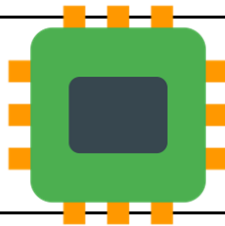


2018

API



IO

DOSSIER REVUE DE PROJET

Cartes entrées / sorties polyvalentes industrielles commandées par
Arduino & Raspberry.



FOURNIER Robin – VIAN Thomas – MACKER-CAVALIER Mathieu

SOMMAIRE



1. Présentation et situation du projet	3
1.1 Contexte de réalisation	3
1.2 Synoptique	4
2. Diagrammes UML & SYSML	5
2.1 Diagramme de cas d'utilisation	5
2.2 Diagramme de Blocs.....	6
2.3 Diagramme de Séquence.....	7
3. Répartition des fonctions	10
4. MACKER CAVALIER Mathieu IR1	15
4.1 Introduction	15
4.2 Diagramme de Gantt (prévisionnel) et plan des itérations.....	15
4.3 Diagramme de Blocs	17
4.4 Activités réalisées	17
4.4.1 Appropriation du cahier des charges et de la documentation initiale	17
4.4.2 Etude du convertisseur USB↔ISS.....	18
4.4.3 Mise en œuvre du module USB-ISS.....	19
4.4.4 Etude du composant MCP23017 + Préparation de la carte de prototypage	21
4.4.5 Communication avec le module MCP23017 a travers de l'I2C	23
4.4.6 Codage C++ de la supervision des E/S numériques.....	27
4.4.7 Gestion de version de logicielle.....	36
4.4.8 Etude du composant LTC2487	38
4.4.9 Présentation de solution de contournement pour le module LTC2487.....	40
4.4.10 Codage C++ de la supervision des Entrées analogiques.....	41
4.4.11 Travail restant.....	44
4.5 Annexe de travail.....	45
4.5.1 Journal de Bord.....	45
4.5.2 Schémas.....	46

VIAN Thomas EC2	49
Présentation du projet	49
Schéma analogique API IO final	50
Analyse des schémas	53
Schéma de structurel Analogique	56
Présentation du Level shifter	58
 6. FOURNIER Robin EC3	 61
6.1 Introduction	61
6.2 Base de travail	64
6.3 Extenseur 16 E/S I2C	65
6.4 Structure entrées digitales	67
6.5 Structure sortie commande relais	68
6.6 Level Shifter	70
6.7 Résumé	71
6.8 Essais et Vérifications des Adresses	72
6.9 Shield RPI	73
6.10 Schéma de principe Entier Carte DI/DO	75
6.11 Routage Entier Carte DI/DO	78
6.12 Routage Entier Carte DI/DO	79
6.13 Problèmes rencontrés	80
6.14 Partie Physique	81
6.15 Journal de Bord	86
Annexe Nomenclature	88

1. Présentation et situation du projet

1.1 Contexte de réalisation



Constitution de l'équipe de projet :	Étudiant 1 EC ☒ IR	Étudiant 2 ☒ EC IR	Étudiant 3 ☒ EC IR
Projet développé :	Au lycée ou en centre de formation ☐ En entreprise ☒ Mixte ☐		
Type de client ou donneur d'ordre (commanditaire) :	Entreprise ou organisme commanditaire : ☒ Oui ☐ Non Nom : API..... Adresse : 402 avenue des Lacs 84270 VEDENE..... Contact : M. BUNEL Jérôme..... Origine du projet : > Idée : Lycée ☒ Entreprise ☐ > Cahier des charges : Lycée ☒ Entreprise ☐ > Suivi du projet : Lycée ☒ Entreprise ☐		
Si le projet est développé en partenariat avec une entreprise :	Nom de l'entreprise : API..... Adresse de l'entreprise : 402 avenue des Lacs 84270 VEDENE..... Adresse site : https://sites.google.com/sasapi.fr/accueil Tél. : 0615806855..... Courriel : jbunel@sasapi.fr		

Étudiant 1 : MACKER-CAVALIER Mathieu IR

Étudiant 2 : VIAN Thomas EC

Étudiant 3 : FOURNIER Robin EC

La société API est une société proposant des services, elles intervient dans plusieurs domaines comme :

- l'industrie du verre
- l'agro-alimentaire
- L'industrie du papier
- Les carrières
- Les machines spéciales
- TPE et artisans
- ...

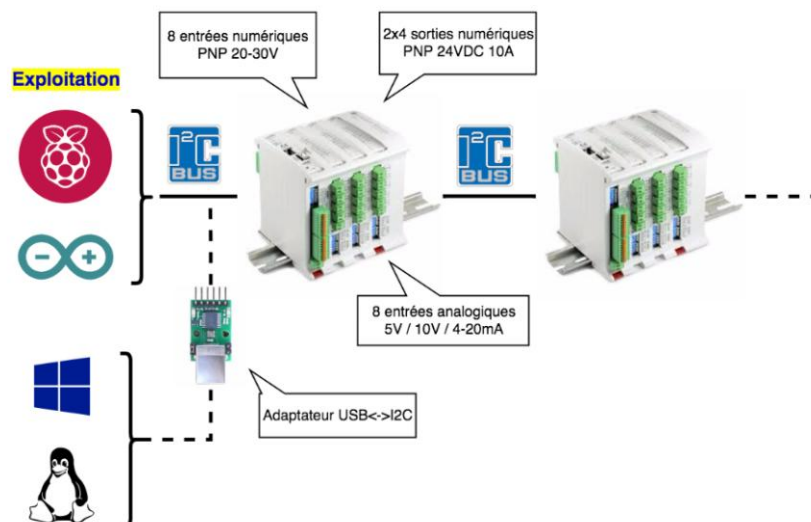
Elle possède également une activité axée sur la production et la gestion des énergies.

Dans ces domaines, API propose notamment les services suivants :

- Audit des réseaux d'énergie (air et électrique).
- Mesures, historisations, bilan des consommations.
- Préconisation d'économie d'énergie (variation de vitesse, régulation ...)
- Automatisation des procédés de gestion, Domotique, Immotique.
- Etude et calcul d'installation photovoltaïque
- ...

Le projet est né suite au besoin de l'entreprise API de posséder une carte communiquant en I2C, chainable, permettant de commander des sorties numériques et d'acquérir des signaux numériques et analogique. Ces cartes vont notamment remplacer des vieux automates plus coûteux. Ces cartes devront pouvoir être commandés à partir d'un ordinateur, une carte Arduino et enfin une carte RaspBerry Pi. Enfin une application graphique de test doit être fournie afin de pouvoir piloter les différentes E/S de la carte.

1.2 Synoptique



La configuration entre la carte et le système qui l'opère (Windows ou Linux) s'agit de la partie dont l'élève IR s'occupera.

La partie « opératrice » du système s'agit de la carte en elle-même, cette partie est réalisée par les deux élèves EC.

Contraintes financières (budget alloué) :

Budget estimé de 200 à 300€

L'entreprise API participera au financement du projet.

Contraintes de développement (matériel et/ou logiciel imposé / technologies utilisées) :

La spécification, conception et codage seront modélisés.

Contraintes qualité (conformité, délais, ...) :

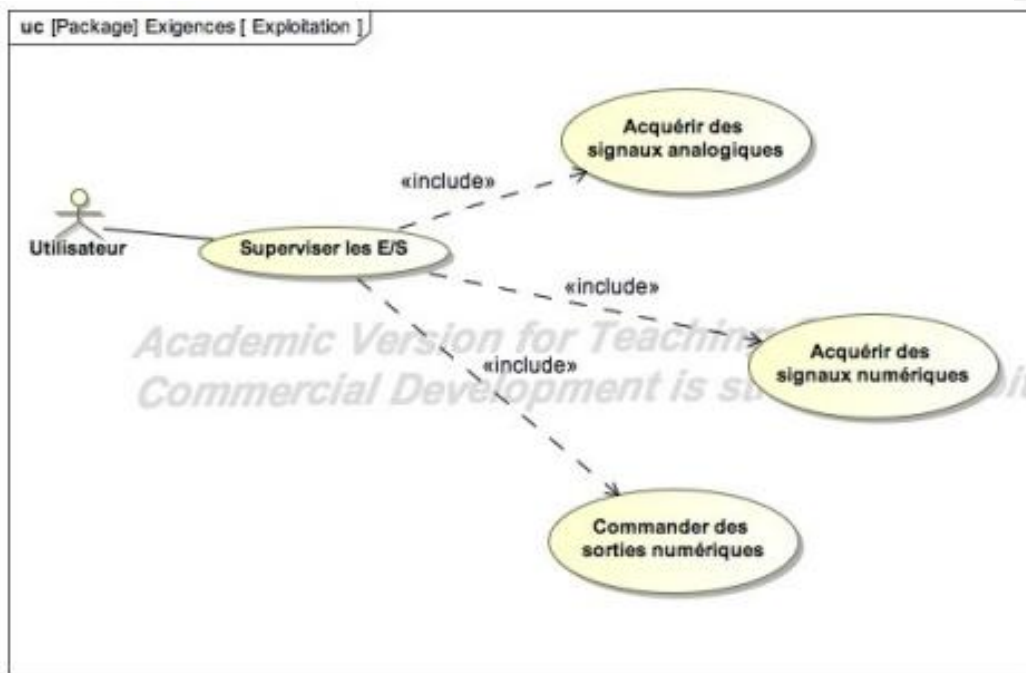
Maintenable, maniable (ergonomie)

Contraintes de fiabilité, sécurité :

Les accès logiciels seront sécurisés.

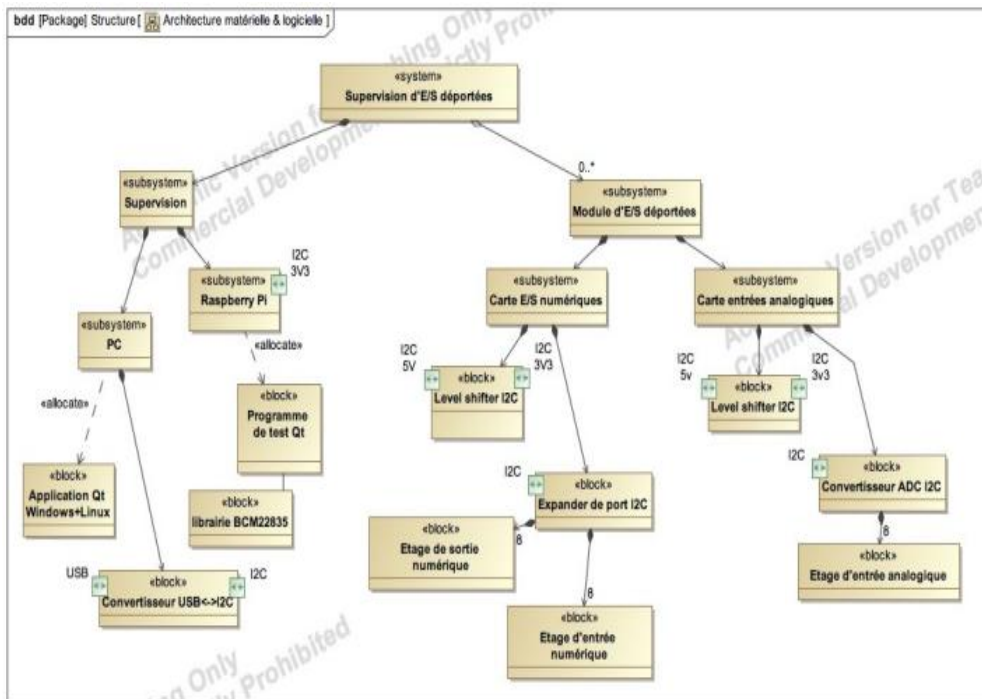
2. Diagrammes UML & SYSML

2.1 Diagramme de cas d'utilisation



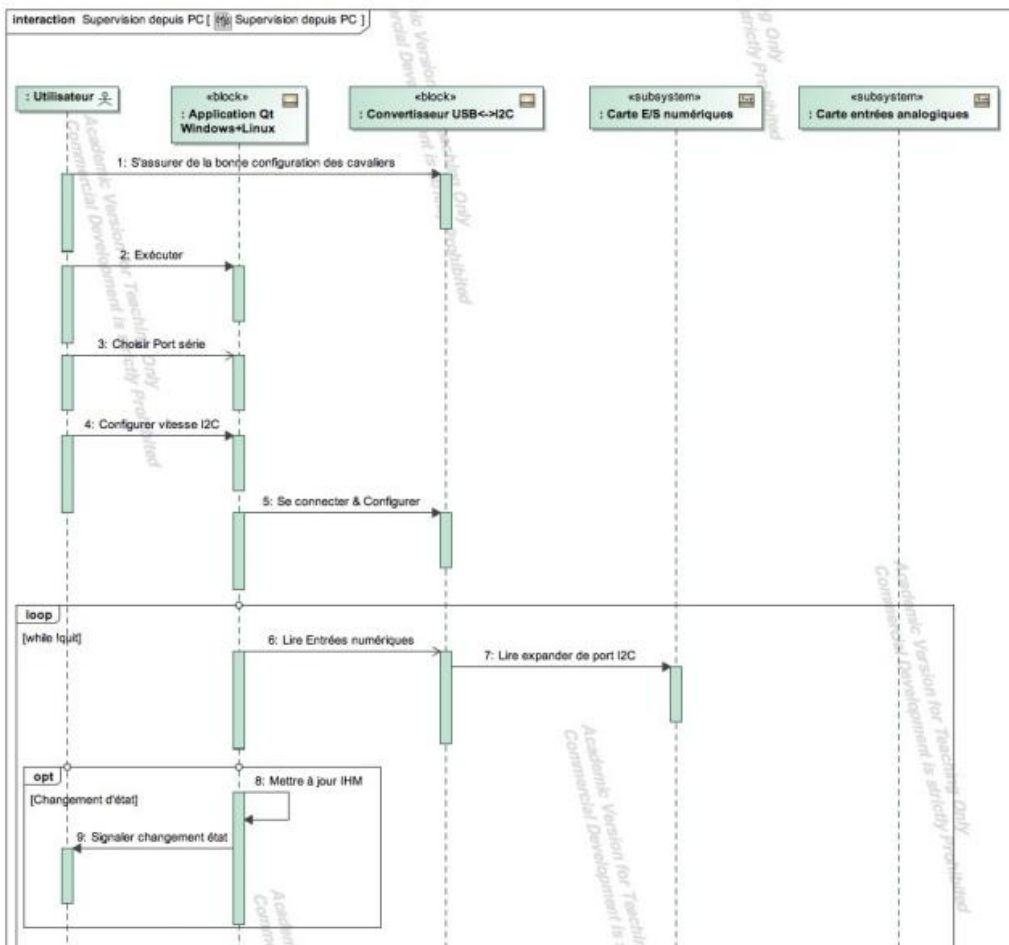
La carte API IO doit permettre contrôler les entrées et les sorties. Pour cela il faut pouvoir acquérir des signaux analogiques et des signaux numériques, il doit également pouvoir commander des sorties numériques.

2.2 Diagramme définition de block

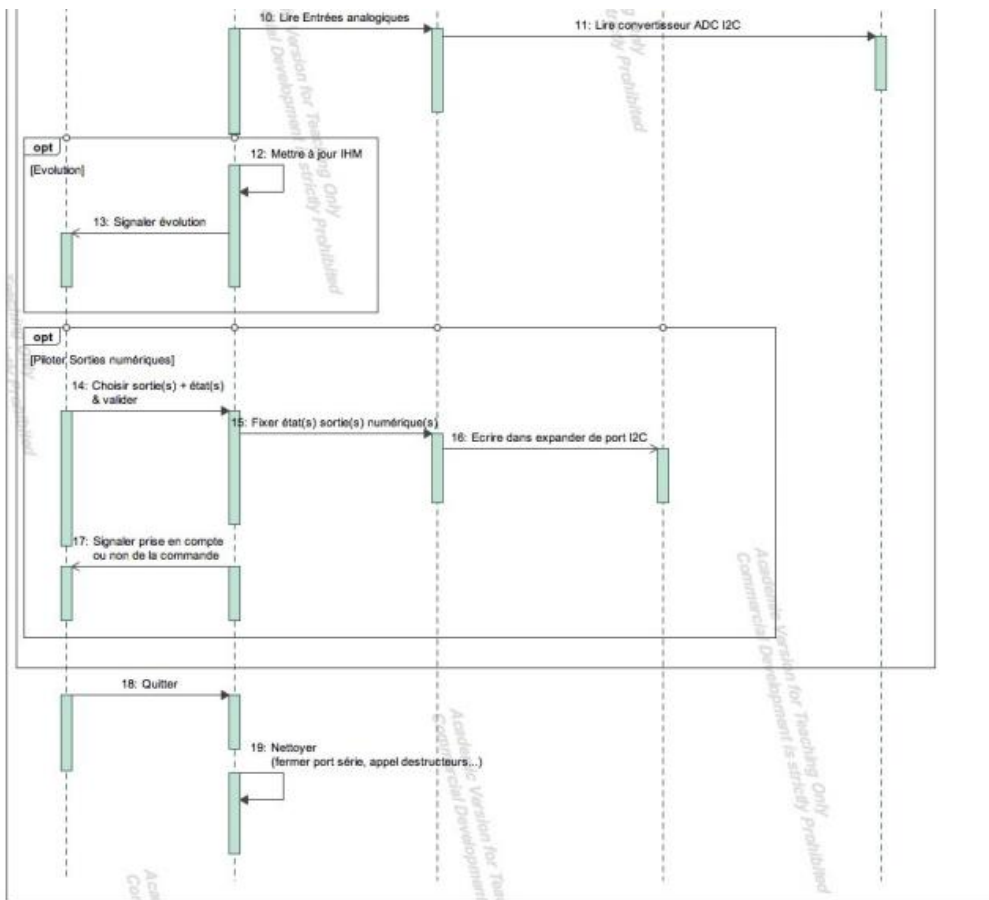


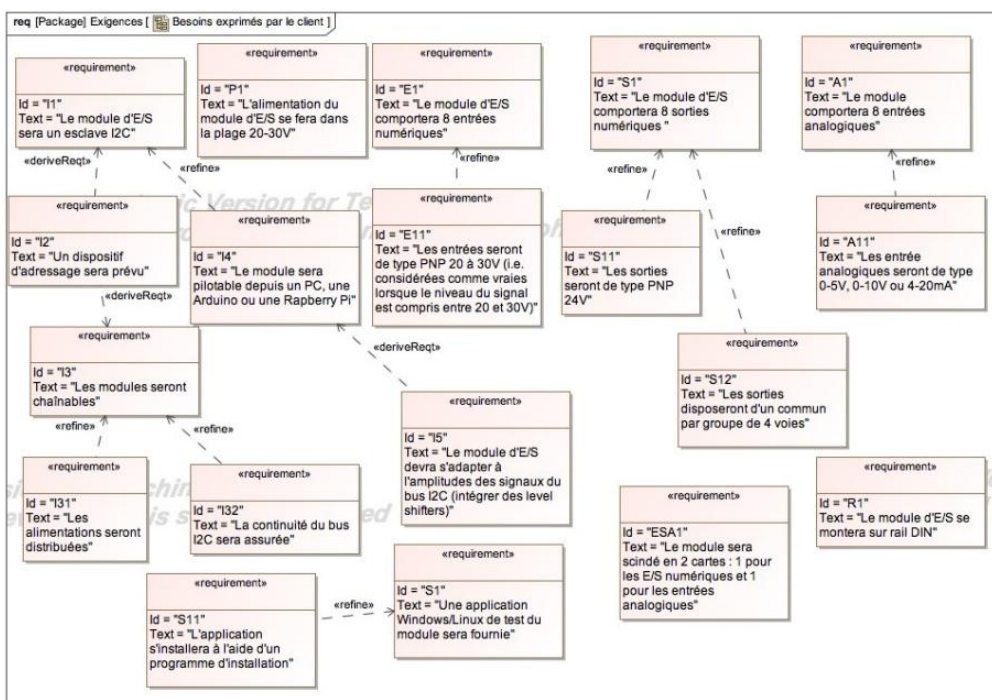
Donc pour ce système il y a 2 sous systèmes, une partie Supervision dont s'occupe l'étudiant IR et une partie module d'E/S déportées à la charge des étudiants EC. Lesquels sont répartis sur la carte E/S numériques et sur la carte d'entrée analogique.

