deux domaines électriques sont donc isolés :

- le domaine de l'ordinateur, de l'USB et du pied
- le domaine du bus et du fruit.

Si un problème électrique advient sur le domaine du bus et du fruit (court-circuit, sur-tension), l'ordinateur ne peut donc être touché.

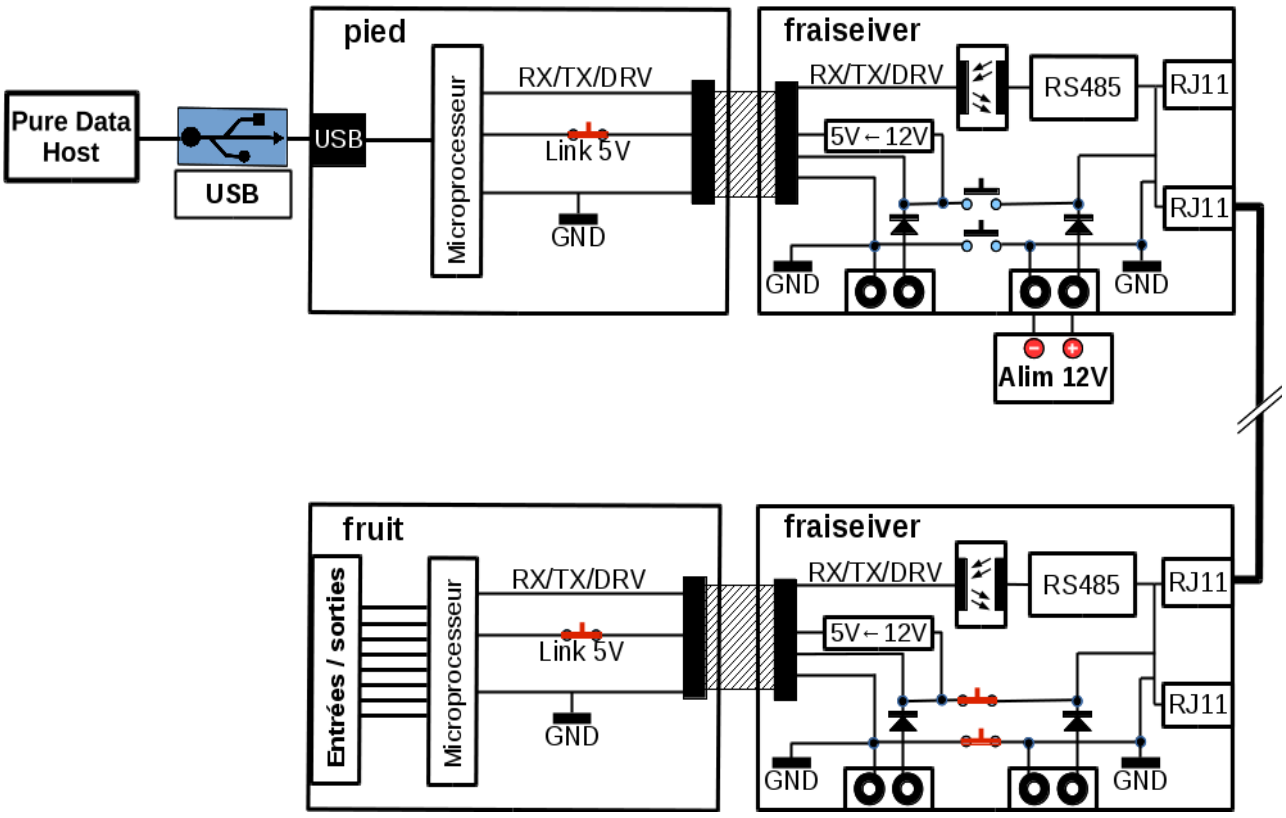


Illustration 10: Alimentation par le bus fraise

Il est possible de réalimenter le bus sur un Fraiseiver éloigné, si la tension disponible à cet endroit est insuffisante du fait de la distance et/ou de la consommation électrique des fruits.

4. Alimentation isolée du bus Fraise

Enfin, lorsque le dispositif demande une puissance importante et qu'il est situé à distance, on peut alimenter localement la grappe comme sur le schéma ci-dessous, tout en garantissant une sécurité grâce à l'isolation galvanique apportée par le Fraiseiver.

Trois domaines électriques distincts sont alors isolés galvaniquement entre eux, apportant ainsi une sécurité maximale et une immunité aux perturbations électriques :

- le domaine de l'ordinateur, de l'USB et du pied
- le domaine du bus longue distance
- le domaine du fruit

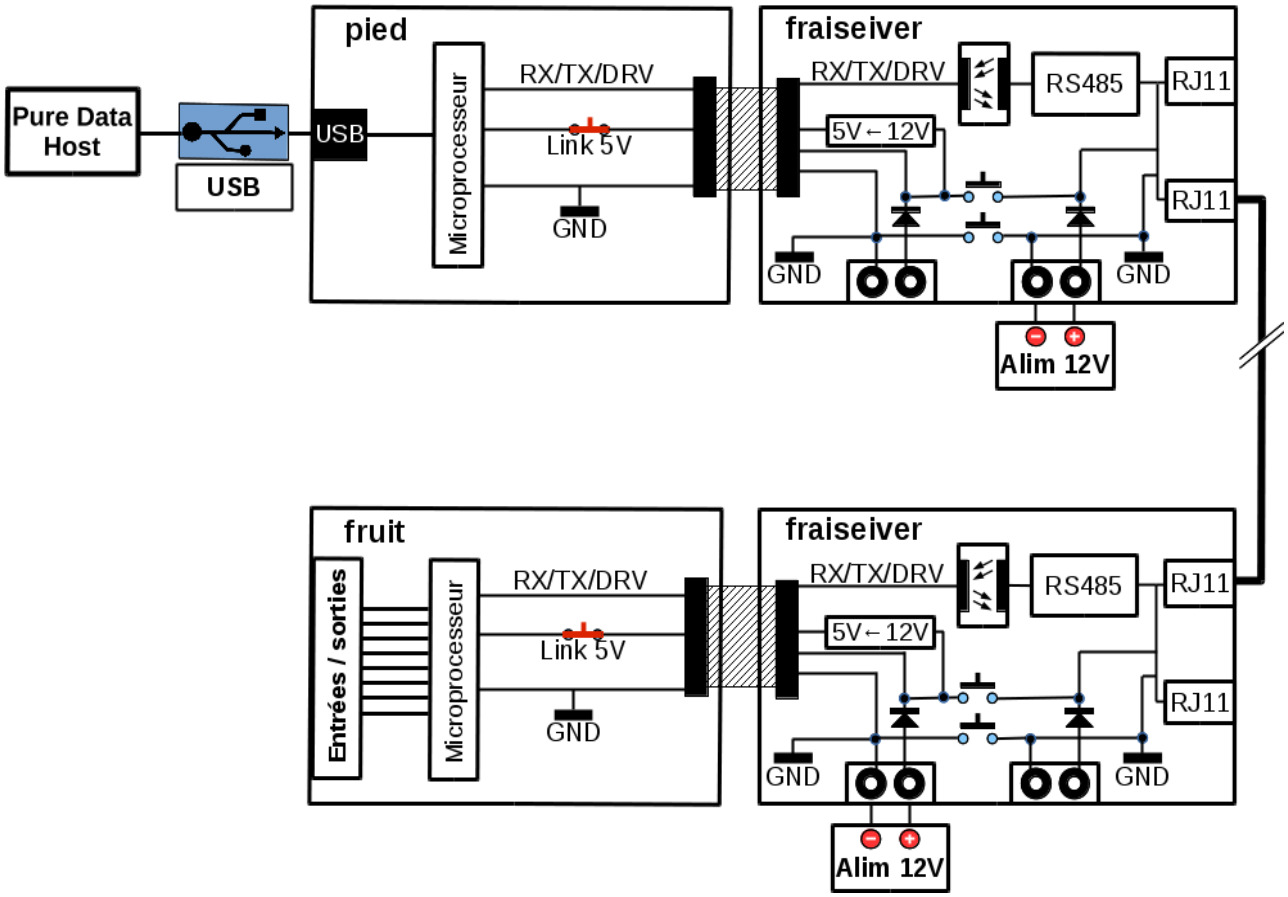


Illustration 11: Alimentation isolée du bus fraise

VI. Architecture logicielle

Ce chapitre explique l'assemblage des différents niveaux logiciels qui composent Fraise : le **bootloader**, le **firmware**, le programme du **pied**, et l'environnement de développement et d'exécution dans **Pure Data**.

1. Bootloader

Les fruits sont pré-programmés (« d'usine ») avec un petit programme, appelé **bootloader**, qui restera inchangé durant toute la vie de la carte. Ce programme est chargé du nommage du fruit et de la programmation du firmware dans la mémoire du microcontrôleur, et il est doté à cet effet d'un protocole de communication minimale avec l'ordinateur.