2.3 Diagramme de séquence



2.3 Diagramme de séquence





3. Répartition des fonctions



Étudiant 1 IR	<i>Liste des tâches assurées par l'étudiant</i> • Mettre en œuvre l'adaptateur USB<->I2C • Etudier les datasheets des composants I2C • Câbler une plaquette d'essai disposant des composants I2C présents dans la carte IO I2C • Concevoir/Coder/Tester des classes C++ d'accès aux composants I2C (Expander, ADC) via l'adaptateur USB<->I2C • Concevoir/Coder/Tester une application Qt Windows+Linux de pilotage de cartes IO I2C • Créer un programme d'installation de l'application sur Windows et éventuellement un paquet type rpm pour Linux • Assurer la gestion de version logicielle (Git+Bitbucket) • Rédiger un manuel de démarrage rapide pour l'installation et l'utilisation de l'application	Installation : Framework Qt (x64) Mise en œuvre : Framework Qt/C++, Port expander et ADC sur bus I2C Configuration : Réalisation : Concevoir une application de pilotage de la carte IO I2C Documentation : Guide d'installation, manuel utilisateur, dossier de développement
-----------------------------	---	---

Étudiant 2 EC	<i>Liste des tâches assurées par l'étudiant</i> • Analyser le schéma proposé. • Valider le choix d'un convertisseur analogique/numérique. • Concevoir/Réaliser/Tester une carte s'insérant dans un rail DIN de largeur 72mm. • Développer une application permettant à partir d'une carte Raspberry Pi de visualiser la valeur des entrées analogiques. • Faire de même à partir d'une carte Arduino.	Installation : Sur Raspberry Pi : librairies C/C++ BCM2835 Sur Arduino : librairie wire Mise en œuvre : Valider par prototypage rapide le choix d'un convertisseur analogique numérique parmi ceux proposés. Proposer un autre modèle le cas échéant. Réalisation : Suite aux essais préalables, participer avec l'autre étudiant EC à la finalisation du schéma complet de la carte afin de respecter une cohérence de conception. Concevoir un circuit imprimé intégrant tous les éléments du cahier des charges de la carte d'entrées analogiques. Documentation : Bibliothèque C/C++ d'accès au convertisseur. Documents de fabrication de la carte. Ces documents devront permettre une fabrication industrielle du circuit imprimé.
-----------------------------	---	--

3. Répartition des fonctions



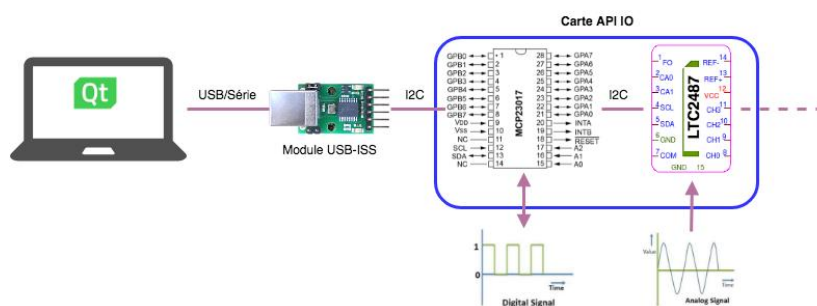
Étudiant 3 EC	<i>Liste des tâches assurées par l'étudiant</i> - Analyser le schéma proposé. - Valider le bon fonctionnement des structures relatives aux entrées/sorties logiques. - Concevoir/Réaliser/Tester une carte s'insérant dans un rail DIN de largeur 72mm. - Développer une application permettant à partir d'une carte Raspberry Pi de visualiser l'état des entrées logiques, et de piloter les sorties logiques. - Faire de même à partir d'une carte Arduino.	Installation : Sur Raspberry Pi : librairies C/C++ BCM2835 Sur Arduino : librairie wire Mise en œuvre : Valider par prototypage rapide la structure d'acquisition des niveaux logiques et celle de pilotage des sorties. Apporter des modifications si nécessaire. Réalisation : Suite aux essais préalables, participer avec l'autre étudiant EC à la finalisation du schéma complet de la carte afin de respecter une cohérence de conception. Concevoir un circuit imprimé tous les éléments du cahier des charges de la carte d'E/S numériques. Documentation : Bibliothèque C/C++ d'accès aux entrées/sorties logiques. Documents de fabrication de la carte. Ces documents devront permettre une fabrication industrielle du circuit imprimé.
Tous les étudiants	✓ <i>Domaines de physique à traiter par l'ensemble des étudiants de l'équipe projet :</i> - Traitement de signaux analogiques. - Numérisation de signaux analogiques.	

Partie personnelle étudiant n°1
Mathieu MACKER-CAVALIER
(Option IR)

4. MACKER CAVALIER Mathieu IR1

4.1 Introduction

La tâche qui m'a été confié consiste à superviser la carte API IO depuis une application Qt graphique (fonctionnant sur un ordinateur Windows voire Linux) qui s'interface avec les composants I2C qui gèrent les entrées/sorties à l'aide d'un module connecté par USB au PC et avec lequel on communique par liaison série type RS232.



L'application doit permettre la prise en charge de plusieurs cartes. Il faut donc penser lors du développement à permettre la communication avec plusieurs de ces modules de conversion Liaison série ↔ I2C.

4.2 Diagramme de Gantt (prévisionnel) et plan des itérations

Projet	201 h	Jeu 11/01/18	Jeu 24/05/18		
Tâches Communes	10 h	Jeu 11/01/18	Mer 17/01/18		
Lecture dossier	2 h	Jeu 11/01/18	Jeu 11/01/18		Mathieu Macker;Robin Fournier;Thomas Vian
Analyse de l'existant	1 h	Jeu 11/01/18	Jeu 11/01/18	3	Mathieu Macker;Robin Fournier;Thomas Vian
Réunion de synthèse	1 h	Jeu 11/01/18	Jeu 11/01/18	4	Mathieu Macker;Robin Fournier;Thomas Vian
Réaliser un plan des itérations	2 h	Jeu 11/01/18	Ven 12/01/18	5	Mathieu Macker;Robin Fournier;Thomas Vian
Réaliser un planning prévisionnel	4 h	Ven 12/01/18	Mer 17/01/18	6	
Tâches IR	191 h	Mer 17/01/18	Jeu 24/05/18		
Plan d'itération 1	61 h	Mer 17/01/18	Jeu 15/02/18		
Etudes des documentations	4 h	Mer 17/01/18	Jeu 18/01/18	7	Mathieu Macker
Etudier les datasheets des composants I2C	5 h	Jeu 18/01/18	Jeu 18/01/18	10	Mathieu Macker
Câbler une plaquette d'essai disposant des composants I2C présents dans la carte IO I2C	6 h	Ven 19/01/18	Mer 24/01/18	11	Mathieu Macker
Réaliser des différents diagrammes et travail sur le management du projet	20 h	Mer 24/01/18	Jeu 01/02/18	12	Mathieu Macker
Concevoir/Coder/Tester des classes C++ d'accès aux composants I2C (Expander, ADC)	26 h	Jeu 01/02/18	Jeu 15/02/18	13	Mathieu Macker
Plan d'itération 2	52 h	Jeu 15/02/18	Jeu 29/03/18		
Concevoir/Coder/Tester une application Qt Windows/Linux de pilotage de cartes IO I2C	52 h	Jeu 15/02/18	Jeu 29/03/18	14	Mathieu Macker
Plan d'itération 3	78 h	Jeu 29/03/18	Jeu 24/05/18		
Créer un programme d'installation de l'application sur Windows et éventuellement un paquet type rpm pour Linux	26 h	Jeu 29/03/18	Jeu 12/04/18	16	Mathieu Macker
Assurer la gestion de version logicielle (Git+Bitbucket)	26 h	Jeu 12/04/18	Ven 11/05/18	18	Mathieu Macker
Rédiger un manuel de démarrage rapide pour l'installation et l'utilisation de l'application	26 h	Ven 11/05/18	Jeu 24/05/18	19	Mathieu Macker

Itération 1

<i>Description dans l'itération</i>	<i>Pourquoi dans cet ordre</i>
Etudes des documentations	Il s'agit de la première action faite afin de prendre connaissance du sujet. Elle est primordiale
Conception d'une plaquette et mise en œuvre des composants I2C et de l'adaptateur USB↔I2C	Il s'agit ici de prendre en main le module de convertisseur utilisé par le système pour transmettre les commandes aux modules I2C
Concevoir/coder/tester des classes C++ permettant l'accès aux composantes I2C	Réalisation de classes permettant l'accès aux modules réaliser avant la conception du programme

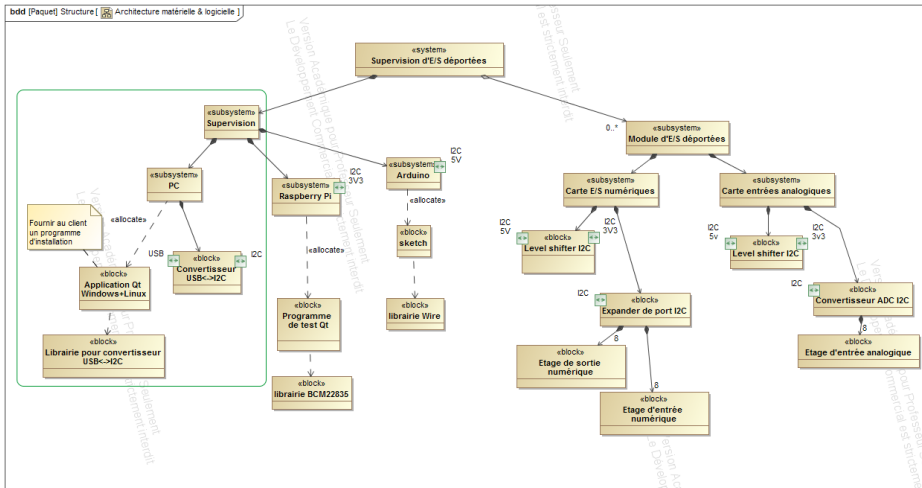
Itération 2

Concevoir/tester une application Qt Windows + Linux	Action pouvant être réalisée après la création des classes nécessaires auparavant
---	---

Itération 3

Créer un programme d'installation de l'application	Actions réalisées à la fin du projet de façon à permettre une mise en œuvre rapide par le client
Rédiger un manuel de démarrage rapide de l'installation et utilisation de l'application	

4.3 Diagramme de Blocs



Mis en forme : Normal

Dans le diagramme de blocs du système complet, tel que désiré par le client, la supervision des entrées/sorties se fera en phase d'exploitation depuis une carte Arduino voire une Raspberry Pi.

Cependant, la société API maîtrise ces plateformes et désire développer les programmes elle-même sachant que les besoins de leurs clients peuvent énormément varier.

Suite à un échange de mails lors de l'élaboration du cahier des charges, il a été néanmoins décidé que le lycée devait développer une application capable de visualiser/piloter les différentes entrées/sorties afin de pouvoir tester rapidement la/le(s) carte(s) électronique(s) conçue(s) par mes collègues d'EC.

La partie entourée en vert dans le diagramme de blocs du système complet correspond donc à la tâche qui m'a été attribué.

4.4 Activités réalisées

4.4.1 Appropriation du cahier des charges et de la documentation initiale

Afin de bien comprendre l'objectif du projet, il m'a été remis un cahier des charges.

Outre la présentation générale du projet (pour quelle raison il a été envisagé, le besoin auquel il répond, en quoi il consiste exactement...), le cahier des charge comprenait également divers diagrammes SysML/tableaux pour présenter le projet, identifier les différentes tâches et bien définir le rôle de chacun.

J'ai également eu des références vers de la documentation et notamment celle du module USB-ISS qui constitue le matériel avec lequel mon application s'interface directement. Cette documentation se présente en 2 parties :

1. Une première partie consacrée à la présentation générale du module et ces spécifications techniques. Elle commence par présenter le module, introduire les différents modes dans lequel il peut être utilisé, comment le module est reconnu par l'ordinateur et enfin introduit les commandes de configuration générale.

Voir https://www.robot-electronics.co.uk/htm/usb_iss_tech.htm

2. La seconde partie porte uniquement sur le fonctionnement en tant que convertisseur Série↔I2C. Elle introduit les différentes commandes secondaires que le module utilise, à quoi elles servent. La documentation décrit les octets de chaque commande et enfin donne des exemples.

Voir https://www.robot-electronics.co.uk/htm/usb_iss_i2c_tech.htm

4.4.2 Etude du convertisseur USB↔ISS :

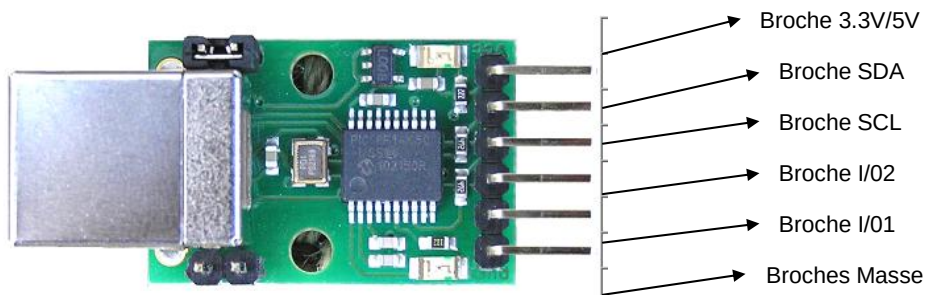
Le module de convertisseur USB-ISS est un module qui permet la communication entre un ordinateur et différents composants qui peuvent utiliser les modes de communications suivants :

Bus I2C
Bus Spi
Liaison Série TTL (0-3.3V/5V)

Le module est alimenté par le port USB et peut communiquer avec des composants fonctionnant soit en 3,3V et soit en 5V. La configuration se fait par un cavalier (en haut à gauche dans l'image ci-dessous) : Il faut le retirer pour un interfaçage en 3,3V et le mettre en place pour un interfaçage en 5V

Parmi les différents modes de communication proposés, c'est le mode I2C qui sera utilisé étant donné que c'est à travers ce bus que communique les composants choisis pour piloter les entrées/sorties à savoir le MCP23017 pour les 16 entrées/sorties numériques et les LTC2487 pour les 8 entrées analogiques.

Brochage du module USB-ISS :



Le module USB-ISS possède 1 commande de configuration générale qui prend la forme d'une trame débutant par l'octet 0x5A.

Cet octet est ensuite suivi par un octet qui définit une sous-commande :

- 0x01 pour connaître le mode actuel de fonctionnement du module (I2C, SPI...) , son identifiant (toujours 0x07) et sa version logicielle.
- 0x02 pour définir ce mode de fonctionnement
- 0x03 pour avoir le numéro de série du module l'USB codé sur 8 octets

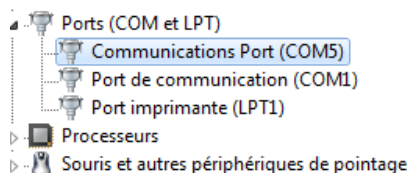
Chaque sous-commande est enfin suivie par un certain nombre d'octets qui définissent les valeurs de paramétrage propres à ces sous-commandes.

Dans le cadre du projet, il a été retenu un mode de fonctionnement I2C avec une fréquence d'horloge de 100kHz voire 400 kHz qui est la fréquence maximum supportée par les 2 composants utilisés.

Attention : Le module USB-ISS requiert l'installation d'un driver sous Windows. Il peut être récupéré via un lien présent sur la page de documentation du module. (https://www.robot-electronics.co.uk/html/usb_iss_tech.htm)

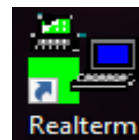
4.4.3 Mise en œuvre du module USB-ISS

Une fois connecté il faut s'assurer que le module apparaît bien dans la liste des périphériques :



Ici, le module utilise le Port (COM5), le port varie selon le système d'exploitation.

On peut ensuite utiliser un logiciel d'émulation de terminal série comme Realterm (Windows) pour communiquer avec le module car, bien que relié par un port USB, il est reconnu comme un port série grâce au driver installé.

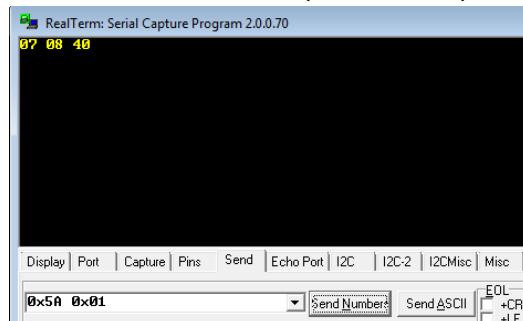


La documentation mentionne que n'importe quelle configuration du port série fonctionne étant donné que celui-ci est émulé. On peut donc choisir n'importe quelle vitesse (9600bauds, 115200...) et format (bits de donnée, parité, stop) pour la configuration.

Ci-dessous figure la trame à envoyer (0x5A 0x01) pour obtenir des informations sur le module (0x07 0x08 0x40) :

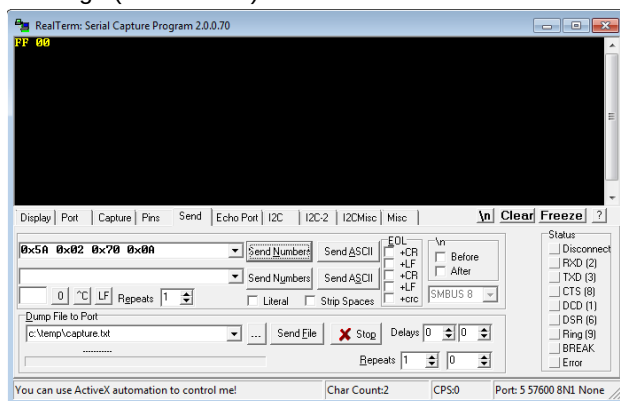
- 0x07 : identifiant du module (toujours 0x07)
- 0x08 : version du firmware

- 0x40 : mode de fonctionnement actuel (I2C à 100kHz).



Ci-dessous la commande de configuration utilisée pour le projet :

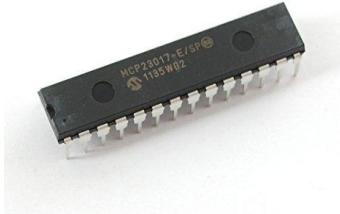
- 0x5A : Commande de configuration générale
- 0x02 : sous-commande de configuration du mode de fonctionnement
- 0x70 : spécification du mode de fonctionnement. Ici, mode I2C avec fréquence de 400kHz générée par hardware.
- 0x0A : configuration en entrée des broches IO1 et IO2, non utilisées dans le projet. Les configurer en entrée permet de limiter les conséquences en cas de mauvais câblage (court-circuit).



La réponse du module (0xFF 0x00) nous indique que la commande a bien été prise en compte. Celle-ci aurait été 0x00 0x00 dans le cas contraire.

4.4.4 Etude du composant MCP23017 + Préparation de la carte de prototypage

Etude du MCP23017



Le MCP23X17 est un composant qui fournit un total de 16 broches d'entrées/sorties qui peuvent être configurées chacune soit en entrée soit en sortie.

Selon sa version, le MCP23X17 est pilotable soit par bus I2C soit par bus SPI. La référence exacte du composant est alors :

MCP23017 pour la version bus I2C

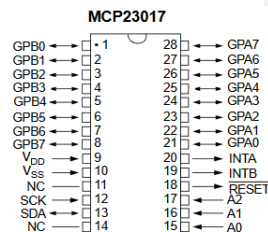
MCP23S17 pour la version bus SPI

Dans le cadre du projet, nous utilisons donc le MCP23017.

Registres

Le MCP23017 dispose de 8 registres internes de 8 bits.

Parmi ceux-ci, on peut citer les registres IODIRA et IODIRB qui permettent de configurer le sens de chaque broche d'E/S des 2 ports 8 bits du composant (GPAX et GPBX): 1 pour une entrée, 0 pour une sortie.



Un autre registre important est celui appelé IOCON qui sert à la configuration générale du composant. Le bit 7 (BANK) de ce registre contrôle à quelles adresses internes sont accessibles les différents registres.

Dans la configuration choisie (IOCON.BANK=1), les registres IODIRA et IODIRB prennent respectivement les adresses 0x00 et 0x10.

Les registres GPIOA et GPIOB, dans lesquelles figurent les états des broches, auront quant à eux les adresses 0x09 et 0x19.

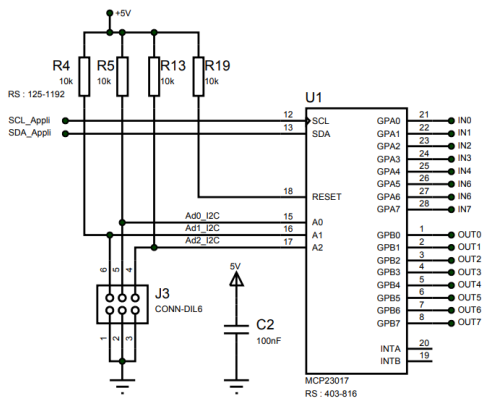
Adresse I2C

En mode standard, chaque composant I2C dispose d'une adresse 7 bits sur le bus. Celle-ci est souvent configurable par le biais de broches que l'on va relier soit à la masse soit au 5V et qui vont déterminer l'adresse finale. Dans le cas du MCP23017, ces broches sont celles identifiées A2, A1 et A0.

Schéma

Sur le schéma structurel qui m'a été fourni par mes camarades d'EC, on peut voir que :

les broches du port A sont en entrée et les broches du port B sont en sortie (labels)
l'adresse I2C est modifiable grâce à l'utilisation de cavaliers sur les broches A0, A1 et A2. Si elles sont toutes à la masse, l'adresse 7 bits du composant est 0x20.



Commande du module USB-ISS à utiliser pour accéder au MCP23017

Le module USB-ISS dispose de plusieurs commandes de communication I2C qui permettent de s'adapter à la majorité des composants I2C existants.

Après lecture de la datasheet, la communication avec le MCP23017 doit se faire avec la commande du convertisseur USB-ISS nommée : **I2C_AD1**

Elle permet de lire/écrire sur un composant disposant de registres adressables par 1 octet (« Read/Write 1 byte addressed devices (the majority of devices will use this one) »)

Dans notre projet, une trame d'écriture contiendra 5 octets :

0x55 pour la commande I2C_AD1

Adresse du composant + bit de lecture/écriture. Pour une adresse 7 bits I2C égale à 0x20, l'octet à indiquer sera 0x40 pour une écriture (et 0x41 pour une lecture)

Adresse du registre interne

Nombre d'octets que l'on souhaite écrire (toujours 1 dans notre programme)

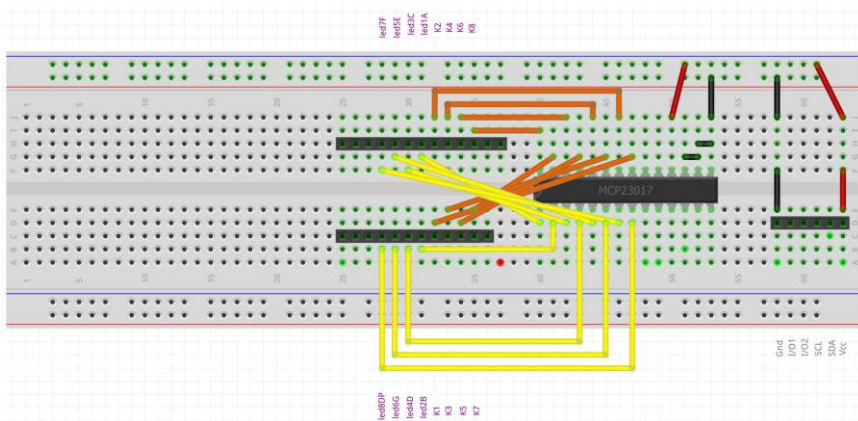
L'octet de donnée à écrire.

Pour la lecture la commande est la même sauf qu'il n'y a pas le dernier octet étant donné que l'on envoie aucune donnée. Le 4^{ème} octet indique juste le nombre d'octet à lire.

Câblage

Pour commencer il a fallu créer un schéma pour cette carte, afin de déterminer comment relier les différents constituants (module USB-ISS, MCP23017, latine de test disposants de

DELs et d'interrupteurs). Celui a été réalisé avec Fritzing.

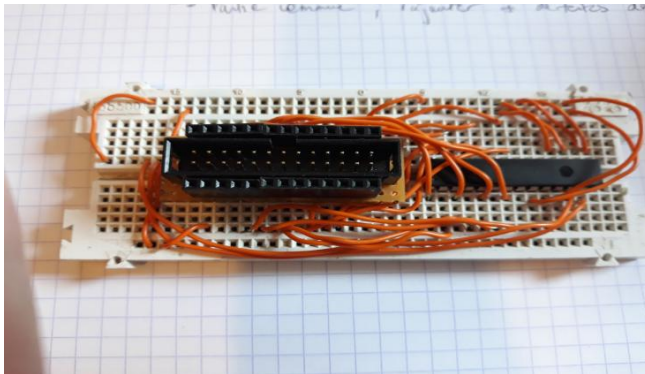


- Les câbles jaunes représentent les broches de sorties du MCP23017 vers les broches pilotant les DELs de la platine de test reliée à la carte de prototypage par un connecteur 28 broches.
- Les câbles orange représentent les broches d'entrées du MCP23017 vers les broches de la platine de test reliées aux interrupteurs.

Les câbles rouges sont reliés à l'alimentation 5V fournie par le module USB-ISS (connecteur 6 broches en bas à droite).

- Les câbles noirs sont reliés à la masse.

Ci-dessous la carte de prototypage câblée :



4.4.5 Communication avec le module MCP23017 a travers de l'I2C

Communication avec le MCP23017 a travers de l'I2C

Avant de piloter les broches d'E/S, il faut configurer le module USB-ISS et le MCP23017.

Ceci est réalisé avec Realterm en envoyant les trames suivantes :

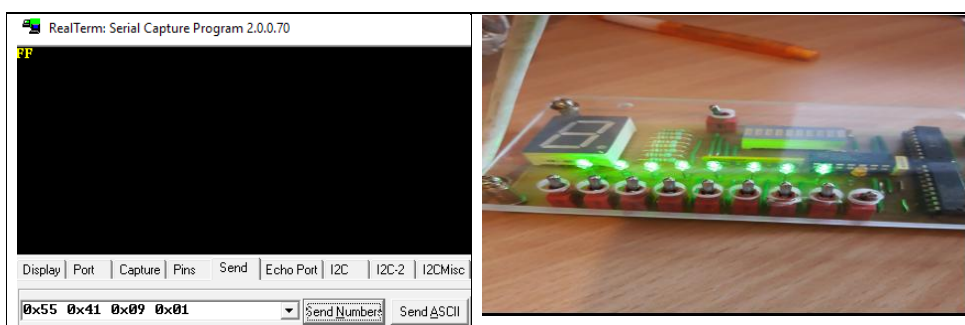
- 0x5A 0x02 0x70 0x0A // USB-ISS : mode I2C, 400kHz hardware
- 0x55 0x40 0x0A 0x01 0xA0 // MCP23017 : IOCON.BANK=1, IOCON.SEP=1 (IOCON@0x0A=0xA0)
- 0x55 0x40 0x01 0x00 0xFF // MCP23017 : Port A en entrée (IODIRA@0x00 = 0xFF)
- 0x55 0x40 0x01 0x10 0x00 // MCP23017 : Port B en sortie (IODIRB@0x10 = 0x00)

Après exécution de ces commandes, on peut alors commencer à piloter les entrées et les sorties.

Ci-dessous la commande saisie sur Realterm pour forcer toutes les sorties à 1 (GPIOB@0x19=0xFF) et son résultat sur la platine de test (tous les segments de l'afficheur sont allumés):



Ci-dessous la commande saisie sur Realterm pour lire toutes les entrées du Port A (GPIOA@0x09 on récupère FF) et son résultat sur la platine de test (tous les interrupteurs de la plaquette sont allumés):



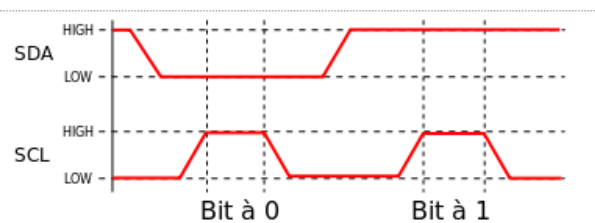
Le protocole I2C

L'I2C est un bus informatique inventé en 1982 par Philips, il permet de relier un microprocesseur et différents circuits.

I2C est un bus série synchrone bidirectionnel half-duplex fonctionnant en mode maître-esclave et sur lequel plusieurs équipements, maîtres et/ou esclaves, peuvent être connectés.

L'I2C utilise 2 fils (+ la masse)

SDA = (Serial Data Line) : ligne de données bidirectionnelle,
SCL = (Serial Clock Line) : ligne d'horloge.

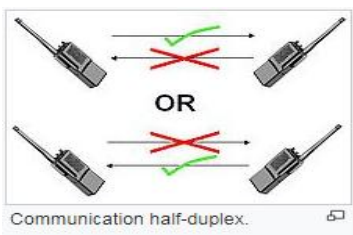


Chaque coup d'horloge = 1bit

SDA définit si le bit est à 1 ou 0.

Dans ce protocole l'émetteur envoie une trame de données et attend un acquittement (ACK=Acknowledge) de la part du récepteur.

En I2C, l'émetteur et le récepteur ne peuvent pas communiquer en même temps : c'est du Half-Duplex.



Initialement lors d'une communication I2C half-duplex, l'émetteur est le maître et le récepteur l'esclave.

Le message est composé de deux parties, la demande et la réponse.

La demande :

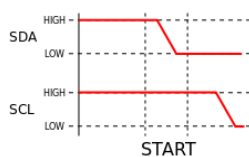
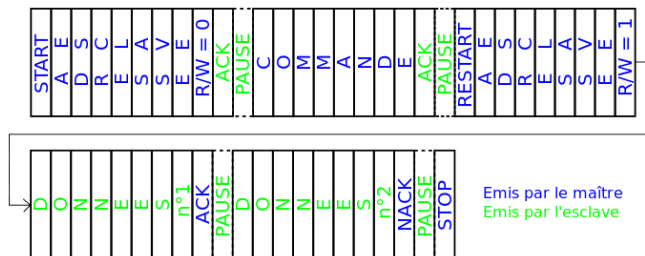
- Emission d'une condition de START par le maître (« S »),
- Emission de l'octet ou des octets d'adresse par le maître pour désigner un esclave,
- avec le bit R/W à 0 (voir la partie sur l'adressage ci-après),
- Réponse de l'esclave par un bit d'acquiescement ACK (ou de non-acquiescement NACK),
- Après chaque acquiescement, l'esclave peut demander une pause (« PA »).
- Emission d'un octet de commande par le maître pour l'esclave,
- Réponse de l'esclave par un bit d'acquiescement ACK (ou de non-acquiescement NACK),
- Emission d'une condition de RESTART par le maître (« RS »),

Emission de l'octet ou des octets d'adresse par le maître pour désigner le même esclave, avec le bit R/W à 1.

La réponse :

Le maître devient récepteur, l'esclave devient émetteur :

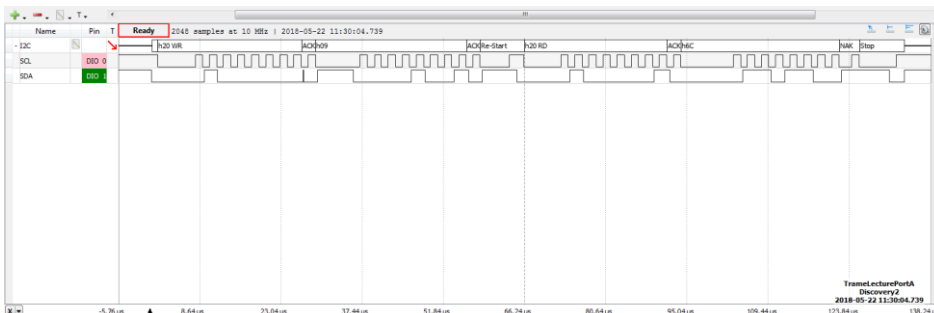
Émission d'un octet de données par l'esclave pour le maître,
Réponse du maître par un bit d'acquiescement ACK (ou de non-acquiescement NACK),
(Émission d'autres octets de données par l'esclave avec acquiescement du maître),
pour le dernier octet de données attendu par le maître, il répond par un NACK pour
mettre fin au dialogue,
Émission d'une condition de STOP par le maître (« P »).



Le bit de START est caractérisé par un front descendant de SDA pendant que SCL est toujours au niveau haut.

Trame relevée du projet :

Trame relevé du projet :



Il s'agit d'une séquence correspondant à une lecture sur le port A du MCP23017. On peut voir que la séquence commence lors d'un front descendant de SDA alors que SCL est à 1

(**condition de start**) et se termine sur un front montant de SDA lorsque SCL est à 1 (**condition de stop**).

La séquence est constituée de 2 échanges : une commande envoyée par le maître suivi de la réponse de l'esclave. On a donc 2 trames I2C successives séparées par une **condition de restart** qui permet au maître de garder le contrôle du bus entre la requête et la réponse. Il y a 9 impulsions d'horloge pour chaque octet transmis (la 9ème impulsion correspond à l'acquittement).

Le 1er octet transmis suite au start correspond à l'adresse 7 bits du composant (0x20) suivie du bit de lecture/écriture (traduit par h20WR dans l'interpréteur I2C donc en octet l'adresse 40 en écriture). On trouve ensuite l'adresse du registre à écrire.