Au démarrage du fruit, et en l'absence de communication standard sur le bus, le bootloader reste actif pendant 4 secondes, avant de lancer l'exécution du firmware ; si au contraire une communication normale est détectée, le firmware est lancé immédiatement.

2. Nommage, programmation, adressage

Avant toute opération, le fruit doit être nommé : par l'intermédiaire du bootloader, l'ordinateur lui assigne un nom unique (chaîne de maximum 16 caractères), qui servira ensuite pour l'identifier. Par sécurité, cette opération nécessite que le fruit soit connecté seul sur le réseau Fraise, afin d'éviter de donner le même nom à plusieurs fruits. Ce nom est stocké dans la mémoire EEPROM du microcontrôleur.

Ensuite le firmware peut être transféré dans la mémoire du fruit, toujours par le biais du bootloader.

Il est enfin nécessaire d'assigner au fruit une adresse (entre 1 et 126), qui est l'identifiant utilisé dans toutes les autres communications (entrée/sortie standard). C'est la bibliothèque Fraise incluse dans le firmware qui reçoit automatiquement la nouvelle adresse et la stocke dans la mémoire EEPROM.

Toutes ces opérations sont réalisées par le biais de l'interface Pd (voir plus bas).

3. Firmware

a. Les fichiers : config.h et main.c

Le code source du firmware est composé de deux fichiers textes :

- **config.h** : il sert à configurer les modules utilisés, par exemple à régler le nombre maximum d'entrées analogiques à considérer.
- **main.c** : il contient véritablement le firmware, qui initialise le fruit, acquiert et traite les informations provenant des capteurs, envoie ces informations à l'ordinateur, et réagit aux messages provenant de ce dernier.

b. Les fonctions de base : setup() et loop()

Le fichier **main.c** doit tout d'abord définir le type de carte utilisée, par exemple :

```
#define BOARD Versa1
```

Il doit ensuite définir au moins deux fonctions (reprenant la convention Processing/Arduino) :

`void setup(void)` : c'est la fonction qui initialise le fruit (en appelant « fruitInit() ») et les modules (par exemple : « analogInit() »). Elle n'est appelée qu'une seule fois, au démarrage du firmware.

`void loop(void)` : cette fonction est appelée en boucle et réalise les opérations définies par le programmeur. Elle doit appeler les fonctions de service de l'interface Fraise « fraiseService() » et des modules utilisés, comme « analogService() ».

c. Envoi et réception de messages

Deux types de messages sont définis par le protocole Fraise :

- les messages contenant des données brutes, c'est à dire une suite d'octets de n'importe quelle valeur. Ces messages sont envoyés grâce à la fonction « fraiseSend(buf,len) » (le premier octet du buffer *buf* doit être le caractère 'B'), et reçus par la fonction « fraiseReceive() », qui doit dans ce cas être définie par le programmeur dans main.c.

- les messages de type texte (codage ASCII), envoyés par la fonction standard « printf » (le premier octet de la chaîne doit alors être le caractère 'C'), et reçue par la fonction « fraiseReceiveChar() », à définir dans main.c.

d. Entrées / sorties

Le rôle essentiel du firmware est de diriger les *pins* d'entrée-sortie, c'est à dire les « pattes » du microcontrôleur, accessibles par des connecteurs. Ces *pins* peuvent en effet avoir plusieurs fonctions (certaines fonctions ne sont disponibles que sur certaines *pins*) :

- Sortie numérique : la *pin* impose un potentiel de 0V (« 0 ») ou 5V (« 1 »). Elle peut servir par exemple à allumer une LED.
- Entrée numérique : elle mesure la tension électrique, et lui attribue la valeur 0 ou 1 selon sa valeur, selon que la tension est inférieure ou supérieure à un certain seuil. Elle peut par exemple mesurer l'état d'un interrupteur.
- Entrée analogique : elle mesure la tension électrique, et lui attribue une valeur entre 0 et 1023. Elle peut mesurer la position d'un potentiomètre.
- Port série de différentes sortes : peut réaliser une entrée ou une sortie série, par exemple de type DMX512, MIDI, I2C, SPI...
- PWM : pour graduer l'intensité d'une lampe ou la vitesse d'un moteur.

La bibliothèque standard Fraise définit les fonctions suivantes pour configurer les *pins* :

pinModeDigitalIn(pin) : entrée numérique
pinModeAnalogIn(pin) : entrée analogique
pinModeDigitalOut(pin) : sortie numérique
pinModeAnalogOut(pin) : sortie PWM

Les fonctions suivantes agissent sur les *pins* ainsi configurées :

digitalRead(pin) : lire l'état de l'entrée numérique
digitalWrite(pin, value) : changer la valeur de la sortie numérique
analogWrite(pin, value) : changer le rapport cyclique de la sortie PWM

Le nom des *pins* est défini pour chaque carte. Par exemple la Versa1.0 définit les connecteurs : K1, K2, … K12.

Les fonctions plus élaborées sont prises en charge par des modules (voir plus bas).

e. Interruptions

Du fait de l'architecture matérielle du microcontrôleur utilisé (PIC18F26K22), deux niveaux de priorité sont définis pour la gestion des interruptions matérielles. La communication Fraise utilise les interruptions de basse priorité, laissant libre le niveau de haute priorité et permettant ainsi de réagir à des événements (changement de niveau sur certaines *pins*, déclenchement d'un *timer*…) avec un temps de latence inférieur à la microseconde.
Le programmeur peut définir le comportement en cas d'interruption dans les deux fonctions suivantes :
void highInterrupts()
void lowInterrupts()

f. Gestion du temps

La bibliothèque Fraise garde le compte du nombre de microsecondes écoulées depuis le lancement du firmware, qu'on peut consulter grâce à la fonction :
time(void), ou timeISR() si on l'appelle depuis une fonction de gestion des interruptions matérielles.

Elle fournit également un mécanisme de compte à rebours (délai) par les fonctions :
delayStart(delay, micros) : initialisation d'un délai
delayFinished(delay) : retourne 1 si le temps est écoulé, 0 sinon.

g. EEPROM

Le microcontrôleur contient une zone mémoire spéciale permettant de mémoriser des données durablement. La bibliothèque Fraise (via le module **eeparams**) permet au programmeur de définir une fonction centrale pour déclarer les données à mémoriser :
EEdeclareMain()

Cette fonction peut contenir des déclarations de paramètres :
EEdeclareChar(unsigned char *data) : déclaration d'un entier de 1 octet (0 à 255)
void EEdeclareInt(unsigned int *data) : déclaration d'un entier de 2 octets (0 à 65 535)
void EEdeclareLong(unsigned long *data) : déclaration d'un entier de 4 octets (0 à 4 294 967 295)

Dans le setup(), on peut rappeler la valeur des paramètres stockés dans l'EEPROM avec :
EEreadMain();

Enfin, pour écrire la valeur actuelle des paramètres dans l'EEPROM, appeler :
EEwriteMain();

h. Modules

De nombreuses bibliothèques (modules) sont disponible en fonction des besoins de l'application, comme :

- **analog** : gestion des entrées analogique / mesure d'effet capacitif.
- **switch** : automatisation de la surveillance de l'état de commutateurs.
- **dcmotor** : gestion de moteur à courant continu ; intègre la régulation vitesse/position et un générateur de rampe (accélération / vitesse maximale) pour la commande de position.
- **servo** : commande de servomoteurs.
- **dimmer** : gradation de lampe à incandescence en courant alternatif.
- **dmx** : générateur de trame DMX512, pour commander des gradateurs de lumière.
- **dmx_slave** : entrée DMX512
- **i2c_master** : dialogue avec des périphériques I2C
- **ADXL345** : dialogue avec accéléromètre ADXL345 ; utilise le module i2c_master.

Ces modules doivent d'abord être déclarés au début du fichier main.c ; par exemple pour utiliser le module analog if faut ajouter la ligne :
#include <analog.h>

Ensuite le module doit être initialisé dans setup() :
analogInit() ;

Suit éventuellement la configuration du module, toujours dans setup() ; par exemple pour sélectionner une entrée analogique :
analogSelect(0,K1) : déclare le connecteur K1 pour le canal d'entrée analogique 0.

Le module fournit le plus souvent une fonction de service, à appeler dans loop() :
analogService() ;

Il peut également mettre à disposition une fonction qui automatise l'envoi des données vers l'odinateur :
analogSend() ; (envoie la valeur des entrées analogiques ayant changé suffisamment depuis leur dernier envoi)
Cette fonction peut être appelée dans loop() au rythme souhaité, par exemple toutes les 5 millisecondes.