Le 2nd octet transmis suite au restart correspond à l'adresse 7 bits du composant (0x20) suivie du bit de lecture/écriture (traduit par h20RD dans l'interpréteur I2C donc en octet l'adresse 41 en lecture). On trouve ensuite l'adresse du registre à lire.

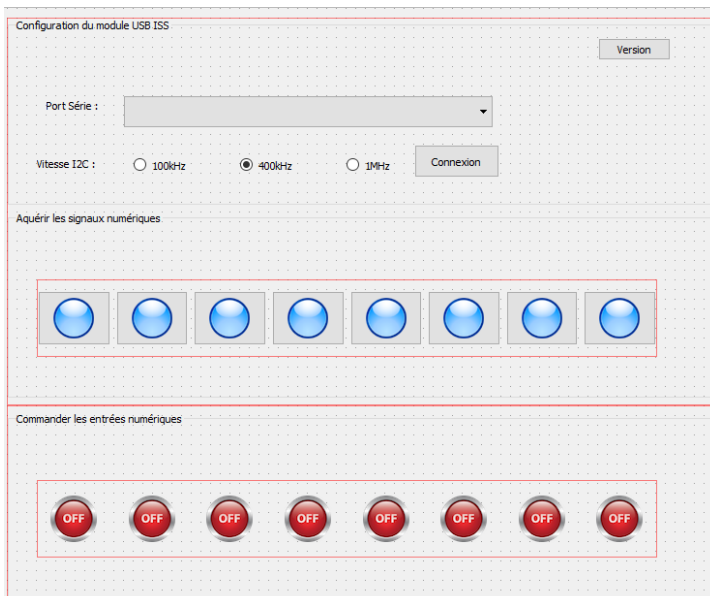
4.4.6 Codage C++ de la supervision des E/S numériques



J'ai utilisé l'environnement de développement (framework) Qt.

Il propose un certain nombre de fonctionnalités (interface graphique, communication série...) à travers un ensemble de classes C++.

Ci-dessous, l'interface graphique de mon application de supervision des E/S numériques :





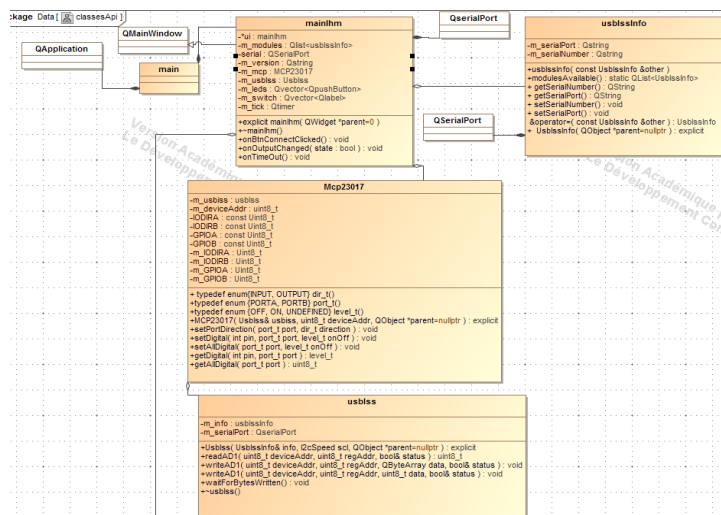
Cette IHM comporte 3 parties :

1. Connexion/Configuration du module USB-ISS
2. Commande des sorties numériques
3. Visualisation des entrées numériques

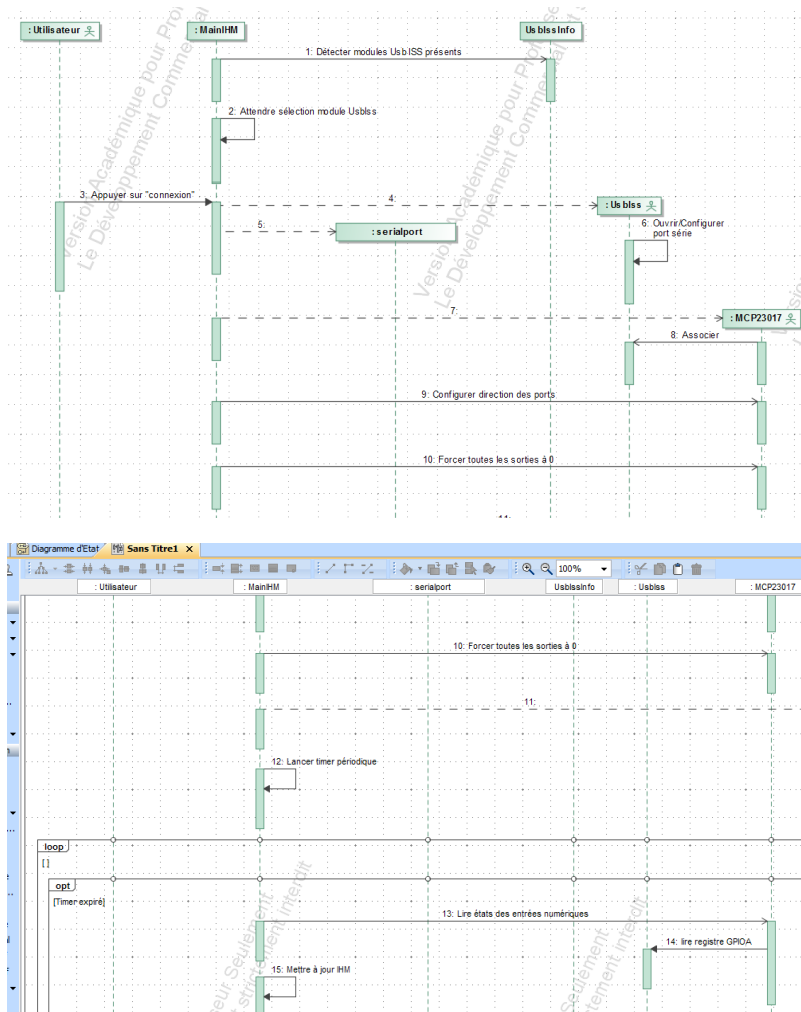
La commande des sorties numériques se fait directement par appui sur les boutons représentés sous forme de DELs.

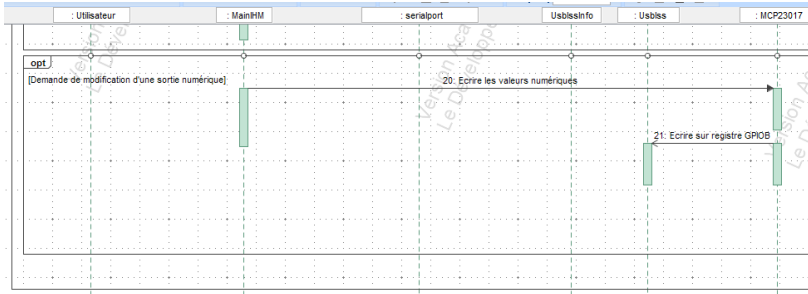
La visualisation des entrées se fait automatiquement grâce à un timer cyclique qui, lorsqu'il expire, va scruter l'état des entrées et rafraîchir l'IHM en conséquence.

Ci-dessous le diagramme des classes de mon application en rapport avec la gestion des E/S numériques :



Le fonctionnement général de l'application est expliqué ci-dessous sous forme d'un diagramme de séquence :





Classe UsbIssInfo :

Il s'agit d'une classe que j'ai créée pour ce projet. Elle permet de faire une détection de tous les modules USB-ISS branchés sur l'ordinateur à travers une méthode statique appelée

`availableModules()`.

```

QList<UsbIssInfo> UsbIssInfo::availableModules() {
    // On récupère la liste des ports séries détectés sur la machine (port
    série matériel ou virtuel)
    QList<QSerialPortInfo> infosSerialPorts =
    QSerialPortInfo::availablePorts();
    QList<UsbIssInfo> infosModulesUsbIss;
    QSerialPort * port = new QSerialPort();
    bool quit = false;

    // On parcourt la liste des ports série détectés
    for (const QSerialPortInfo &info : infosSerialPorts) {
        UsbIssInfo module;

        qDebug() << info.portName();
        qDebug() << info.description();
        qDebug() << info.productIdentifier();

        // Si le fabricant du port série actuel contient "Devantech Ltd."
        if( info.manufacturer() == "Devantech Ltd." ) {
            // On rentre en communication avec ce module pour récupérer son
            n° de version
            Devantech Ltd.
            // . Configuration du port série
            port->setPortName(info.portName());
            port->setBaudRate(QSerialPort::Baud9600);
            port->setDataBits(QSerialPort::Data8);
            port->setParity(QSerialPort::NoParity);
            port->setFlowControl(QSerialPort::NoFlowControl);
            port->setStopBits(QSerialPort::OneStop);

            // . Ouverture du port série
            port->open(QSerialPort::ReadWrite);

            // . Demande du numéro de version
            port->write("\x5A\x03");

            // . Attente & traitement de la réponse
            while(!quit) {
                port->waitForReadyRead(100);
            }
        }
    }
    return infosModulesUsbIss;
}
  
```

```

        if ( port->bytesAvailable() == 8) {
            quit = true;
        }
    }

    // On mémorise dans un objet UsbIssInfo d'une part le n° de
    // version du module et le nom du port série auquel il est connecté
    module.setSerialNumber (QString(port->readAll()));
    module.setSerialPort (info.portName());
    infosModulesUsbIss.append (module);

    // On ferme le port série
    port->close();
}

return infosModulesUsbIss;
}

```

Cette méthode est appelée dans le constructeur de la classe gérant l'IHM (MainIHM) de façon à remplir la liste déroulante appelée « Port série ».

La détection se fait par analyse de l'information de fabricant retournée pour chaque port série détecté par la méthode availablePorts() de la classe QSerialPortInfo.

Classe UsbIss :

Cette classe à plusieurs missions :

- Ouvrir le port série choisi par l'utilisateur dans la liste déroulante.
- Configurer le module de convertisseur Usb-ISS
- Proposer une interface avec les commandes du module USB-ISS

Constructeur

Le constructeur ouvre le port série auquel est connecté le module USB-ISS choisi et le configure. Il reçoit en paramètre plusieurs valeurs et notamment I2CSpeed sc1 qui contient la fréquence d'horloge de l'I2C choisie dans l'IHM. Le paramètre info contient les informations du port série associé au module USB-ISS

```

UsbIss::UsbIss(UsbIssInfo& info, I2cSpeed sc1, QObject *parent) :
    QObject(parent)
{

    m_serialPort = new QSerialPort();

    m_serialPort->setPortName(info.getSerialPort());
    m_serialPort->setBaudRate(QSerialPort::Baud9600);
    m_serialPort->setDataBits(QSerialPort::Data8);
    m_serialPort->setStopBits(QSerialPort::OneStop);

    bool ret =m_serialPort->open(QSerialPort::ReadWrite);
}

```

I2CSpeed est une énumération qui permet de limiter à un certain nombre de valeurs valides la fréquence I2C passée en paramètre :

```
Q_OBJECT
public:
    // énumération des différentes vitesses I2C
    enum I2cSpeed {
        sw_100kHz = 0x40,
        sw_400kHz = 0x50,
        hw_100kHz = 0x60,
        hw_400kHz = 0x70,
        hw_1MHz = 0x80,
    };
    Q_ENUM(I2cSpeed)
    //constructeur

    explicit UsbIss(UsbIssInfo& info, I2cSpeed scl, QObject *parent = nullptr);
```

Dans la suite du constructeur, on envoie la commande de configuration du module USB-ISS (expliquée lors de sa présentation générale) après avoir insérée dans celle-ci la valeur de la fréquence d'horloge I2C

```
if( ret ) {

    QByteArray trameConfig("\x5A\x02#\x0A");

    trameConfig[2] = scl;

    qDebug() << QString(trameConfig.toHex());

    m_serialPort->write(trameConfig);

    // Attendre la réception du n° de série

    bool quit = false;

    while(!quit) {

        m_serialPort->waitForReadyRead(100);

        if( m_serialPort->bytesAvailable() == 2) {

            quit = true;

        }

    }

    QByteArray response = m_serialPort->readAll();
```

Méthode WriteAD1()

La méthode writeAD1() constitue l'interface logicielle avec la commande I2D_AD1 du module USB-ISS utilisée en écriture. Elle ne permet l'écriture que d'un seul octet mais peut être surchargée pour en envoyer plusieurs.

```
void UsbIss::writeAD1(uint8_t deviceAddr, uint8_t regAddr, uint8_t data, bool
&status) {    //waitForBytesWritten

    QByteArray frame("\x55");
```

```

frame.append(deviceAddr);

frame.append(regAddr);

frame.append('\x01');

frame.append(data);

qDebug() << "writeAD1() : " + QString(frame.toHex());

status = m_serialPort->write(frame) != -1 ? true : false;

waitForBytesWritten();

}

```

Classe MCP23017 :

Cette classe MCP23017 a plusieurs missions :

- Configurer le MCP23017 (registre IOCON)
- Configurer la direction des ports d'E/S
- Proposer des méthodes pour lire/écrire sur les ports d'E/S numériques

Constructeur

Il reçoit en paramètre une référence vers un objet de la classe UsbIss. Celui-ci est mémorisé en interne (liste d'initialisation) de façon à pouvoir être utilisé par la suite pour s'adresser au module UsbIss par l'intermédiaire des méthodes readAD1() et writeAD1().

Le constructeur se contente d'envoyer la commande de configuration du MCP23017 (programmation du registre IOCON)

```

MCP23017::MCP23017(UsbIss& usbiss, uint8_t deviceAddress, QObject *parent) :
QObject(parent)

    , m_usbiss(usbiss), m_deviceAddr(deviceAddress)

{

    bool ok;

    m_usbiss.writeAD1(m_deviceAddr, 0x0A, 0xA0, ok); // 0xA0 <- IOCON.SEQOP=1,
IOCON.BANK=1

```

Méthode setPortDirection()

Cette méthode permet de configurer la direction des broches d'E/S des ports A ou B:

```

void MCP23017::setPortDirection( port_t port , dir_t direction){
    uint8_t reg = (port == PORTA) ? 0x00 : 0x10; //ici on test si le port est
PORTA ou PORTB
    uint8_t dir = (direction == INPUT) ? 0xFF : 0x00; //selon le port trouvé on
lui donne la direction entrée ou sortie

    bool ok;
    m_usbiss.writeAD1(m_deviceAddr, reg, dir, ok); // on appelle la méthode pour
écrire

```

```

    if( ok ) {
        uint8_t * iodir = (port == PORTA) ? &m_IODIRA : &m_IODIRB;
        *iodir = (direction == INPUT) ? 0xFF : 0x00;
    }
}

```

Méthode `getAllDigital()` :

Cette méthode permet de lire l'état de toutes les broches d'un port d'E/S

```

uint8_t MCP23017::getAllDigital(port_t port){

    uint8_t * gpio = (port == PORTA) ? &m_GPIOA : &m_GPIOB; // on crée un objet
    gpio dans lequel on écrit la valeur du Port

    uint8_t reg = (port == PORTA) ? GPIOA : GPIOB; // on crée un objet reg pour
    l'octet de registre il aura soit la valeur de GPIOA ou GPIOB

    bool ok;

    *gpio = m_usbiss.readAD1(m_deviceAddr, reg, ok); // appel de la méthode de
    lecture le résultat est alors *gpio

    return *gpio; // la méthode retourne la valeur pointée par l'objet *gpio
}

```

Méthode `setDigital()` :

Cette méthode permet de fixer l'état d'une broche de sortie d'un port d'E/S après avoir vérifiée qu'elle est effectivement configurée en sortie.

```

void MCP23017::setDigital(int pin, port_t port , level_t onOff){

    uint8_t * iodir = (port == PORTA) ? &m_IODIRA : &m_IODIRB;

    uint8_t * gpio = (port == PORTA) ? &m_GPIOA : &m_GPIOB;

    uint8_t reg = (port == PORTA) ? GPIOA : GPIOB;

    if( (*iodir & (1 << pin)) == 0x00) {

        *gpio = (onOff == ON) ? *gpio & ~(1 << pin) : *gpio | (1 << pin);

        bool ok;

        m_usbiss.writeAD1(m_deviceAddr, reg, *gpio, ok);

    } else {

        qDebug() << "setDigital() : la broche indiquée n'est pas configurée en
        sortie";

    }

}

```

Classe `MainIHM` :

Cette classe gère l'IHM. Elle dispose de méthodes qui agissent sur le matériel lorsque certains événements se produisent :

Appui sur le bouton « connecter » de l'IHM

↳ ouverture port série, configuration du module USB-ISS

Appui sur un bouton de pilotage de sortie numérique

↳ envoi d'une commande d'écriture dans le port de sortie du MCP23017

Expiration du timer cyclique

↳ lecture du port d'entrée du MCP23017 et mise à jour de la partie l'IHM qui représente l'état des entrées.

Dans la terminologie de Qt, les événements sont appelés des **signaux** et les méthodes qui les traitent sont appelées des **slots**.

```
/**
 * Slot appelé sur appui du bouton Connecter/Déconnecter de l'IHM
 */

void mainIhm::onBtnConnectClicked()
{
    if (this->ui->cbAvailableModules->isEnabled()) {
        this->ui->btnConnect->setText("Déconnecter");
        this->ui->cbAvailableModules->setEnabled(false);
        this->ui->rbtn100khz->setEnabled(false);
        this->ui->rbtn400khz->setEnabled(false);
        this->ui->rbtn1mhz->setEnabled(false);
        this->ui->grpboxOutput->setEnabled(true);
        this->ui->grpboxInput->setEnabled(true);

        UsbIss::I2cSpeed FreqI2c;

        FreqI2c = ui->rbtn100khz->isChecked()? UsbIss::hw_100kHz :
        UsbIss::hw_400kHz;
        FreqI2c = ui->rbtn1mhz->isChecked()? UsbIss::hw_1MHz : FreqI2c ;
        m_usbIss = new UsbIss (m_modules[ ui->cbAvailableModules-
        >currentIndex() ], FreqI2c);

        m_mcp = new MCP23017(*m_usbIss, 0x40); //??
        m_mcp->setPortDirection(MCP23017::PORTA, MCP23017::INPUT);
        m_mcp->setPortDirection(MCP23017::PORTB, MCP23017::OUTPUT);
        m_mcp->setAllDigital(MCP23017::PORTB, MCP23017::ON);
        for(int i = 0; i < m_leds.length(); i++) {
            m_leds.at(i)->setChecked(false);
        }
        m_ltc = new LTC2487(*m_usbIss, LTC2487_I2C_ADDRESS);
        m_tick.setInterval(250);
        m_tick.start();
    }
    else{
        this->ui->btnConnect->setText("Connecter");

        this->ui->cbAvailableModules->setEnabled(true);
    }
}
```

```

        this->ui->rbtn100khz->setEnabled(true);
        this->ui->rbtn400khz->setEnabled(true);
        this->ui->rbtn1mhz->setEnabled(true);
        this->ui->grpboxOutput->setEnabled(false);
        this->ui->grpboxInput->setEnabled(false);

        m_tick.stop();
        delete m_mcp;
        delete m_ltc;
        delete m_usbIss;
    }
}
/**

*   Slot appelé sur appui sur une DEL de l'IHM

*/

void mainIhm::onOutputChanged(bool state) {
    QObject * signalSrc = QObject::sender();
    QIcon icon;

    for(int i = 0; i < m_leds.length(); i++) {
        if(signalSrc == m_leds[ i ]) {
            m_mcp->setDigital(i, MCP23017::PORTB, state == false ?
MCP23017::ON : MCP23017::OFF);
        }
    }
}
/**

*   Slot appelé sur expiration du timer cyclique

*/

void mainIhm::onTimeOut() {
    uint8_t portVal;

    portVal = m_mcp->getAllDigital(MCP23017::PORTA);

    for(int i = 0; i < m_switchs.length(); i++) {
        if( (portVal & (1 << i)) != 0) {
            m_switchs.at(i)-
>setPixmap(QPixmap(QString::fromUtf8(":/img/on.png")));
        } else {
            m_switchs.at(i)-
>setPixmap(QPixmap(QString::fromUtf8(":/img/off.png")));
        }
    }

    double ai = m_ltc->getAnalog(0);
    ui->->display(ai);
    ui->pbAI0->setValue(static_cast<int>(ai * 1000));
}

```

4.4.7 Gestion de version de logicielle

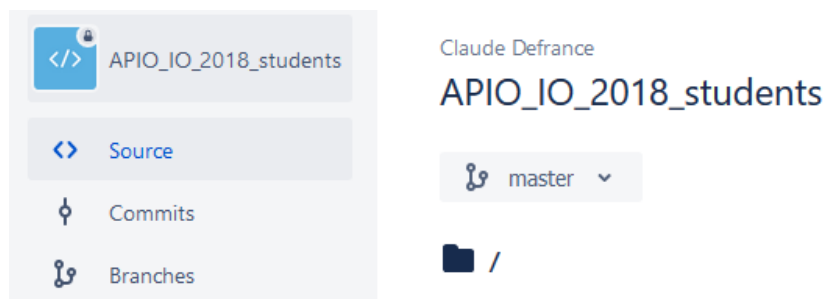
La gestion de version de logiciel permet de sauvegarder son travail, de pouvoir suivre le développement à chaque étape, de plus si le projet est effectué en groupe chaque personne

a accès à ce dépôt plusieurs personnes peuvent effectuer des modifications sur le même code en même temps il n'y a pas de conflit.

Pour effectuer cette gestion de version il a fallu plusieurs choses :

- Création d'un compte Attlasian
- Un espace sur BitBucket ou placer le code
- Installer sourceTree
- Définir un emplacement de fichier ou ranger le code pour que sourceTree utilise cet emplacement
- Commit le code

Espace bitbucket

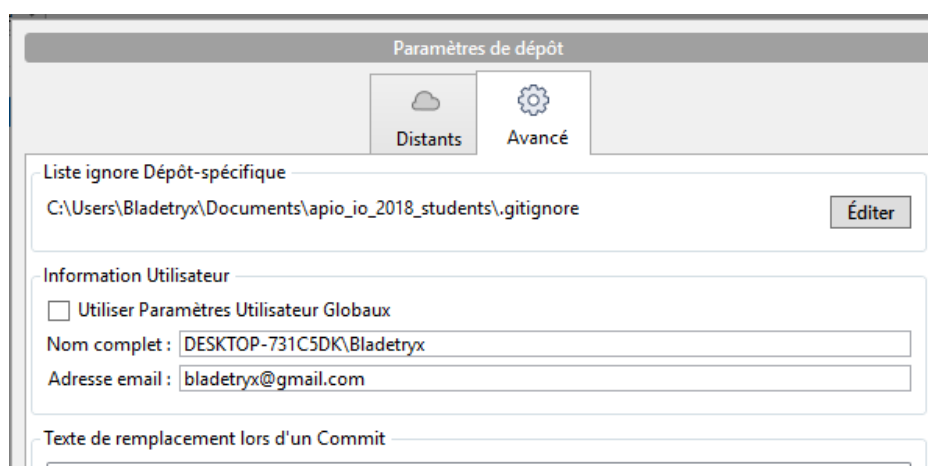


Pour sourceTree :

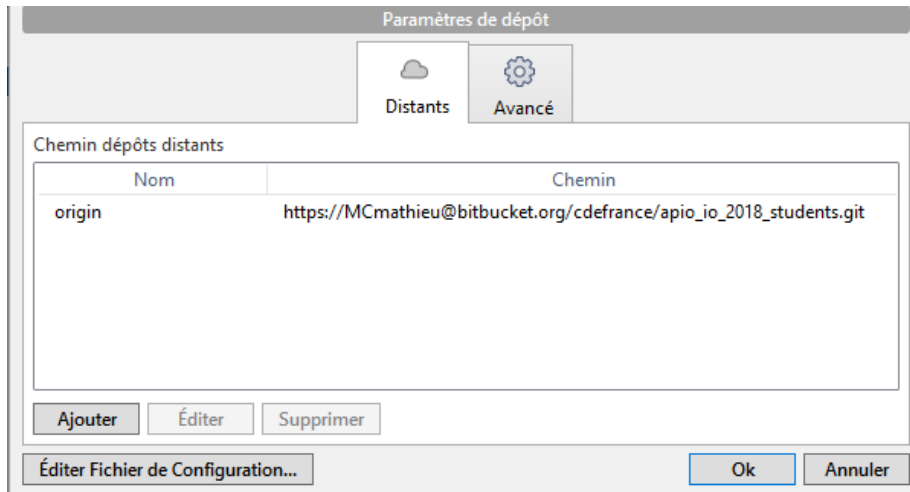
Lors de son installation il faut utiliser un compte Attlasian crée au préalable.

Une fois installé il faut le configurer :

1 Définir un répertoire référence pour sourceTree ou l'on va placer les différents codes :

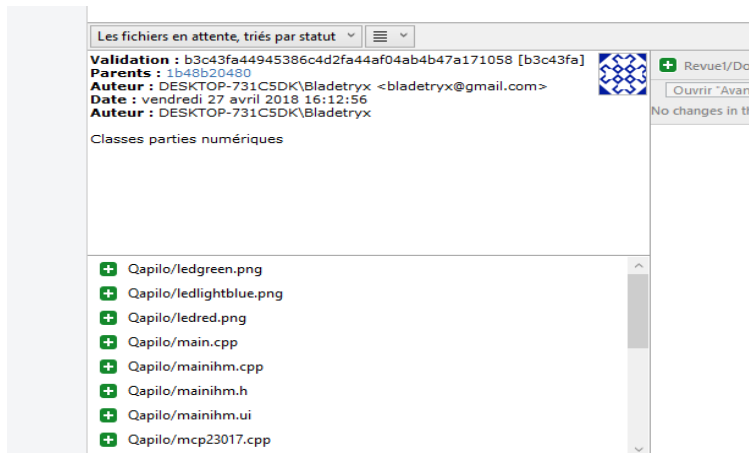


2 Il faut cibler le dépôt BitBucket sur lequel on va travailler



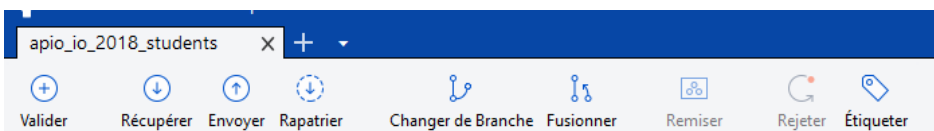
Il faut copier-coller le chemin du dépôt s'affichant lors de la connexion au dépôt.

On peut alors commencer déposer du code et l'envoyer au dépôt bit bucket



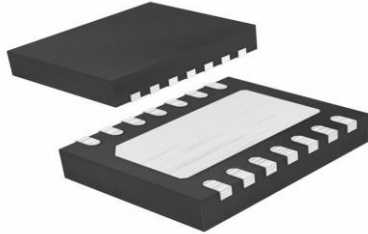
La première étape est le « commit » lors de l'ajout de nouveau fichier dans le répertoire sélectionner auparavant sourcetree propose dans l'onglet de commit ces nouveaux fichiers.

Cela les rajoute localement dans source tree pour les déposer dans le dépôt BitBucket il faut faire une action supplémentaire, il faut l'envoyer.



4.4.8 Etude du composant LTC2487

Etude du LTC2487



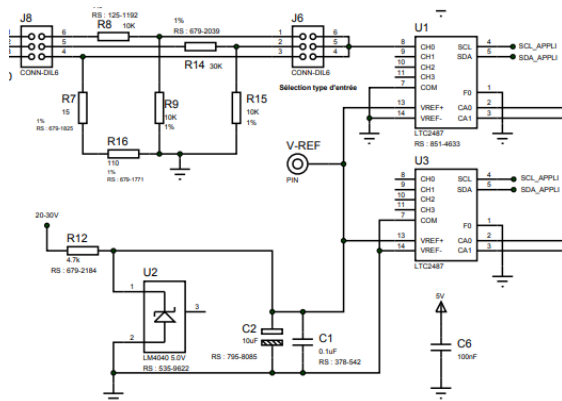
Le LTC2487 est un module composé de 4 voies en entrée analogue.

Le module communique en I2C, il est donc éligible dans le projet et peut être utilisé par le convertisseur i2c.

Le LTC2487 peut notamment être configuré afin de mesurer un signal externe avec l'aide de la combinaison des 4 voies en entrée analogue.

Dans le projet il n'est pas utilisé en différentiel mais en asymétrique c'est-à-dire que les valeurs des voies sont comparées avec une valeur référence au lieu de l'alimentation.

Schéma



Les tensions arrivent aux bornes 8,9,10,11 du composant qui représentent les différentes voies du composant, qui ensuite compare les valeurs reçues avec une valeur référentielle V-REF de 5v et relié à la masse. Il convertit alors la valeur en 16 bit (maximum 65535) encapsulé dans une trame de 24 bit qui est envoyé par SDA et SCL.

Sélections des voies

L'écriture pour sélectionner et lire ces voies se fait en 2 octets.

MUX ADDRESS					CHANNEL SELECTION				
SGL	ODD/ SIGN	A2	A1	A0	0	1	2	3	COM
*0	0	0	0	0	IN ⁺	IN ⁻			
0	0	0	0	1			IN ⁺	IN ⁻	
0	1	0	0	0	IN ⁻	IN ⁺			
0	1	0	0	1			IN ⁻	IN ⁺	
1	0	0	0	0	IN ⁺				IN ⁻
1	0	0	0	1			IN ⁺		IN ⁻
1	1	0	0	0		IN ⁺			IN ⁻
1	1	0	0	1				IN ⁺	IN ⁻

Le premier octet (SGL, ODD, A2, A1, A0) permet de sélectionner les voies tandis que le second octet (IM, FA, FB, SPD, GS2, GS1, GS0) permet de choisir la rejection de fréquence, le mode de vitesse, les mesures de température et le gain.

Devant le bit SGL il y a 3bits imposés : 101.

Donc pour la voie 0 l'octet sera :1011 0000. En hexa : 0xB0

Pour la voie 1 : 1011 0001

En hexa : 0xB1

Pour la voie 2 : 1011 1000 en hexa 0xB8

Pour la voie 3 : 1011 1001 en hexa 0xB9

Converter Configuration																CONVERTER CONFIGURATION			
1	0	EN	SGL	ODD	A2	A1	A0	EN2	IM	FA	FB	SPD	GS2	GS1	GS0				
1	0	0						X	X	X	X	X	X	X	X	Keep Previous			
1	0	1						0	X	X	X	X	X	X	X	Keep Previous			
1	0	1						1	0			0	0	0	0	External Input, Gain = 1, Autocalibration			
1	0	1						1	0			0	0	0	1	External Input, Gain = 4, Autocalibration			
1	0	1						1	0			0	0	1	0	External Input, Gain = 8, Autocalibration			
1	0	1						1	0			0	0	1	1	External Input, Gain = 16, Autocalibration			
1	0	1						1	0			0	1	0	0	External Input, Gain = 32, Autocalibration			
1	0	1						1	0			0	1	0	1	External Input, Gain = 64, Autocalibration			
1	0	1						1	0			0	1	1	0	External Input, Gain = 128, Autocalibration			
1	0	1						1	0			0	1	1	1	External Input, Gain = 256, Autocalibration			
1	0	1						1	0			1	0	0	0	External Input, Gain = 1, 2x Speed			
1	0	1						1	0			1	0	0	1	External Input, Gain = 2, 2x Speed			
1	0	1						1	0			1	0	1	0	External Input, Gain = 4, 2x Speed			
1	0	1						1	0			1	0	1	1	External Input, Gain = 8, 2x Speed			
1	0	1						1	0			1	1	0	0	External Input, Gain = 16, 2x Speed			
1	0	1						1	0			1	1	0	1	External Input, Gain = 32, 2x Speed			
1	0	1						1	0			1	1	1	0	External Input, Gain = 64, 2x Speed			
1	0	1						1	0			1	1	1	1	External Input, Gain = 128, 2x Speed			
1	0	1						1	0			0	0			External Input, Simultaneous 50Hz/60Hz Rejection			
1	0	1						1	0	0	1					External Input, 50Hz Rejection			
1	0	1						1	0	1	0					External Input, 60Hz Rejection			
1	0	1						1	0	1	1					Reserved, Do Not Use			
1	0	1						1	1	0	0	X	X	X	X	Temperature Input, Simultaneous 50Hz/60Hz Rejection			
1	0	1						1	1	0	1	X	X	X	X	Temperature Input, 50Hz Rejection			
1	0	1						1	1	1	0	X	X	X	X	Temperature Input, 60Hz Rejection			
1	0	1						1	1	1	1	X	X	X	X	Reserved, Do Not Use			

Ici les 2 octet : 1 0 1 SGL ODD A2 A1 A0 EN2 IM FA FB GS2 GS1 GS0, donc il y a les 3 premiers bits à valeurs imposés, les 5 suivants sont ceux des voies en les 8 derniers bits sont à 10XX1000 les X sont à 0 donc l'octet vaut 0x88.

4.4.9 Présentation de solution de contournement pour le module LTC2487 :



Suite à un problème sur le composant d'entrée analogique, le seul module LTC2487 ne fonctionne plus.

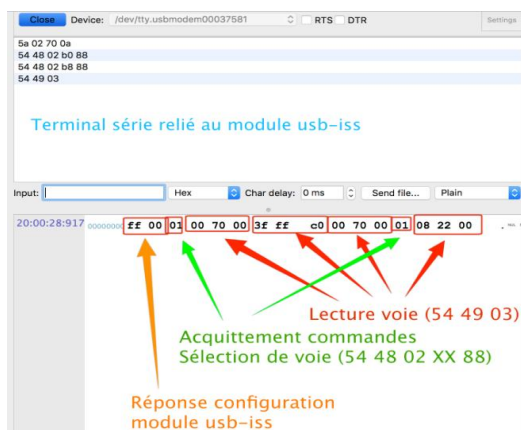
Afin de passer outre ce problème il a été décidé de passer par une carte arduino afin de simuler les voies analogiques.

Comment le programme marche :

Les 4 voies du composant LTC2487 sont donc matérialisées par les broches A0, A1, A2 et A3 sur lesquels on pourra brancher un potentiomètre pour faire varier les valeurs en 0v et 5v.

L'arduino est programmée pour avoir l'adresse I2C 0x24 sur 7 bits, travaillant en 8 bits on décale de 1 vers gauche pour avoir 0x48 il s'agit de l'adresse pour écrire et 0x49 pour lire.

Pour définir les voies à lire il faut au préalable avoir configuré le module usb<->iss de la même façon que la mcp23017. On peut alors choisir la voie à lire avec la commande I2C 0x54.



La commande 0x54 permet de lire/écrire des bits sur des modules qui n'ont pas d'adresse internes ce qui est le cas pour la LTC2487.

// Première commande : configuration de la vitesse I2C

//Seconde commande on écrit pour définir la voie à lire comme étant B0 (voie 0)

//troisième commande on écrit pour définir la voie à lire comme étant B8 (voie 3)

//quatrième commande on lit la voie précédemment choisit

Le programme a l'image du précédent utilise 3 classe, mainIhm pour l'interface utilisateur, la classe usbIss pour configurer la communication I2C et une classe Ltc2487 pour représenter le module afin de définir les voies à lire.