Les modules devant recevoir des données de l'ordinateur fournissent une fonction à ajouter à fraiseReceive() ; par exemple la commande des servomoteurs est assuré par la fonction :
servoReceive();

Le module peut nécessiter une gestion des interruptions, par exemple pour utiliser le module servo la fonction suivante doit être appelée par highInterrupts() :

servoHighInterrupt() ;

Enfin le module peut stocker certaines informations dans la mémoire EEPROM, auquel cas une fonction spéciale doit être ajoutée dans EedeclareMain() ; le module analog peut ainsi mémoriser la calibration des entrées analogique, avec la fonction :
analogDeclareEE();

i. Documentation

L'API du firmware Fraise et des modules est documentée en ligne :
http://metalunet.github.io/Fraise-doc/

Par ailleurs, les modules fournissent des programmes et patchs d'exemple, afin d'en faciliter la prise en main.

4. Le programme du Pied

Le programme du pied est « transparent » pour l'utilisateur, qui n'en voit que l'interface Pure Data (voir le paragraphe suivant) ; il réalise l'interface entre la couche USB et la couche Fraise. Il transmet les messages provenant de l'ordinateur (par l'USB) aux fruit, et interroge ces derniers de façon cyclique afin de relayer leurs messages éventuels vers l'ordinateur. Il peut également basculer sur un mode spécialement dédié à la communication avec le bootloader d'un fruit.

5. Interface avec Pd

La bibliothèque Pd de Fraise met à disposition un objet [pied/pied] pour représenter le pied. Cet objet permet en particulier de vérifier que le pied est bien connecté en USB, et d'interroger ou modifier son identifiant : en effet chaque pied est associé à un numéro (en général 1), ce qui permet potentiellement de connecter plusieurs pieds sans risquer de les confondre. Ce dernier cas de figure est néanmoins rare, un seul pied permettant de connecter un nombre de carte largement suffisant pour la plupart des configurations.

Chaque fruit doit être représenté par un autre objet : [fruit/fruit], appelé avec des arguments qui identifient le fruit. Par exemple :

[fruit/fruit 1 monFruit 10 dossierDuProgramme]

identifie un fruit connecté au pied numéro « 1 », de nom « monFruit », d'adresse « 10 » et dont le programme se situe dans « dossierDuProgramme » (dossier qui doit lui-même être contenu dans le même dossier qui le patch de l'application).

L'objet [fruit/send monFruit] permet d'envoyer des messages au fruit « monFruit », tandis que [fruit/receive monFruit] permet d'en recevoir.

Chaque module fournit également, lorsque c’est nécessaire, des objets Pd qui en simplifient l’utilisation.

Exemples :

[analog/parse] permet de traiter les informations du module « analog » chargé de la mesure de tensions analogiques.

[ramp/ramp] donne l’accès au paramétrage du module de génération d’une rampe contrôlée en vitesse et accélération/freinage.

ANNEXE : détail des cartes existantes

6. Versa1.0 (ou Pied)

Le PCB de la carte Versa1.0 peut être câblé soit en version Pied, avec un processeur 18F2455 et un connecteur USB, soit en Fruit (nommé dans ce cas Versa1.0), utilisant un processeur 18F26K22 et proposant 12 entrés/sorties (IOP) et deux connecteur SIL8 auxiliaires.