Il doit pouvoir :

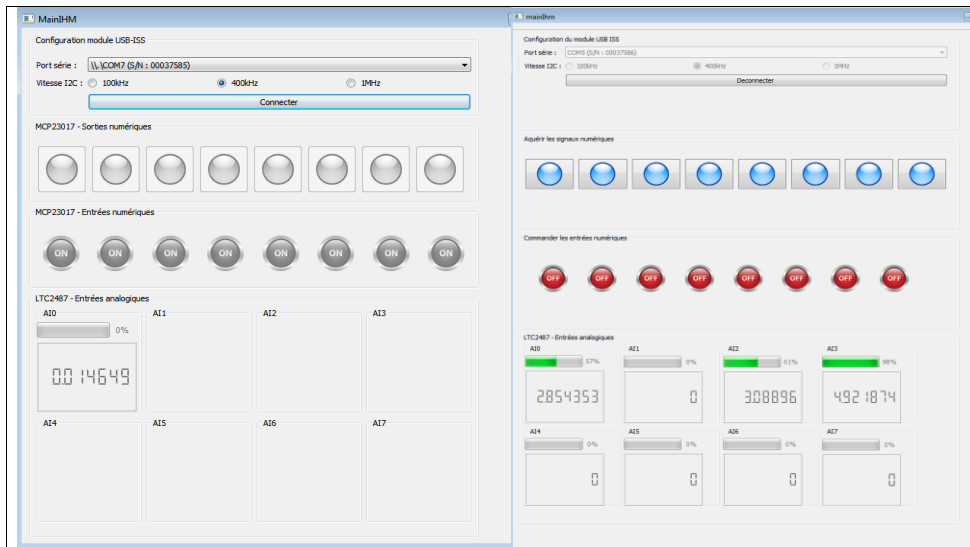
Configurer l'i2c mode

Choisir les voies à lire (ou toutes les lire)

Lire la voie sélectionnée

4.4.10 Codage C++ de la supervision des Entrées analogiques

Ci-dessous, l'interface graphique de mon application de supervision des entrées analogiques :

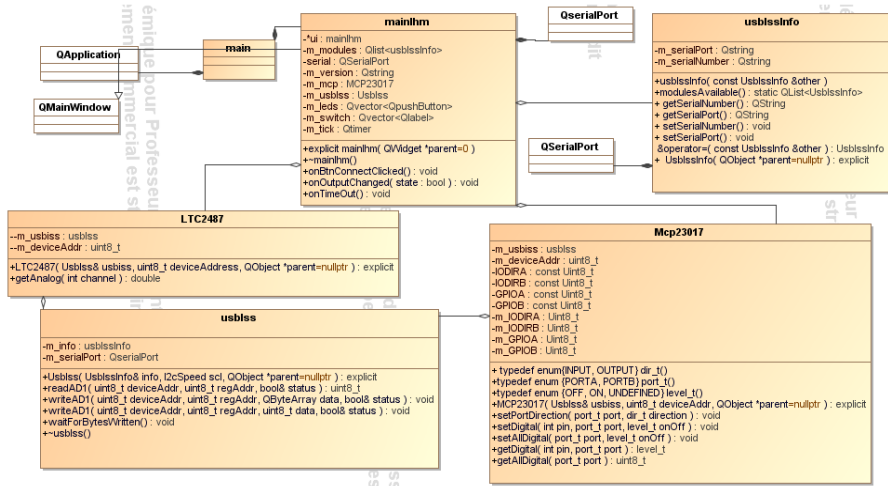


Tout d'abord afin de représenter dans l'ihm les valeurs numériques retourner par le composant LTC2487 il a été décidé d'utiliser un widget **LCD number** .

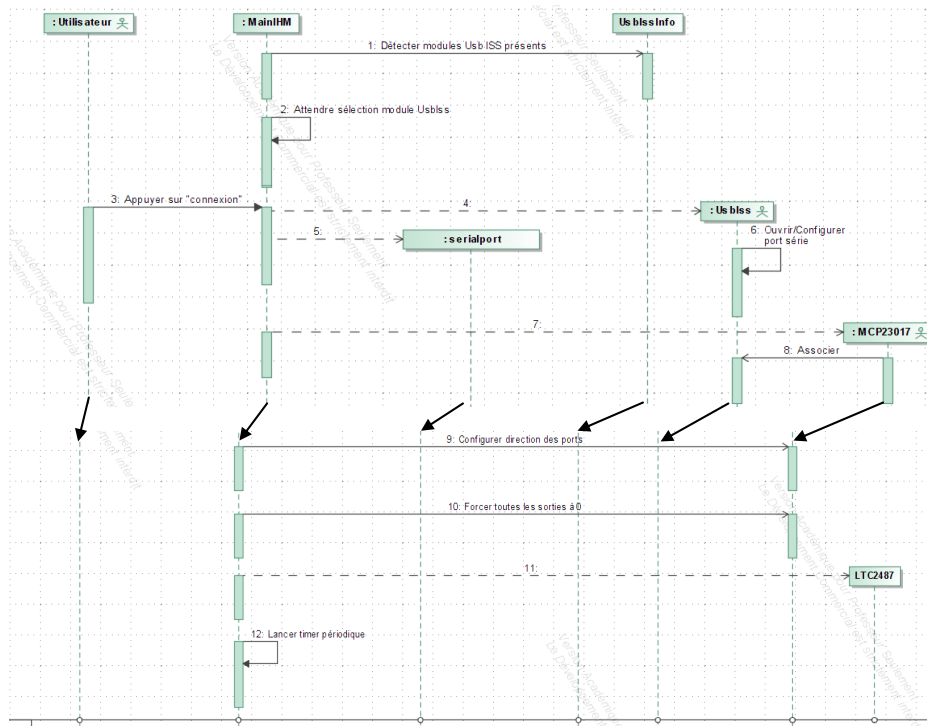
Ce widget permet d'afficher une valeur décimale, hexadécimale, octet ou encore binaire dans n'importe quelle taille.

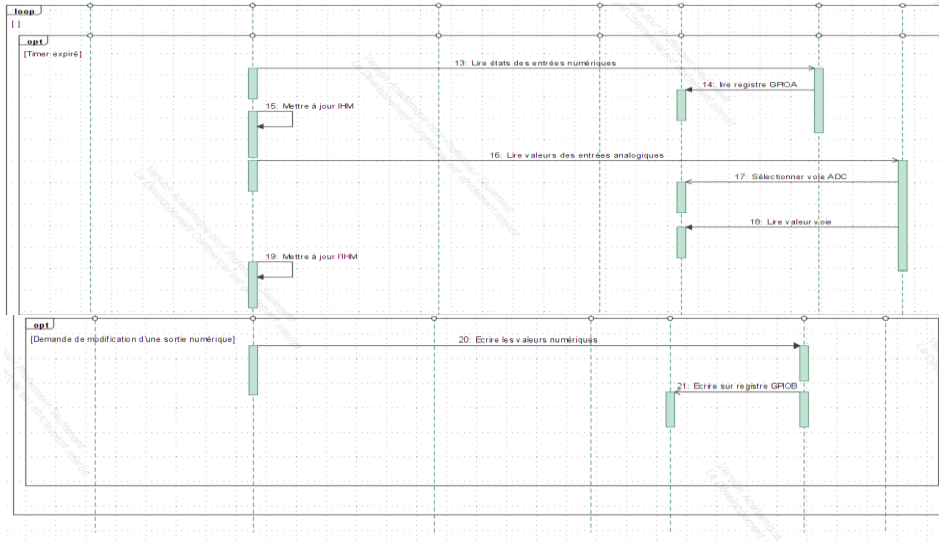
Un slider horizontale à également été utilisé avec pour valeur 0 et 5v, il varie selon la valeur retournée il permet de voir le % de la tension retourné sachant qu'a 5v la tension let le slider à sa valeur maximale.

Ci-dessous le diagramme des classes de mon application en rapport avec la gestion des entrées analogiques :



Le fonctionnement général de l'application est expliqué ci-dessous sous forme d'un diagramme de séquence :





Tout d'abord la classe LTC2487 n'as pas besoin d'effectuer une configuration sur le module comparé à la mcp23017.

Elle a donc juste une méthode `getAnalogue( int channel)` qui permet de sélectionner une voie et de la lire.

```

LTC2487::LTC2487(UsbIss& usbiss, uint8_t deviceAddress, QObject *parent) :
QObject(parent)
, m_usbiss(usbiss), m_deviceAddr(deviceAddress)
{
}

double LTC2487::getAnalogue(int channel) {
    static uint8_t muxAddress[] = {0xb0, 0xb8, 0xb1, 0xb9};
    QByteArray request;
    QByteArray response;
    bool ok;
    uint16_t ana;

    request.append(muxAddress[ channel ]);
    request.append(0x88);

    m_usbiss.writeAD0(m_deviceAddr, request, ok);

    response = m_usbiss.readAD0(m_deviceAddr, 3, ok);

    qDebug() << "valeur convertisseur : " + QString(response.toHex());

    // xx00 0000 0000 0000 00xx xxxx
    ana = (response.at(2) >> 6) & 0x03;
    ana += response.at(1) << 2;
    ana += (response.at(0) & 0x3F) << 10;

    return ana/65535.0 * 5.0;
}
  
```

1. Explications du code :
2. Tout d'abord :
3. On définit un objet muxAdress qui prend les octets des voies à l'intérieur d'un tableau
4. On crée 2 objets request et response
5. Création de l'objet en base 15 qui contiendra la valeur a retourné à l'ihm
6. L'objet request se voit attribuer à la suite la valeur de l'adresse choisie
7. La configuration également
8. On sélectionne la voie à lire en appelant la méthode writeAD0() de la classe USBISS
9. On lit la valeur de la voie retournée encapsulé en 24 bits avec la méthode readAD0()
10. On isole les bits qui nous intéressent (16) sur les 24 présents
11. On effectue une opération afin de recevoir une valeur entre 0 et 5v

4.4.11 Travail restant

Dans ma partie il me reste :

- Faire l'installateur du programme
- Effectuer la rédaction d'un manuel utilisateur

Conclusion et ressenti du projet :

Tout d'abord ce projet c'est trouvé être un véritable challenge, au premiers abords il avait l'air plutôt simple mais c'est une fois que j'ai commencé à travailler dessus que je me suis rendu compte du contraire.

Ce projet m'a beaucoup apporté en terme d'expérience, à l'avenir je pense si l'on me représentait un projet de ce type pouvoir mieux m'adapté et le finir en temps et en heure. De plus il a également parfait mes connaissances sur la programmation en C++ et j'en ai beaucoup appris.

Enfin dans l'ensemble ce projet c'est relevé intéressant et cela ne me dérangerait pas de travailler a nouveau sur un projet de ce type.

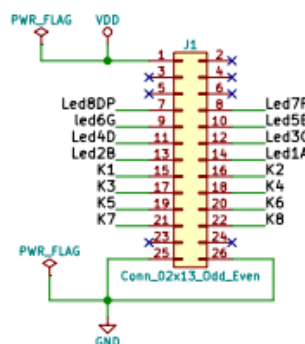
4.5 Annexe de travail

4.5.1 Journal de Bord

10/01/2018	Découverte du projet
11/01/2018	Suite étude du projet + installations logiciels
17/01/2018	Réunions de projet + études documentation I2C
18/01/2018	Suite études documentation I2C
19/01/2018	Diagramme d'état fait (au final non utilisé)
25/01/2018	Réalisation schéma prototypage
26/01/2018	Etude de nouveaux documents pour la réalisation de la plaquette + modification sur le schéma. Câblage plaquette fait
01/02/2018	Test final de la carte avec commandes I2C
14/02/2018	Début codage
15/02/2018	Etude de codes internet + cours

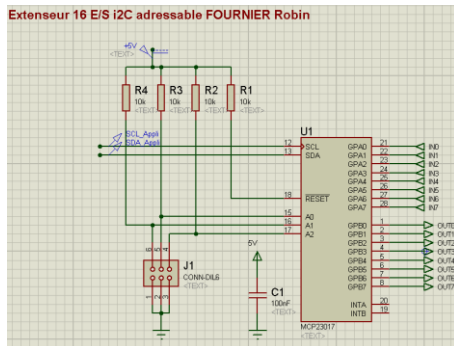
16/02/2018	Suite codages (progrès réalisé)
19/02/2018	Code prototype terminé
22/02/2018	Travail ihm sur le prototype
22/02/2018	Début Classe en C++ pour la partie numérique
22/02/2018	Classe en C++suite
23/02/2018	Classe en C++suite
14/03/2018	Préparation revue de projet 1
15/03/2018	Préparation revue de projet 1
16/03/2018	Préparation revue de projet 1
21/03/2018	Etudes des classes nécessaires pour commander les sorties numériques et acquérir les entrées numériques. Début diagramme de classe
22/03/2018	Diagramme de classe et diagramme de séquence
23/03/2018	Codage classes numérique
28/03/2018	Codages classe numérique
29/03/2018	Codages classe numérique
30/03/2018	Codages classe numérique
04/04/2018	Codages classe numérique
05/04/2018	Codages classe numérique
06/04/2018	Codages classe numérique
11/04/2018	Fin codage classe numérique + réalisation IHM
12/04/2018	Etude dépôt de logiciel création compte Atlassian
13/04/2018	Connexion au dépôt et premier comit
18/04/2018	Accueil entreprise API
19/04/2018	Journée banalisé bts blanc
20/04/2018	Etude data sheets composant analogique
02/05/2018	Classe entrée analogique
04/05/2018	Classe entrée analogique
09/05/2018	Classe entrée analogique
10/05/2018	Classe entrée analogique
11/05/2018	Réalisation IHM + test programme fonctionnelle
16/05/2018	Révision épreuve + carte xmind pour la revue
17/05/2018	Epreuve BTS finale
18/05/2018	Préparation revue(2) finale écrite
23/04/2018	Revue 2 orale
24/05/2018	Préparation revue finale

4.5.2 Schémas



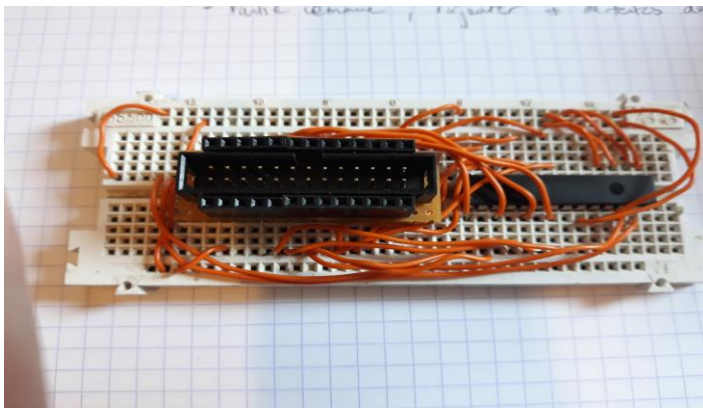
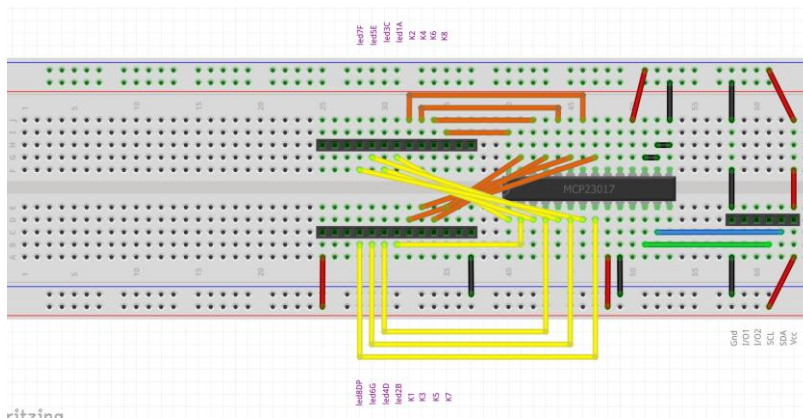
Il s'agit du câblage de la platine commandée par l'expander de port numérique MCP23017

La broche 1 est utilisée pour la tension d'alimentation, les broches 25 ou 26 pour la masse. Les broches Led sont utilisés pour les sorties numérique, et enfin les broches K pour les entrées numériques



Il s'agit d'un schéma remis au début du projet par un membre du groupe. Il s'agit du schéma de câblage du composant MCP23017.

C'est avec l'aide de ces 2 schémas que la plaquette de test a pu être mis en place.



Partie personnelle étudiant n°2

Thomas VIAN

(Option EC)

Etudiant EC 2 (VIAN Thomas)

1) Présentation du projet



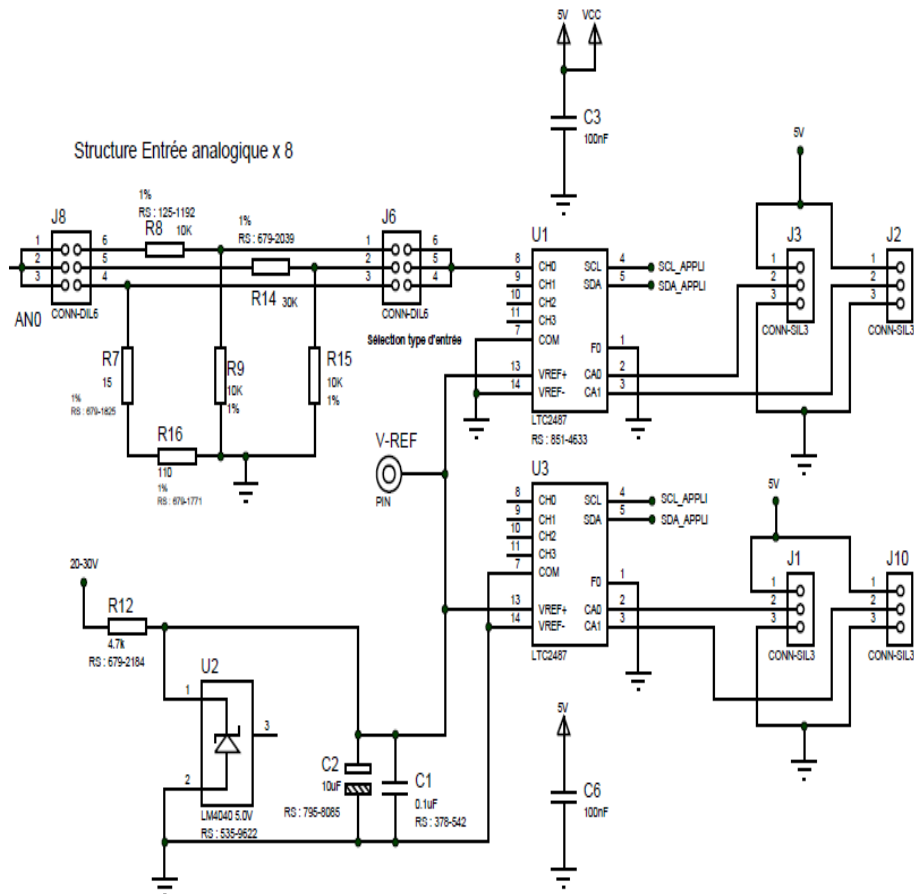
L'entreprise API souhaite disposer d'une carte polyvalente qui lui permet de développer rapidement des applications pour ses clients.

Cette carte devra disposer d'entrées logiques et analogiques, et de sorties à relais.

L'entreprise API souhaite disposer d'une carte polyvalente lui permettant de développer rapidement des applications pour ses clients.

- Cette carte devra disposer d'entrées logiques et analogiques, et de sorties à relais.
- Le pilotage de la carte devra se faire par bus I2C.
- Plusieurs cartes du même type devront pouvoir être chaînées. L'adresse I2C des circuits sur les cartes devra donc pouvoir se faire facilement.
- La partie commande pouvant être réalisée par une carte Arduino, une carte Raspberry Pi, ou un PC disposant d'un adaptateur I2C, la carte développée devra s'adapter aux différentes amplitudes des tensions sur le bus (intégrer un level shifter).
- Une application graphique de test/mise en œuvre rapide sera fournie afin de pouvoir piloter toutes les E/S de la carte

Schéma analogique API IO final

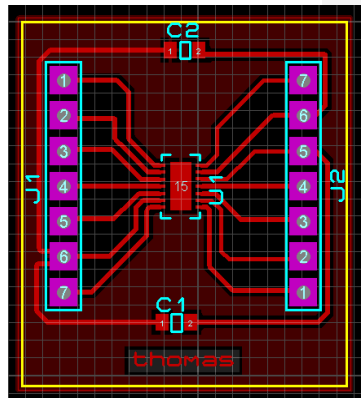
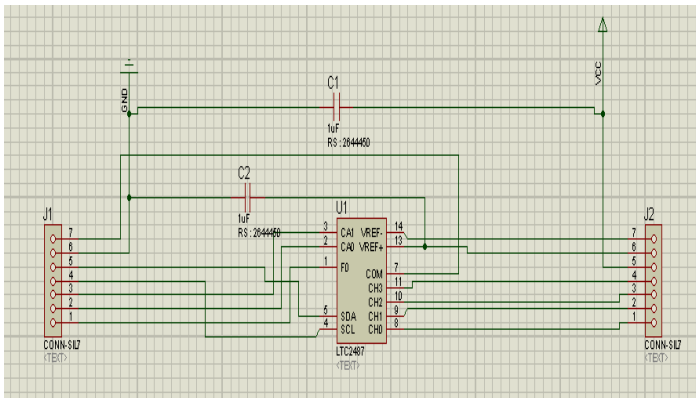


Mr HORTOLLAND a ainsi réalise un schéma analogique pour bien comprendre le fonctionnement du projet pour la parti analogique.

GANTT

Mode	Nom de la tâche	Durée	Début	Fin	Prédécess	Noms ressources
	Projet	139 hr	Jeu 11/01/18	Jeu 12/04/18		Thomas Vian; Mathieu Macker; Robin Fournier
	Tâches Communes	9 hr	Jeu 11/01/18	Ven 12/01/18		Mathieu Macker; Robin Fournier; Thomas Vian
	Lecture dossier	2 hr	Jeu 11/01/18	Jeu 11/01/18		Mathieu Macker; Robin Fournier; Thomas Vian
	Analyse de l'existant	1 hr	Jeu 11/01/18	Jeu 11/01/18	3	Mathieu Macker; Robin Fournier; Thomas Vian
	Réunion de synthèse	1 hr	Jeu 11/01/18	Jeu 11/01/18	4	Mathieu Macker; Robin Fournier; Thomas Vian
	Réaliser un plan des itérations	2 hr	Jeu 11/01/18	Jeu 11/01/18	5	Mathieu Macker; Robin Fournier; Thomas Vian
	Réaliser un planning prévisionnel	4 hr	Jeu 11/01/18	Ven 12/01/18	6	Mathieu Macker; Robin Fournier; Thomas Vian
	Tâches EC 2	130 hr	Mer 17/01/18	Jeu 12/04/18		Thomas Vian
	Analyser le schéma proposé	10 hr	Mer 17/01/18	Jeu 18/01/18	7	Thomas Vian
	Valider le choix d'un convertisseur analogique / numérique	35 hr	Ven 19/01/18	Jeu 08/02/18	9	Thomas Vian
	Concevoir/Réaliser/Tester une carte s'insérant dans un rail DIN de largeur 72mm	30 hr	Jeu 08/02/18	Mer 14/03/18	10	Thomas Vian
	Développer une application permettant à partir d'une carte Raspberry Pi de visualiser la valeur des entrées analogique	30 hr	Mer 14/03/18	Jeu 29/03/18	11	Thomas Vian
	Développer une application permettant à partir d'une carte arduino de visualiser la valeur des entrées analogiques	25 hr	Jeu 29/03/18	Jeu 12/04/18	12	Thomas Vian

Test ADC+Routages



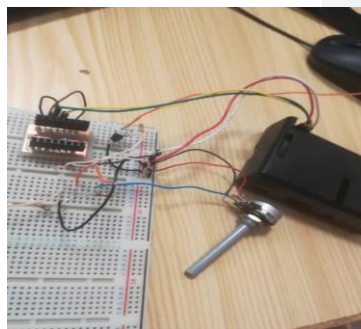
Réalisation du schéma ISIS et du routage ARES

Réalisation du câblage test avec un

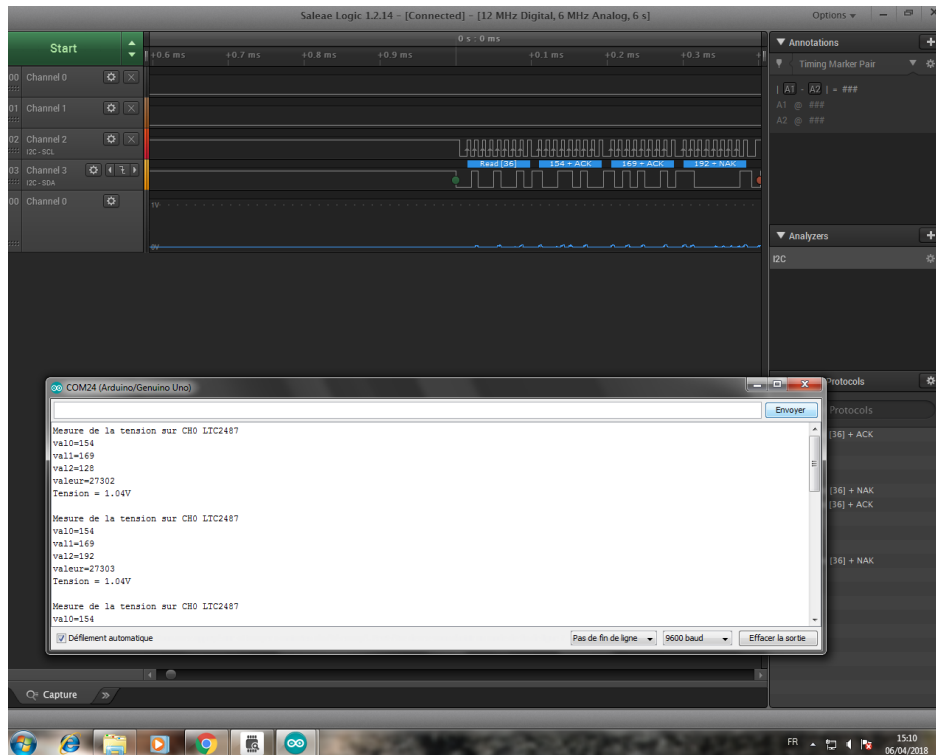
Potentiomètre Afin de varier la tension.

Se câblages est reliée en 5v grâce a la raspberry Pi.

Se câblage consiste à savoir si on obtiens 5V en sortie.



Etude de trame I2C de la carte ADC



Valeur décimal	Valeur en hexadécimal	Valeur en bits
154	9A	1001 1010
169	A9	1010 1001
192	C0	1100 0000

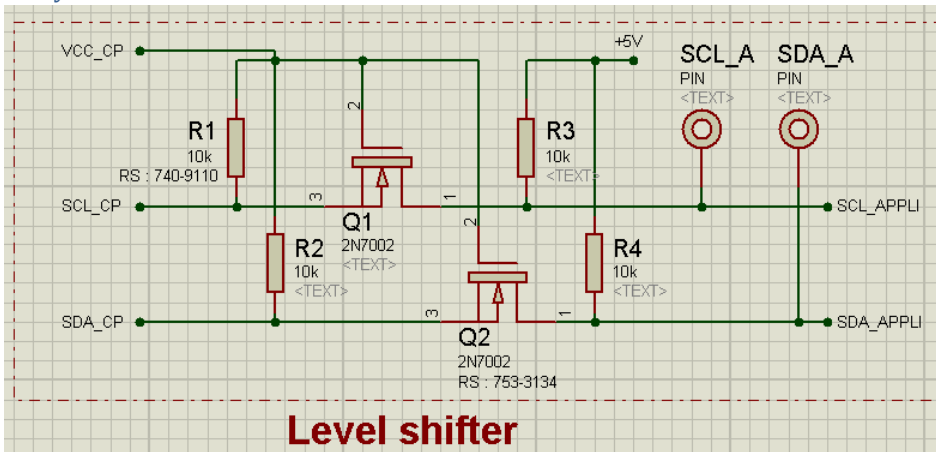
1001 1010 1010 1001 1100 0000

27303

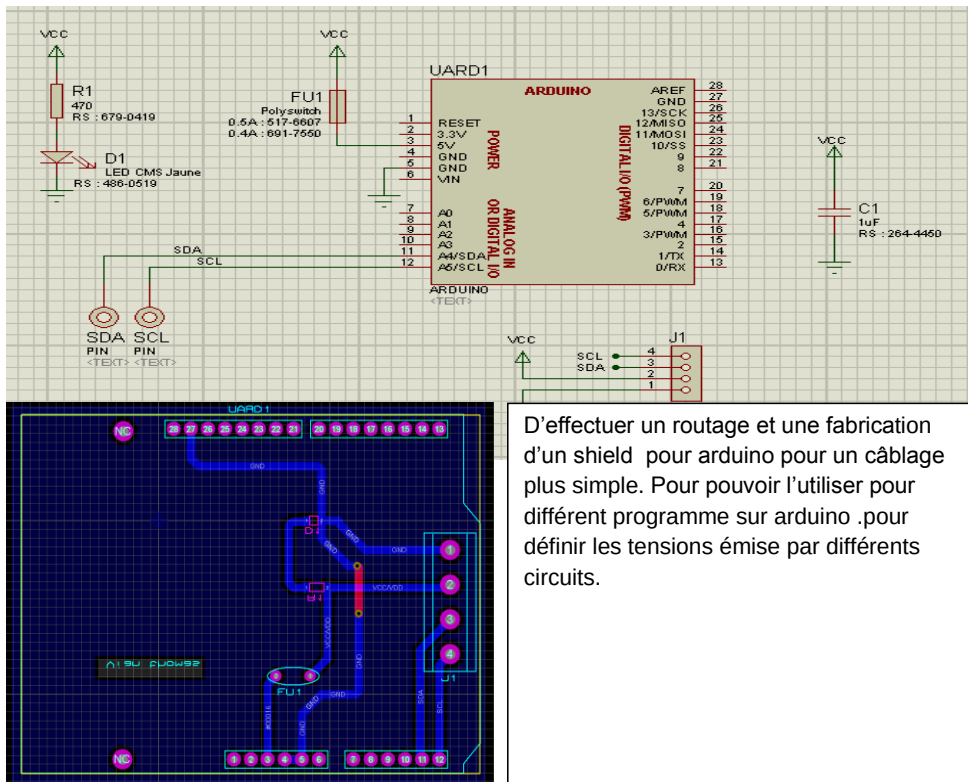
Valeur en décimal

$27303 \times 2.5/2^{16} = 1.04V$ tension obtenue sur arduino et saleae.

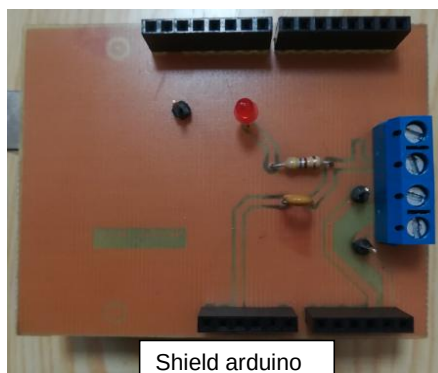
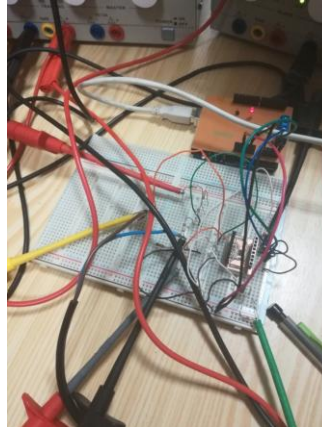
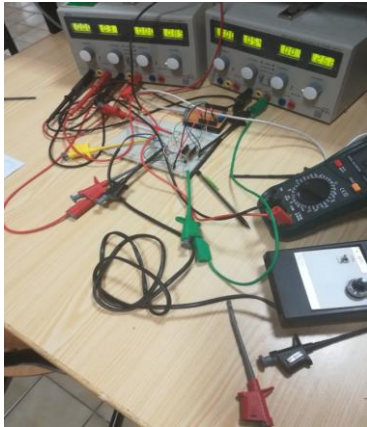
Analyse des schémas



Le level shifter permet de s'adapter en tension afin d'éviter les problèmes de différence de tension entre l'Arduino 5V et la Raspberry 3.3V.



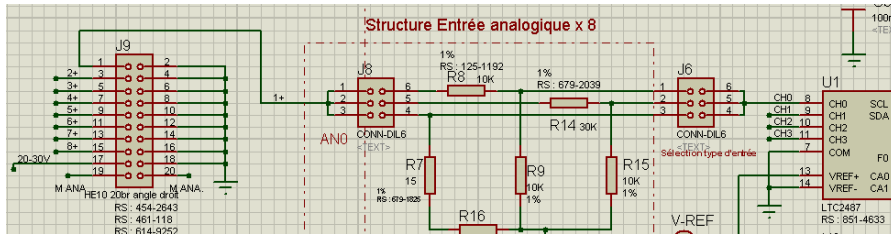
Test de la tension sur arduino



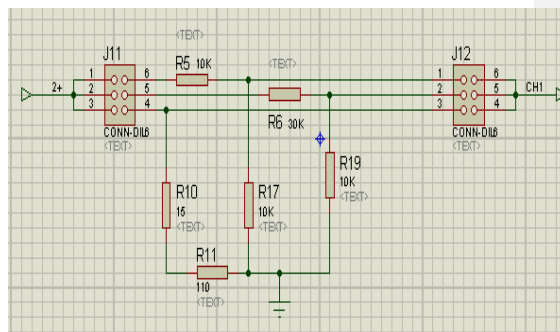
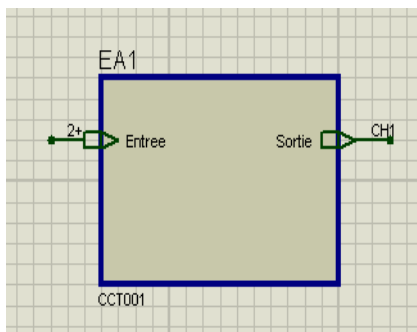
Mon but était de raccorder 3 voltmètres sur 2 alimentations externes. Une alimentation externe en 5 volt mais aussi de 10 volt puis une alimentation en 20V a 30V en entrée ensuite l'entreprise API nous a fournie un générateur d'intensité de 20 mA pour varier l'intensité du circuit afin de pouvoir faire varier l'intensité du circuit.

Le programme arduino permet directement de récupérer les tensions émises par l'alimentation. Surtout il me permet d'avoir une meilleure précision sur les valeurs obtenues par rapport aux voltmètres

Schéma structurel



Mon schéma de principe a besoin de x8 Entrée Analogique. Pour faciliter le schéma on crée x8 sous-circuits.



Un sous-circuit consiste à refaire une structure entrée analogique du schéma pour ma part et l'insérer dans un sous-circuit. Ensuite on a juste à changer les différents LBL du circuit pour les E/S du schéma de principe.