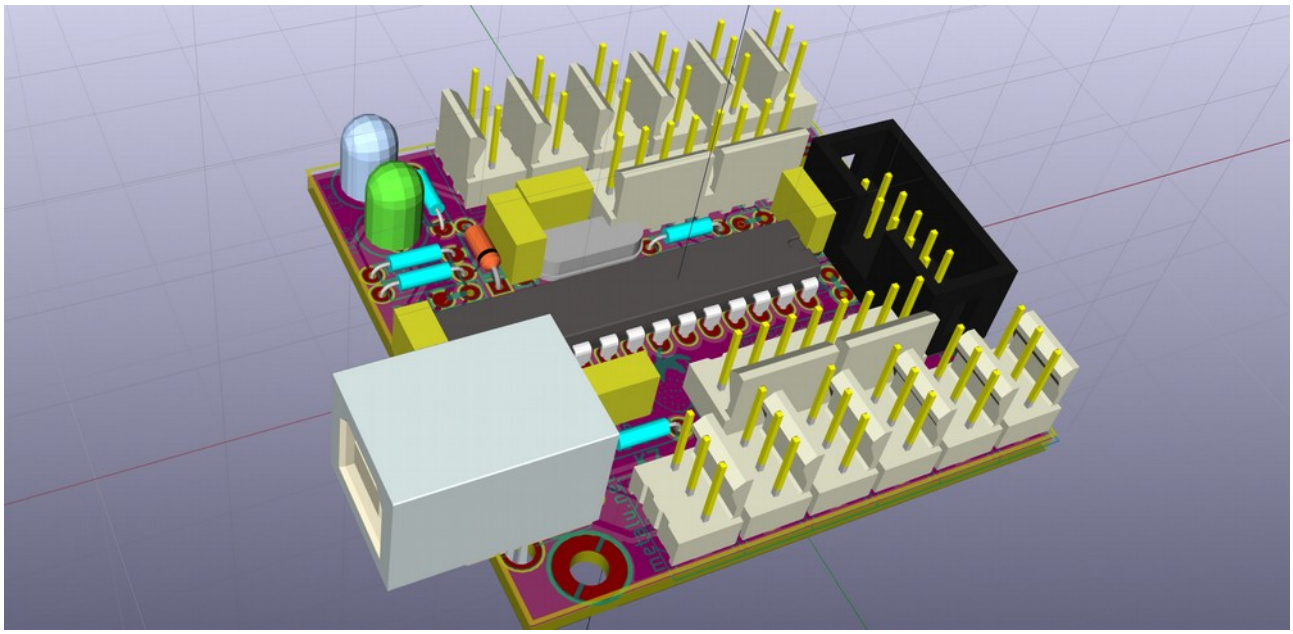


Illustration 12: Vue 3D Versa1.0

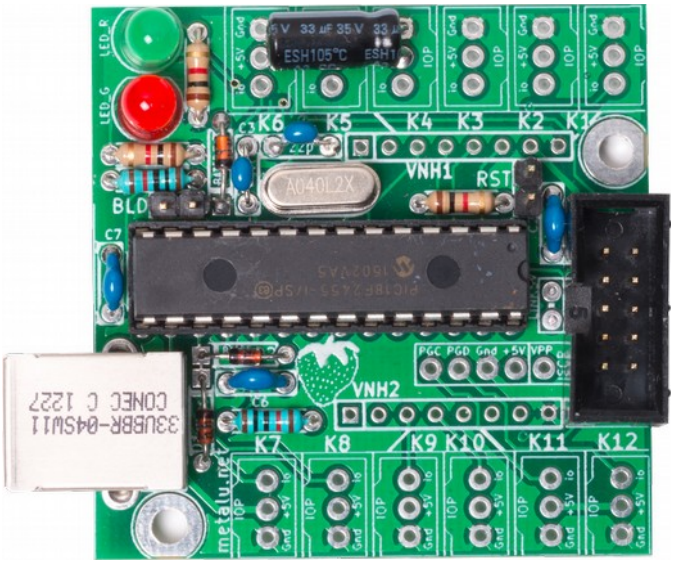


Illustration 13: Photo pied

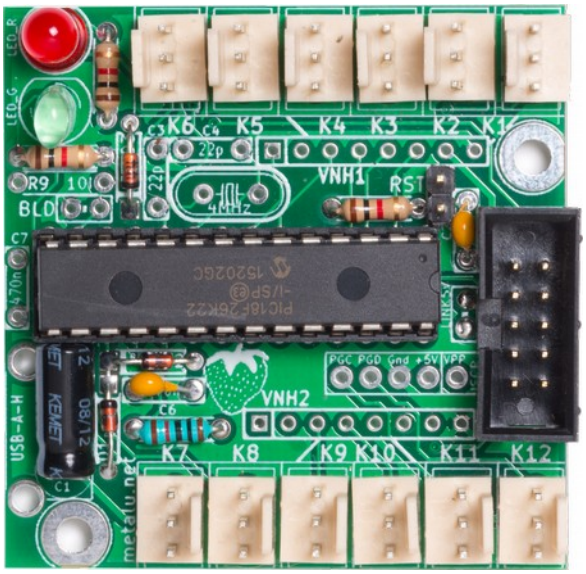


Illustration 14: Photo fruit

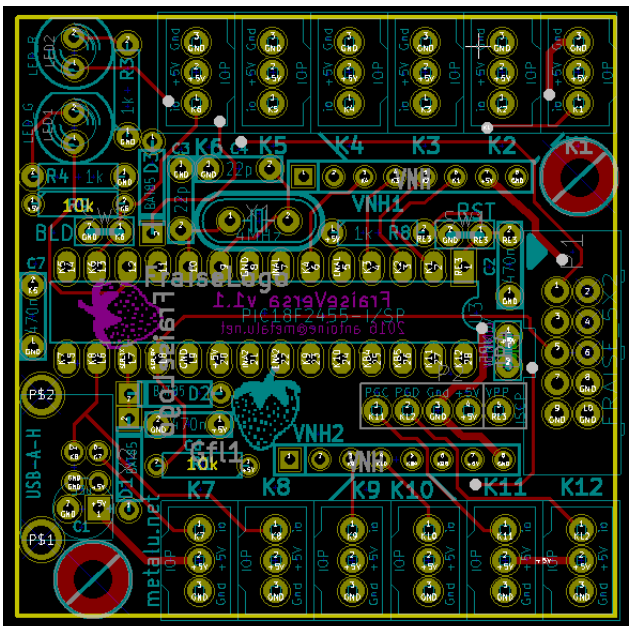


Illustration 15: PCB Versa

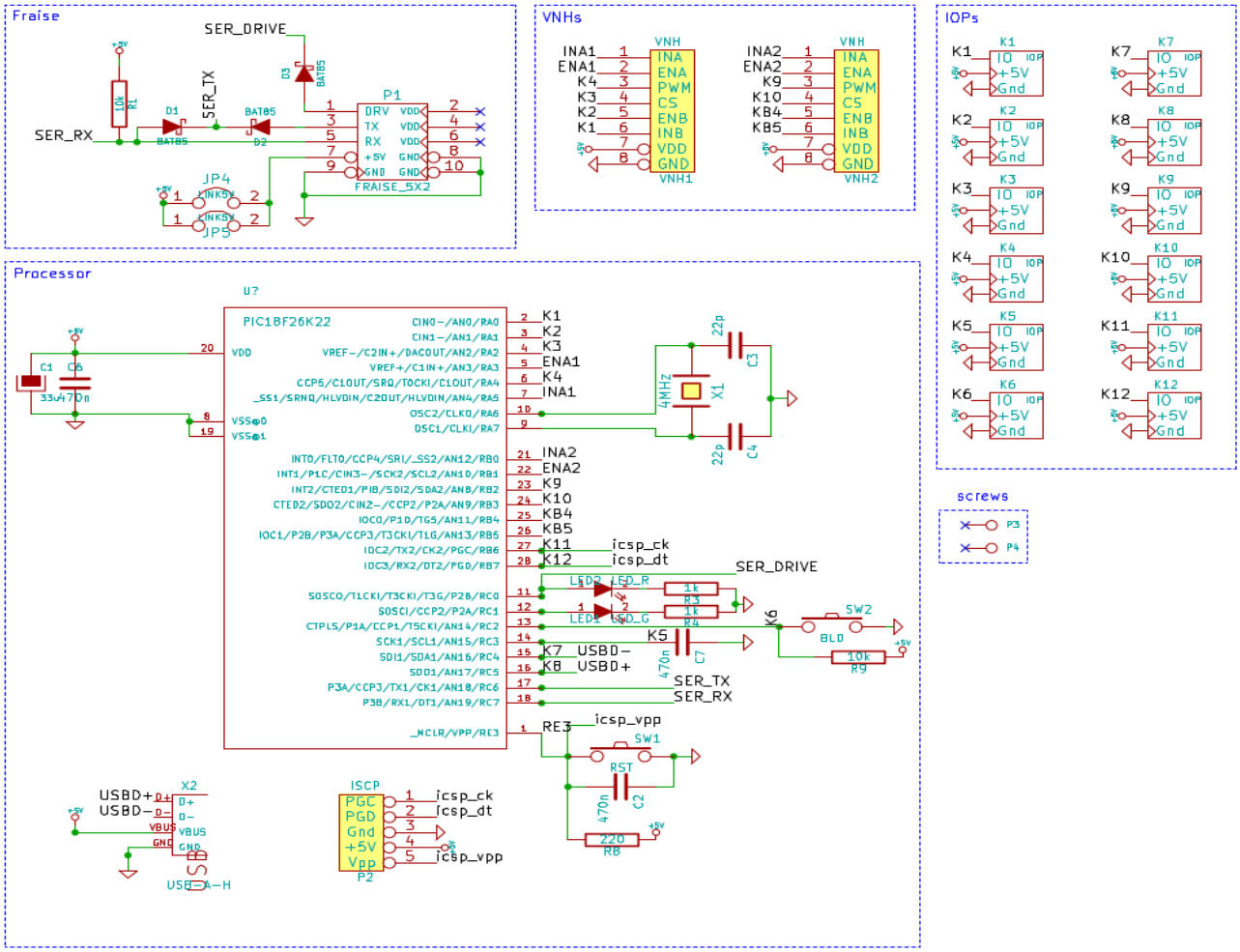


Illustration 16: Schématique Versa

Étude de coût en fonction des principaux distributeur de composants (hors PCB) :

		FARNELL				MOUSER				DIGIKEY				DIGIKEY		
Composants	**QTÉ**	**Ref**	**Prix**	**min**	**Prix total**	**Ref**	**Prix**	**min**	**Prix total**	**Ref**	**Prix**	**min**	**Prix total**	**Prix**	**min**	**Prix total**
PIED																
Résist. 1/4 W	5	Ex :MCF 0.25W 1K	0,023 €	500	0,115 €		0,029 €	100	0,145 €		0,075 €	5000	0,37 €	0,066 €	250	0,330 €
22p	2	K220J15C0GF53L2	0,060 €	500	0,120 €		0,076 €	1000	0,152 €		0,026 €	20000	0,05 €	0,070 €	1	0,140 €
470n	3	C410C474K5R5TA	0,128 €	100	0,384 €		0,118 €	50	0,354 €		0,199 €	100	0,60 €	0,274 €	25	0,822 €
33u	1	EEUEB1V330S	0,090 €	200	0,090 €		0,073 €	500	0,073 €		0,071 €	500	0,07 €	0,080 €	50	0,080 €
Diodes Bat 85	3	BAT85S-TR	0,096 €	500	0,288 €		0,098 €	500	0,294 €		0,044 €	10000	0,13 €	0,045 €	2500	0,135 €
Led V/R	2	HLMP-Y502-F0000	0,115 €	200	0,230 €		0,120 €	100	0,240 €		0,110 €	250	0,22 €	0,103 €	100	0,206 €
USB	1	USB-B-S-RA	0,357 €	100	0,357 €	67068-9000	0,922 €	100	0,922 €	ED2983-ND	0,367 €	100	0,37 €	1,010 €	150	1,010 €
Quartz	1	LFXTAL003074	0,141 €	100	0,141 €		0,124 €	100	0,124 €		0,400 €	100	0,40 €	1,330 €	10	1,330 €
Conn. Nappe	1		1,380 €	100	1,380 €		0,772 €	100	0,772 €		0,687 €	250	0,69 €	1,160 €	250	1,160 €
Jumper	1	10-08-1021	0,161 €	150	0,161 €	22-28-4020	0,064 €	100	0,064 €	22284020	0,086 €	100	0,09 €	1,768 €	100	1,768 €
Microcontrôleur	1	PIC18F2455-I/SP	4,470 €	1	4,470 €		4,440 €	1	4,440 €		3,780 €	1	3,78 €	4,280 €	15	4,280 €
TOTAL					**7,74 €**				**7,58 €**				**6,76 €**			**11,26 €**
FRUIT																
Résist. 1/4 W	5	Ex :MCF 0.25W 1K	0,023 €	500	0,115 €		0,029 €	100	0,145 €		0,075 €	5000	0,38 €	0,066 €	250	0,330 €
470n	2				0,000 €											
33u	1	EEUEB1V330S	0,090 €	200	0,090 €		0,073 €	500	0,073 €		0,071 €	500	0,07 €	0,080 €	50	0,080 €
Diodes Bat 85	3	BAT85S-TR	0,096 €	500	0,288 €		0,098 €	500	0,294 €		0,044 €	10000	0,13 €	0,045 €	2500	0,135 €
Led V/R	2	HLMP-Y502-F0000	0,115 €	200	0,230 €		0,120 €	100	0,240 €		0,110 €	250	0,22 €	0,103 €	100	0,206 €
Connecteurs kk	12	22-23-2031	0,085 €	150	1,020 €		0,131 €	100	1,572 €		0,130 €	100	1,56 €	0,115 €	?	1,380 €
Jumper	1	10/08/21	0,161 €	150	0,161 €	22-28-4020	0,064 €	100	0,064 €	22284020	0,086 €	100	0,09 €	1,768 €	100	1,768 €
Conn. Nappe	1		1,380 €	100	1,380 €		0,772 €	100	0,772 €		0,687 €	250	0,69 €	1,160 €	250	1,160 €
Microcontrôleur	1	PIC18F26K22-I/SP	2,440 €	10	2,440 €		2,550 €	10	2,550 €		2,350 €	25	2,35 €	2,797 €	5	2,797 €
TOTAL					**5,72 €**				**5,71 €**				**5,48 €**			**7,86 €**

Illustration 17: Étude de coût Versa

7. Fraiseiver

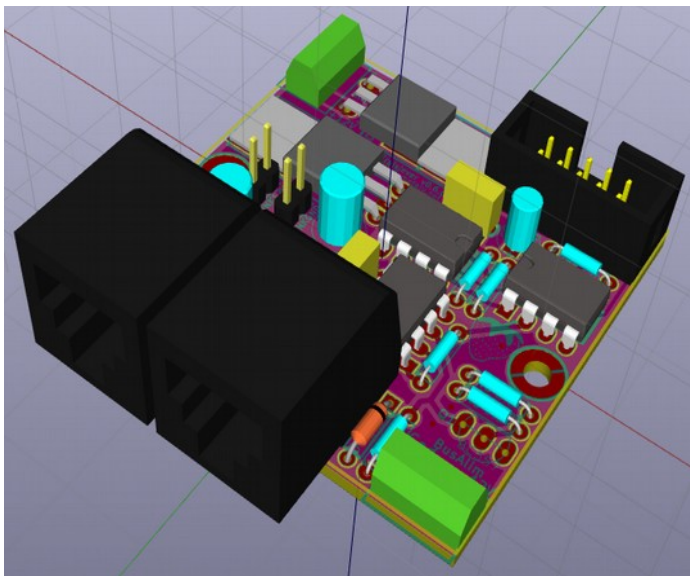


Illustration 18: Vue 3D Fraiseiver

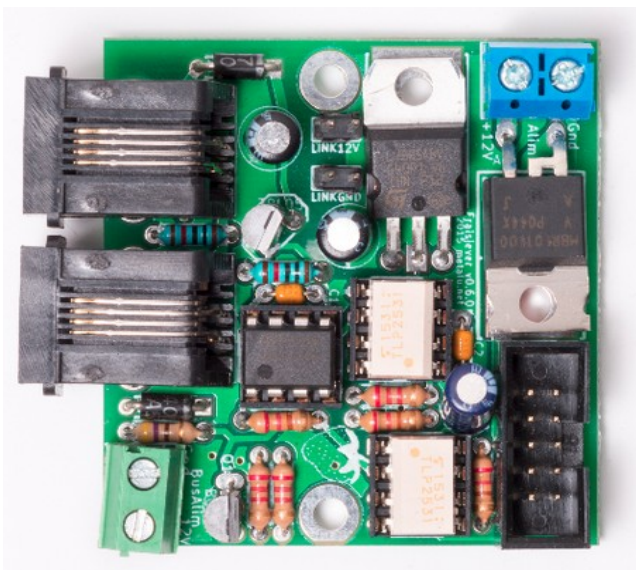


Illustration 19: Photo Fraiseiver

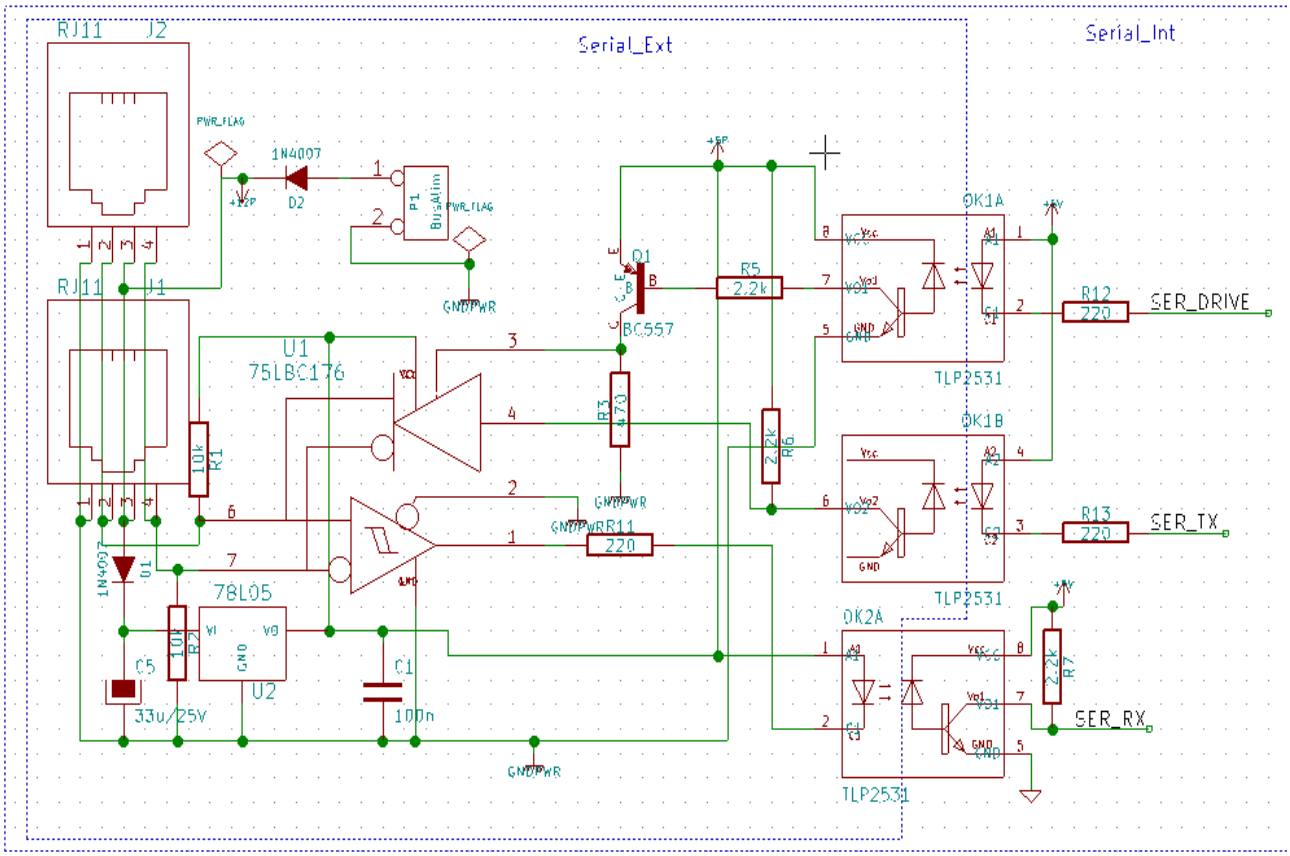


Illustration 20: Schéma Fraiseiver : connexions au bus RS85 et isolation

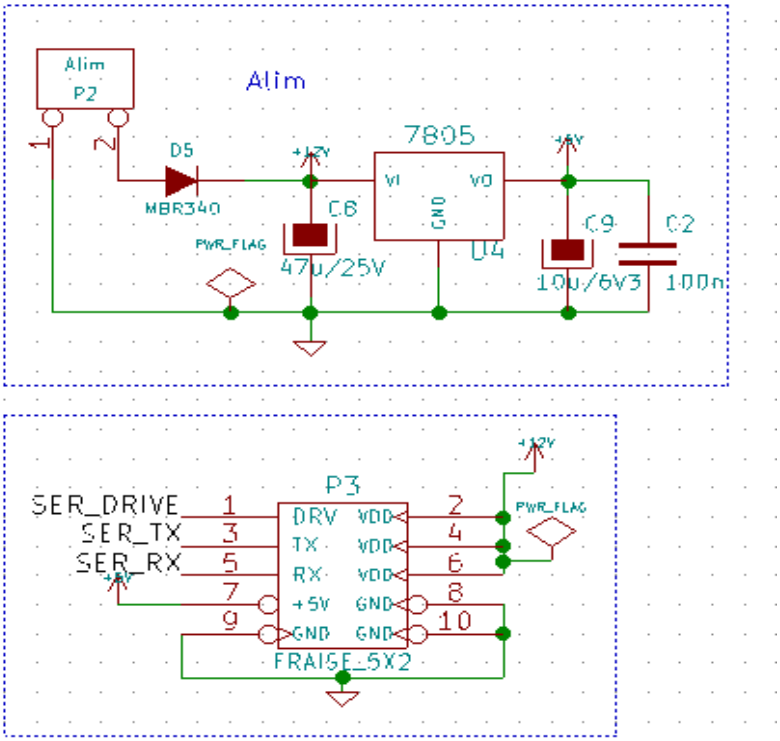


Illustration 21: Schéma Fraiseiver : alimentation et connecteur nappe

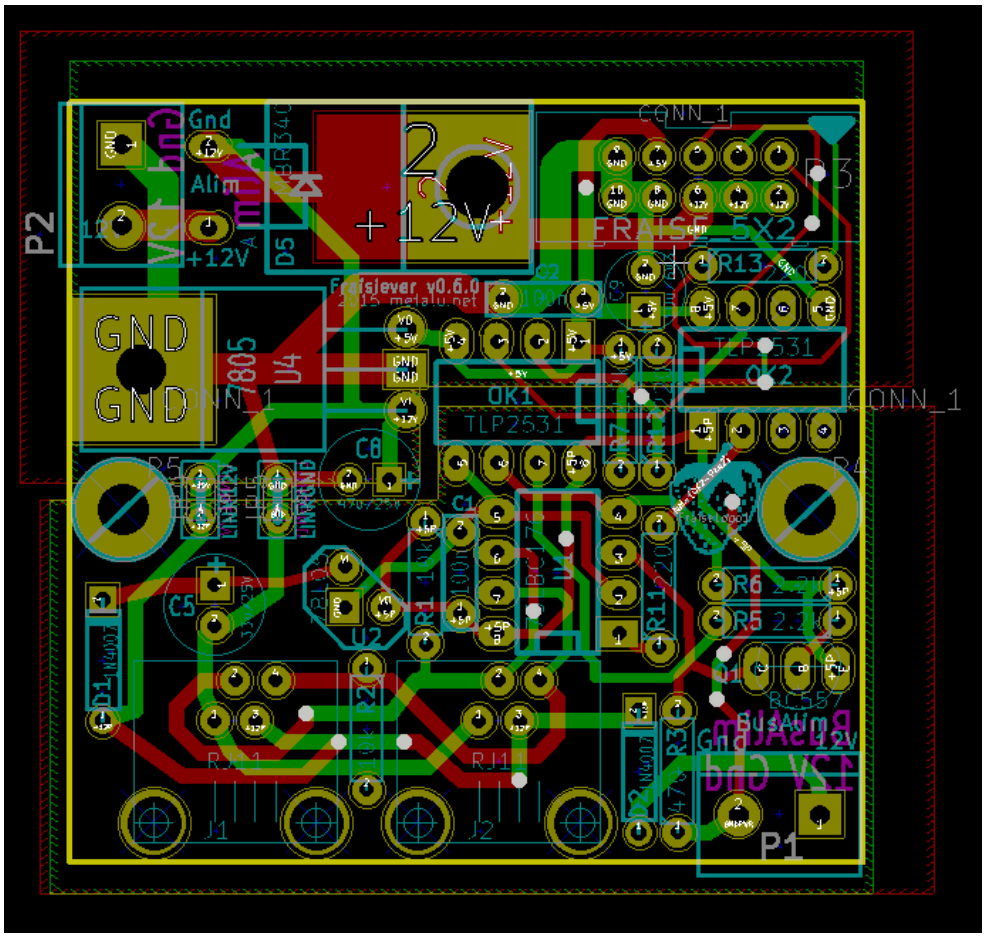


Illustration 22: PCB Fraiseiver

Étude de coût en fonction des principaux distributeur de composants (hors PCB) :

FRAISEIVER		FARNELL				MOUSER			DIGIKEY			RS		
Composants	QTÉ	Ref	Prix	min	Prix total	Prix	min	Prix total	Prix	min	Prix total	Prix	min	Prix total
Résistances 1/4 W	9	Ex : MCF 0.25W 1K	0,023 €	500	0,207 €	0,029 €	100	0,261 €	0,075 €	5000	0,68 €	0,066 €	250	0,594 €
Diode de puissance	2	1N4007	0,052 €	100	0,103 €	0,052 €	100	0,104 €	-	-	-	0,052 €	100	0,104 €
33u/25V	1	USR1E330MDD	0,049 €	200	0,049 €	0,082 €	250	0,082 €	0,065 €	100	0,07 €	0,083 €	100	0,083 €
10u/35V	1	ECA-1VM100	0,052 €	250	0,052 €	0,049 €	100	0,049 €	0,066 €	100	0,07 €	0,040 €	200	0,040 €
Régulateur de tension	1	LM78L05ACZ	0,197 €	150	0,197 €	0,106 €	100	0,106 €	0,177 €	100	0,18 €	0,099 €	100	0,099 €
Régulateur de tension linéaire	1	LM7805	0,291 €	100	0,291 €	0,430 €	100	0,430 €	0,580 €	100	0,58 €	0,440 €	10	0,440 €
Optocoupleurs	2	TLP2531	1,010 €	100	2,020 €	0,716 €	100	1,432 €	0,789 €	100	1,58 €	1,424 €	5	2,848 €
Émetteur/Récepteur RS485	1	75LBC176	0,547 €	100	0,547 €	1,960 €	100	1,960 €	1,890 €	100	1,89 €	0,598 €	50	0,598 €
Diode schottky	1	MBR340	0,187 €	100	0,187 €	0,177 €	100	0,177 €	0,243 €	100	0,24 €	0,252 €	50	0,252 €
Connecteurs nappes	1		1,380 €	100	1,380 €	0,772 €	100	0,772 €	0,687 €	250	0,69 €	1,160 €	250	1,160 €
Connecteurs RJ11	2	RJ11	0,643 €	150	1,286 €	0,464 €	100	0,928 €	1,319 €	100	2,64 €	0,670 €	100	1,340 €
Bornier	2	MC24366	0,238 €	100	0,476 €	1,450 €	100	2,900 €	1,410 €	100	2,82 €	1,132 €	75	2,264 €
TOTAL					**6,795 €**			**9,201 €**			**11,42 €**			**9,822 €**

Illustration 23: Étude de coût Fraiseiver

8. 8X2A

La 8X2A est un fruit spécialisé pour la commande de moteur, elle peut avoir deux configurations en fonction du nombre d'actionneurs à piloter : avec un seul pont de puissance ou deux ponts de puissance.

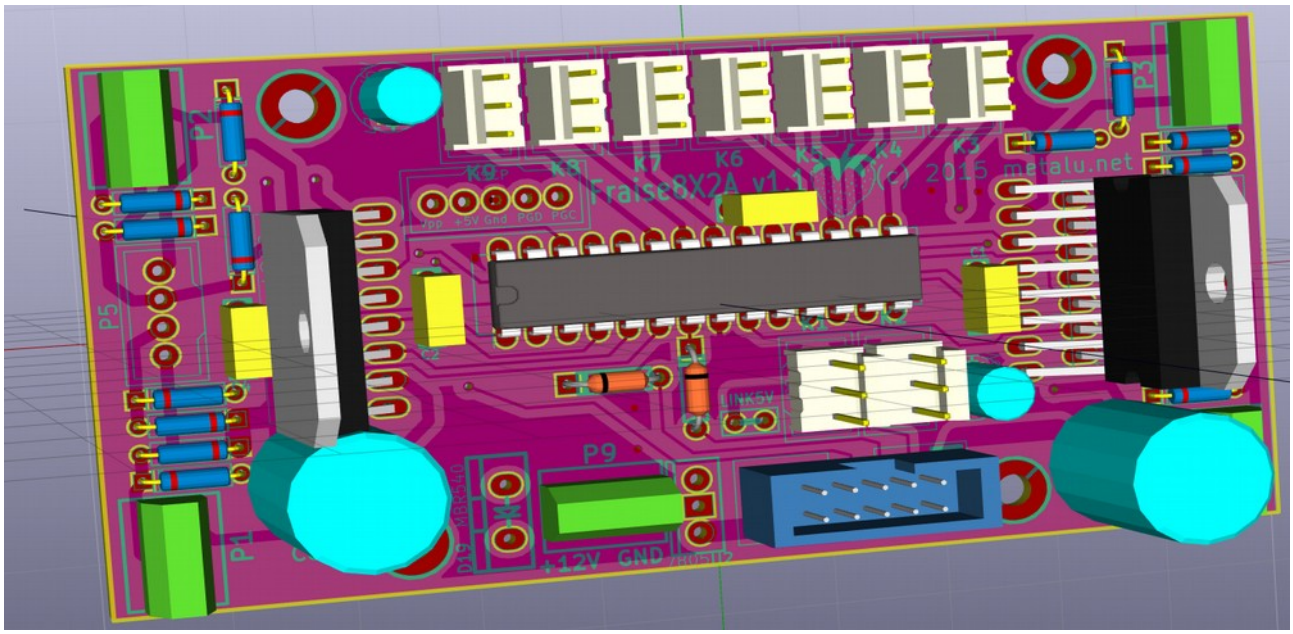


Illustration 24: Vue 3D de la 8X2A avec deux ponts

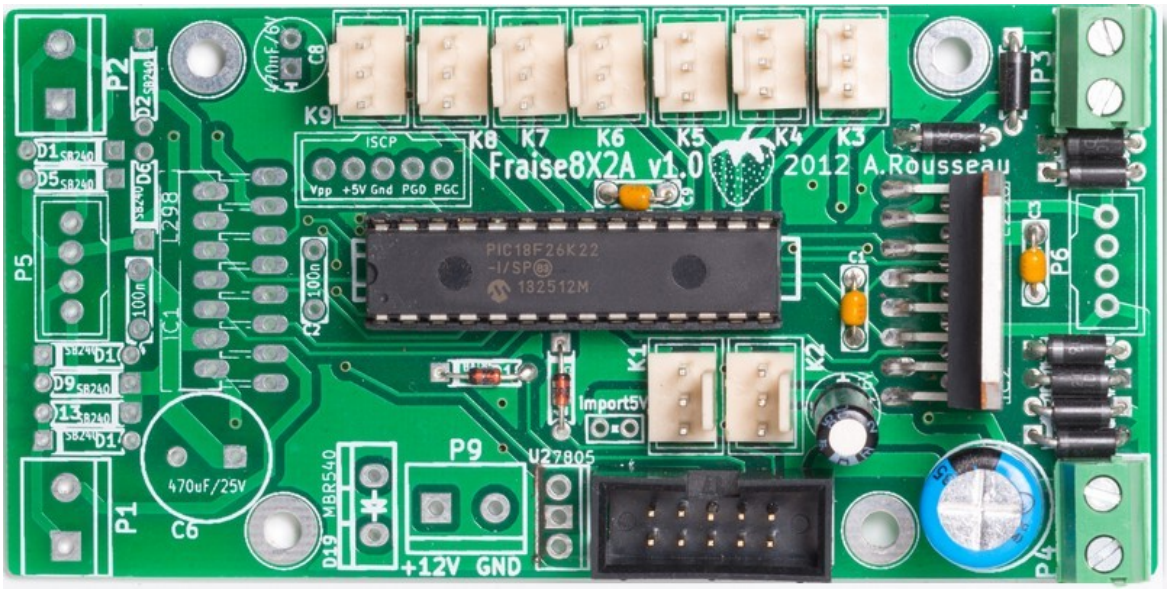


Illustration 25: Photo 8X2A avec un seul pont

Schémas électriques:

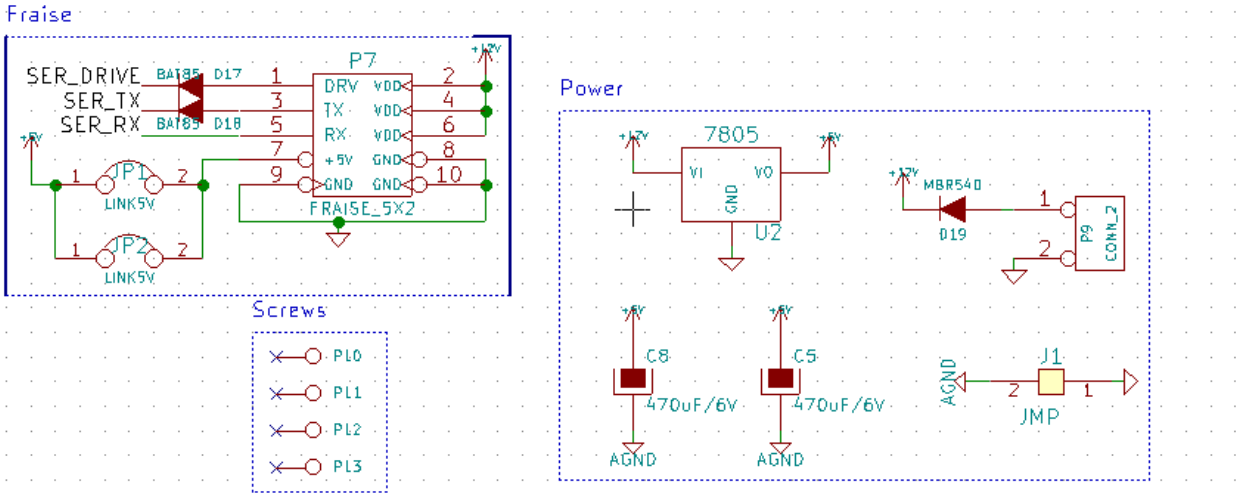


Illustration 26: Schéma 8X2A : alimentation et interface Fraise

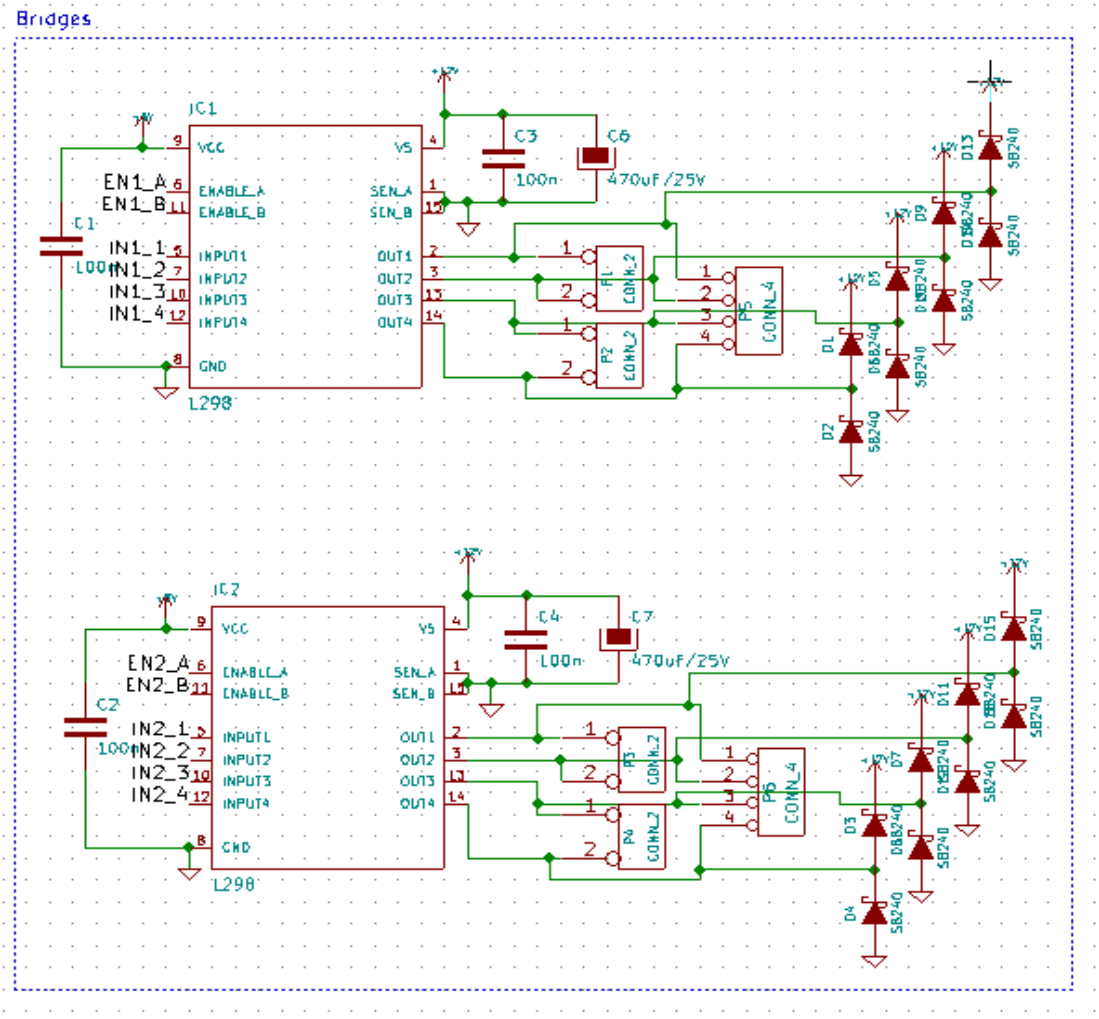


Illustration 27: Schéma 8X2A : ponts de puissance L298

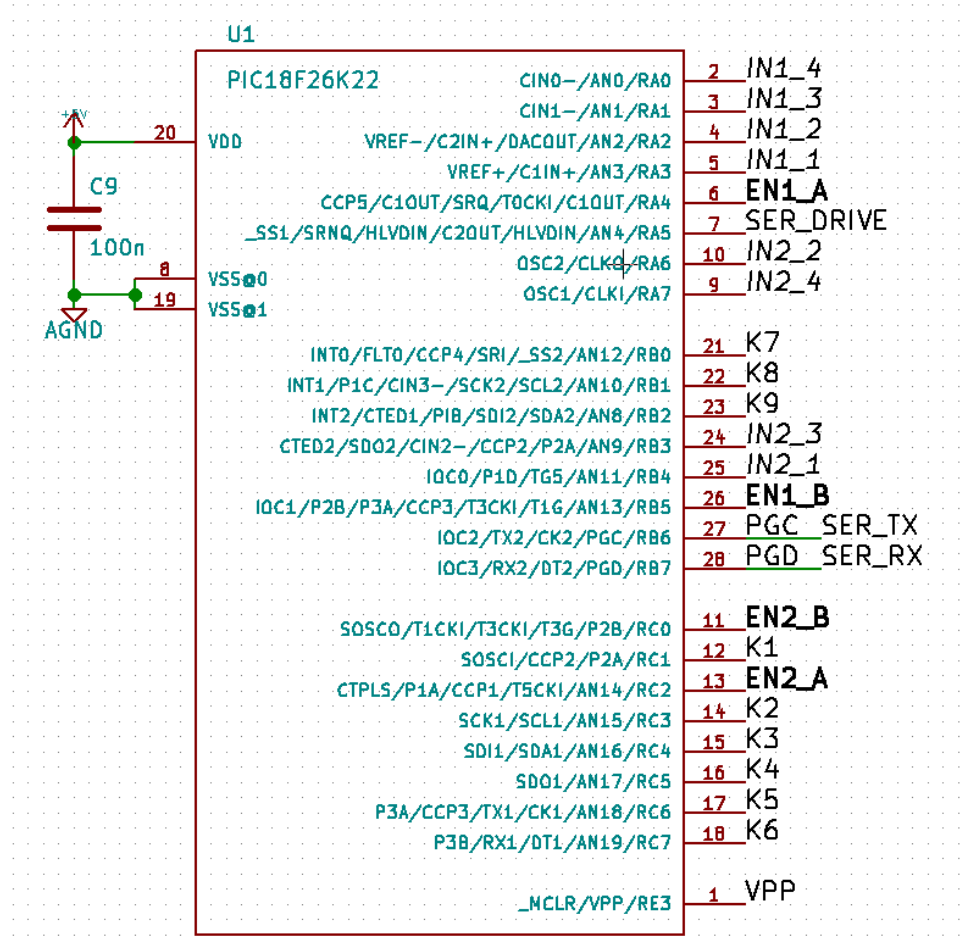


Illustration 28: Schéma 8X2A : connexions microcontrôleur

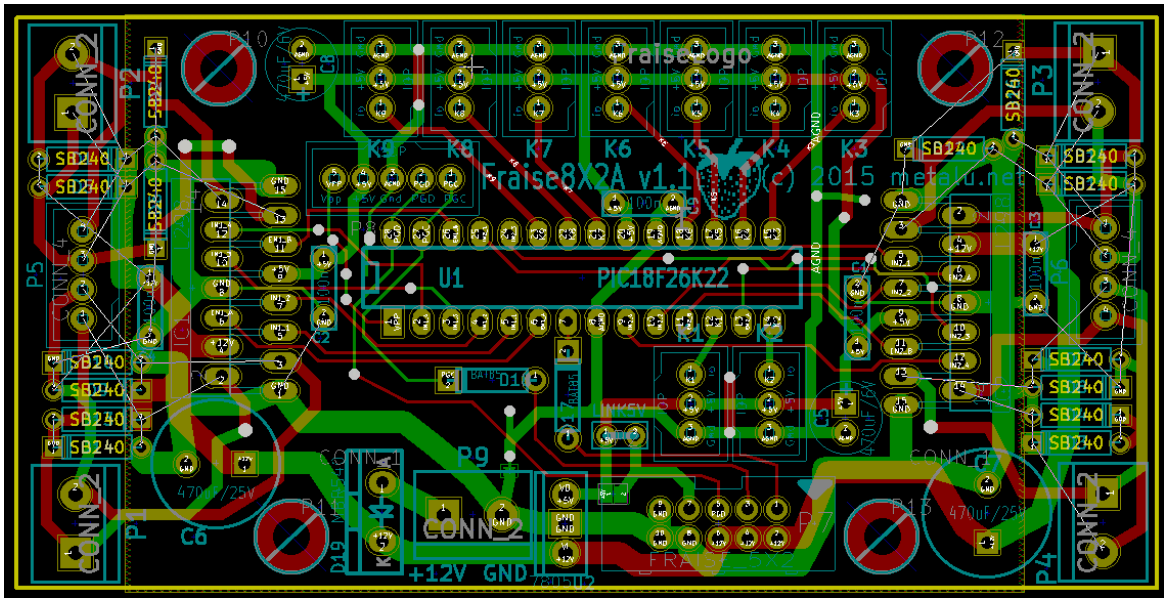


Illustration 29: PCB 8X2A

Étude de coût en fonction des principaux distributeur de composants (hors PCB) :

		FARNELL				MOUSER				DIGIKEY				RS			
Composants	**QTÉ**	**Ref**	**Prix**	**min**	**Prix total**	**Ref**	**Prix**	**min**	**Prix total**	**Ref**	**Prix**	**min**	**Prix total**	**Ref**	**Prix**	**min**	**Prix total**
8X2A/ 1 pont																	
Diodes schottky 2A	8	SB240-E3/54	0,287 €	100	2,296 €		0,190 €	100	1,520 €		0,210 €	100	1,680 €		0,253 €	20	2,024 €
Diodes Bat 85	2	BAT85S-TR	0,096 €	500	0,192 €		0,098 €	500	0,196 €		0,044 €	10000	0,09 €		0,045 €	2500	0,090 €
100n	3	F102K43Y5RP6UK5R	0,154 €	100	0,462 €		0,138 €	100	0,414 €	S102K43Y5PR63K7R	0,127 €	100	0,38 €		0,172 €	25	0,516 €
470n/35v	1	35ZLH470MEFC10X16	0,257 €	100	0,257 €	UBW1V471MHD	1,260 €	100	1,260 €		0,248 €	100	0,25 €		0,346 €	100	0,346 €
Driver moteur	1	L298N	3,060 €	100	3,060 €		2,920 €	100	2,920 €		2,852 €	100	2,85 €		2,490 €	10	2,490 €
Microcontrôleur	1	PIC18F26K22-I/SP	2,440 €	10	2,440 €		2,550 €	10	2,550 €		2,350 €	25	2,35 €		2,797 €	5	2,797 €
Connecteurs nappes	1		1,380 €	100	1,380 €		0,772 €	100	0,772 €		0,687 €	250	0,69 €		1,160 €	250	1,160 €
Bornier	3	MC24366	0,238 €	100	0,714 €	1711725	1,450 €	100	4,350 €	1711725	1,410 €	100	4,23 €	1711725	1,132 €	75	3,396 €
Connecteurs kk	9	22-23-2031	0,085 €	150	0,765 €		0,131 €	100	1,179 €		0,130 €	100	1,17 €		0,115 €	10-240	1,035 €
Jumper	1	10-08-1021	0,161 €	150	0,161 €	22-28-4020	0,064 €	100	0,064 €	22284020	0,086 €	100	0,09 €		1,768 €	100	1,768 €
TOTAL					**11,727 €**				**15,225 €**				**13,77 €**				**15,622 €**
8X2A/ 2 ponts																	
Diodes schottky 2A	12	SB240-E3/54	0,287 €	100	3,444 €		0,190 €	100	2,280 €		0,210 €	100	2,520 €		0,253 €	20	3,036 €
Diodes Bat 85	4	BAT85S-TR	0,096 €	500	0,384 €		0,098 €	500	0,392 €		0,044 €	10000	0,18 €		0,045 €	2500	0,180 €
100n	6	F102K43Y5RP6UK5R	0,154 €	100	0,924 €		0,138 €	100	0,828 €	S102K43Y5PR63K7R	0,127 €	100	0,76 €		0,172 €	25	1,032 €
470n/35v	2	35ZLH470MEFC10X16	0,257 €	100	0,514 €	UBW1V471MHD	1,260 €	100	2,520 €		0,248 €	100	0,50 €		0,346 €	100	0,692 €
Driver moteur	2	L298N	3,060 €	100	6,120 €		2,920 €	100	5,840 €		2,852 €	100	5,70 €		2,490 €	10	4,980 €
Microcontrôleur	1	PIC18F26K22-I/SP	2,440 €	10	2,440 €		2,550 €	10	2,550 €		2,350 €	25	2,35 €		2,797 €	5	2,797 €
Connecteurs nappes	1		1,380 €	100	1,380 €		0,772 €	100	0,772 €		0,687 €	250	0,69 €		1,160 €	250	1,160 €
Bornier	5	MC24366	0,238 €	100	1,190 €	1711725	1,450 €	100	7,250 €	1711725	1,410 €	100	7,05 €	1711725	1,132 €	75	5,660 €
Connecteurs kk	9	22-23-2031	0,085 €	150	0,765 €		0,131 €	100	1,179 €		0,130 €	100	1,17 €		0,115 €	10-240	1,035 €
Jumper	1	10-08-1021	0,161 €	150	0,161 €	22-28-4020	0,064 €	100	0,064 €	22284020	0,086 €	100	0,09 €		1,768 €	100	1,768 €
TOTAL					**17,322 €**				**23,675 €**				**21,00 €**				**22,340 €**

Illustration 30: Étude de coût 8X2A

Index des illustrations