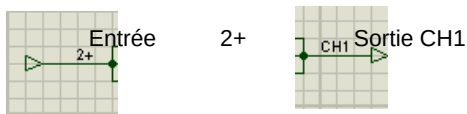
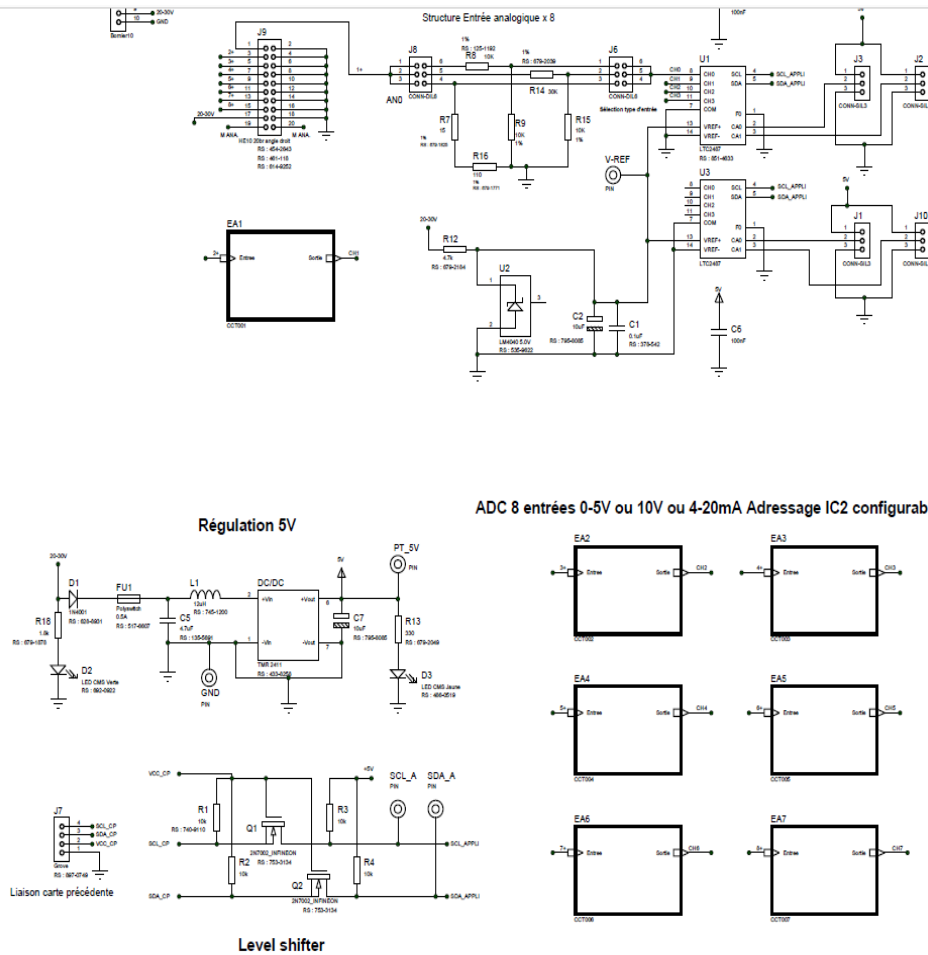


Schéma de structurel Analogique



Resistors

28	R1-R5,R8,R9,R15, R17,R19,R20, R24-R26,R30-R32, R36-R38,R42-R44, R48-R50,R54,R55	10k
8	R6,R14,R21,R27,R33, R39,R45,R51	30K
8	R7,R10,R22,R28,R34, R40,R46,R52	15
8	R11,R16,R23,R29, R35,R41,R47,R53	110
1	R12	4.7k
1	R13	330
1	R18	1.8k

Capacitors

1	C1	0.1uF
2	C2,C7	10uF
3	C3,C4,C6	100nF
1	C5	4.7uF

Integrated Circuits

2	U1,U3	LTC2487
1	U2	LM4040 5.0V

Transistors

2	Q1,Q2	2N7002_INFINEON
---	-------	-----------------

Diodes

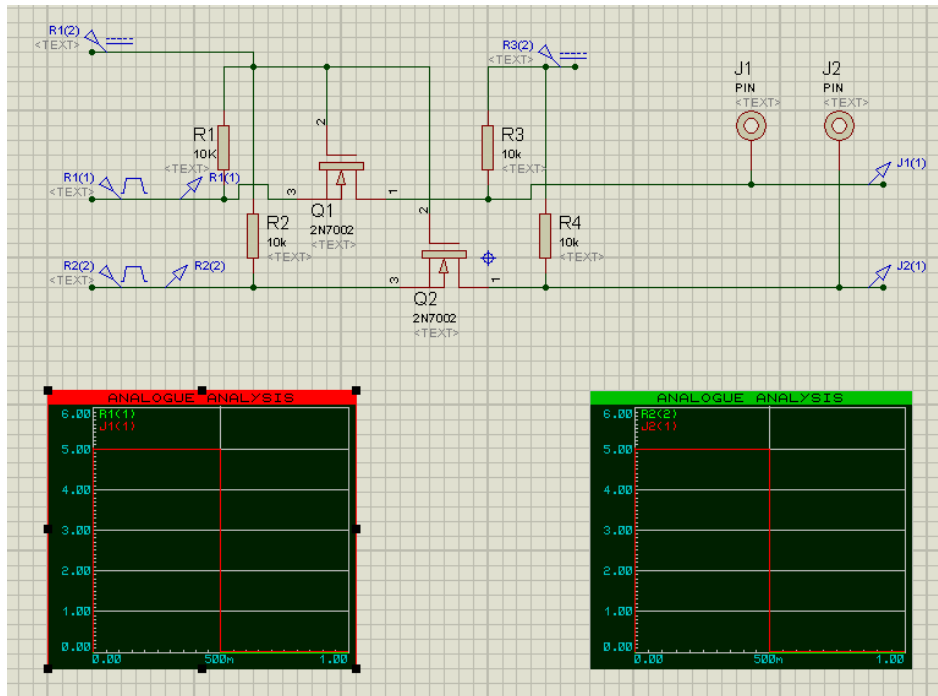
1	D1	1N4001
1	D2	LED CMS Verte
1	D3	LED CMS Jaune

Miscellaneous

1	DC/DC	TMR 2411
1	FU1	Polyswitch
5	GND,PT_5V,SCL_A, SDA_A,V-REF	PIN
4	J1-J3,J10	CONN-SIL3
1	J4	Bornier10
1	J5	Bornier à vis
16	J6,J8,J11-J24	CONN-DIL6
1	J7	Grove
1	J9	HE10 20br angle droi
1	L1	12uH

Liste du différent composant
utilisé pour le schéma structurel
sur ISIS

Présentation du Level shifter



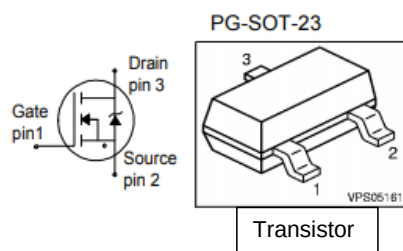
Utilisation du Level shifter pour arduino

Tension d'entrée : 5V

Tension de sortie : 5V

J1= SCL

J2=SDA



Partie personnelle étudiant n°3

Robin FOURNIER

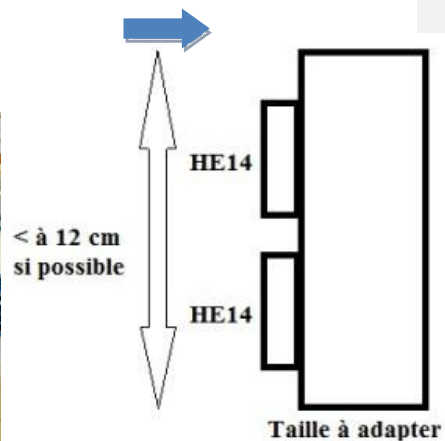
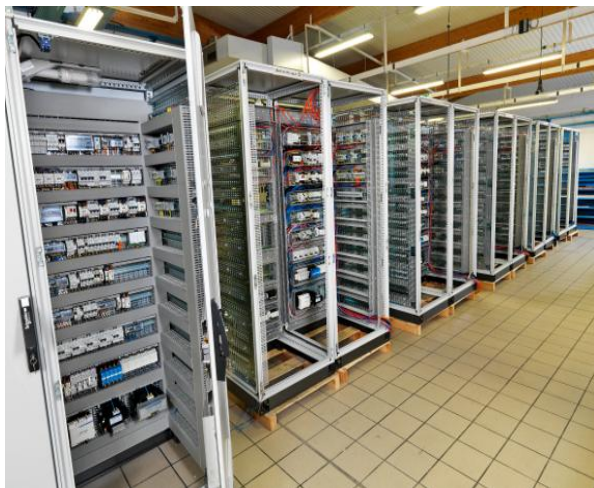
(Option EC)

6. FOURNIER Robin EC3

6.1 Introduction

Présentation de mon contrat :

Ma partie du contrat est de mettre en œuvre une carte polyvalente possédant 8 entrées et 8 sorties numériques. Le pilotage de la carte se fait par bus I2C. L'aménagement de la carte doit comprendre la possibilité de chaîner plusieurs cartes à la suite et de respecter une dimension de 12cm de hauteur afin de pouvoir rentrer dans les armoires industrielles de l'entreprise API. La partie commande est réalisée par une carte Raspberry Pi, par une carte Arduino ou un adaptateur USB/I2C. La carte doit s'adapter aux différentes amplitudes des tensions sur le bus il faut donc intégrer un level shifter (5V<-> 3.3V).

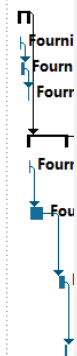


6.1 Introduction

Diagramme de GANTT Prévisionnel :



	Mode	Nom de la tâche	Durée	Début	Fin	F	Noms ressources	D	L	M	M	J	V
1		Durée Totale	131 hr	Jeu 11/01/18	Mar 20/03/18								
2		Début de Projet	4 hr	Jeu 11/01/18	Jeu 11/01/18								
3		Lecture de doss	1 hr	Jeu 11/01/18	Jeu 11/01/18		Fourmier Robin;Macker-Ca						
4		Analyse de l'exi	2 hr	Jeu 11/01/18	Jeu 11/01/18	3	Fourmier Robin;Macker-Ca						
5		Réunion de synthèse	1 hr	Jeu 11/01/18	Jeu 11/01/18	4	Fourmier Robin;Vian Thom Macker-Cavalier Mathieu						
6		Prévision	7 hr	Jeu 11/01/18	Ven 12/01/18	2							
7		Réaliser plan itérations	1 hr	Jeu 11/01/18	Jeu 11/01/18		Fourmier Robin; Macker-Cavalier Mathieu;						
8		Réaliser diagramme Gantt	2 hr	Jeu 11/01/18	Jeu 11/01/18	7	Fourmier Robin; Macker-Cavalier Mathieu; Vian Thomas						
9		Diagramme de Gantt (suite si non fini)	2 hr	Ven 12/01/18	Ven 12/01/18	8	Fourmier Robin; Macker-Cavalier Mathieu; Vian Thomas						
10		Diagramme SYSML	2 hr	Ven 12/01/18	Ven 12/01/18	9	Fourmier Robin; Macker-Cavalier Mathieu;						
11		Schéma et Documentation	23 hr	Mer 17/01/18	Mer 24/01/18	6							
12		Lecture Documentation Matériel	11 hr	Mer 17/01/18	Jeu 18/01/18		Fourmier Robin; Macker-Cavalier Mathieu; Vian Thomas						
13		Étude de schéma	12 hr	Jeu 18/01/18	Mer 24/01/18	12	Fourmier Robin; Macker-Cavalier Mathieu;						
14		Fabrication carte	40 hr	Jeu 25/01/18	Jeu 08/02/18	11							
15		Test Board	40 hr	Jeu 25/01/18	Jeu 08/02/18		Fourmier Robin;Vian Thom						
16		Fabrication	54 hr	Jeu 08/02/18	Mar 20/03/18	14							
17		Fabrication	54 hr	Jeu 08/02/18	Mar 20/03/18		Fourmier Robin;Vian Thom						
18		Revue de Projet 1	1 jour	Mer 21/03/18	Mer 21/03/18								



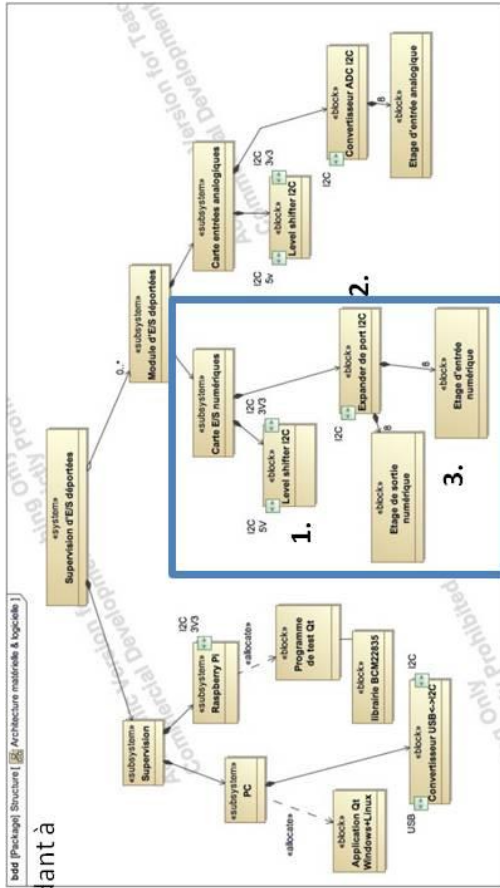
6.1 Introduction

Composants correspondant à chaque blocs:

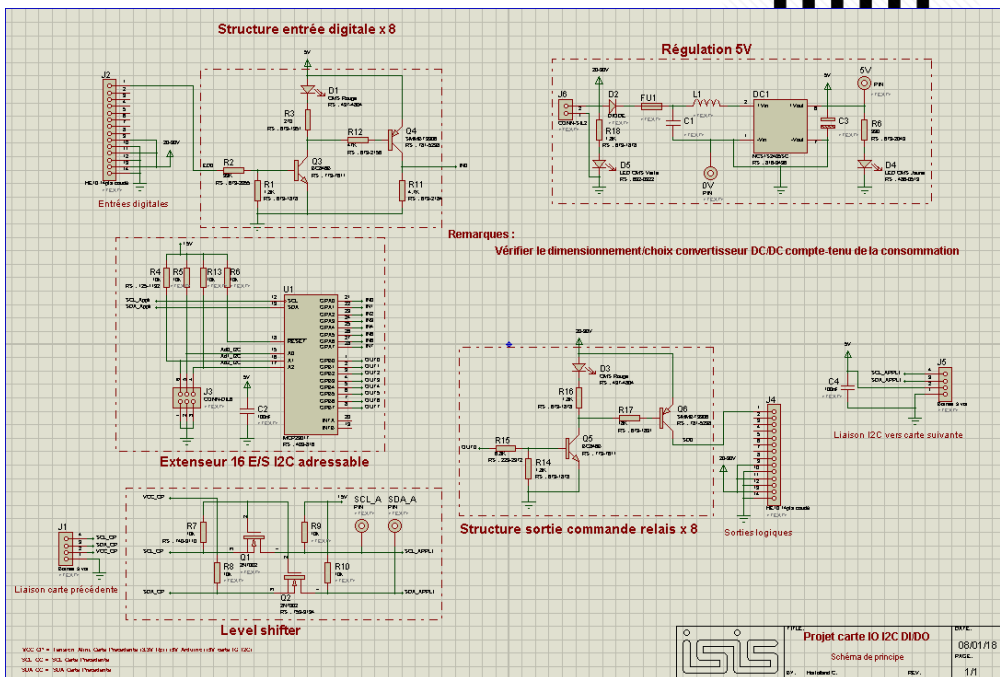
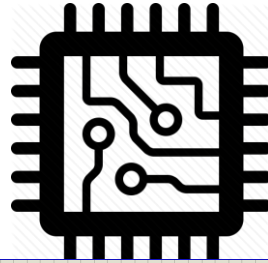
1. Level Shifter
-R7, R8, R10, R9
-Q1, Q2

2. Extenseur I2C
-MCP23017
-R4, R5, R13, R19
-J3

3. E/S digitales
-J1, J2
-5 RES pour entrées
-4 RES pour sorties
-2 TRANSISTORS pour E&S
-1 LED pour E&S



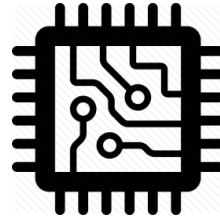
6.2 Base de travail



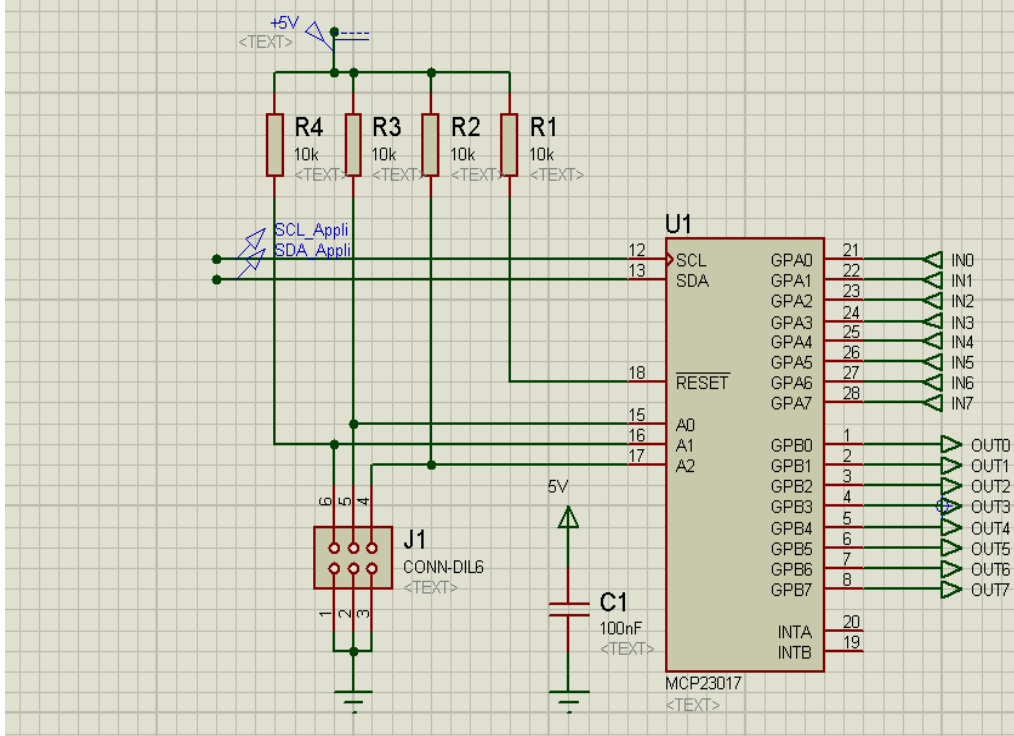
Voici le schéma de travail qui m'a été fourni. Il y est représenté la Structure d'entrée Digitale, la Régulation 5V, l'Extenseur 16 E/S, la Structure de sortie commande relais et le Level Shifter.

Ce schéma évoluera au fil du projet et sera modifier souvent.

6.3 Extenseur 16 E/S I2C

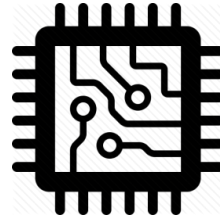
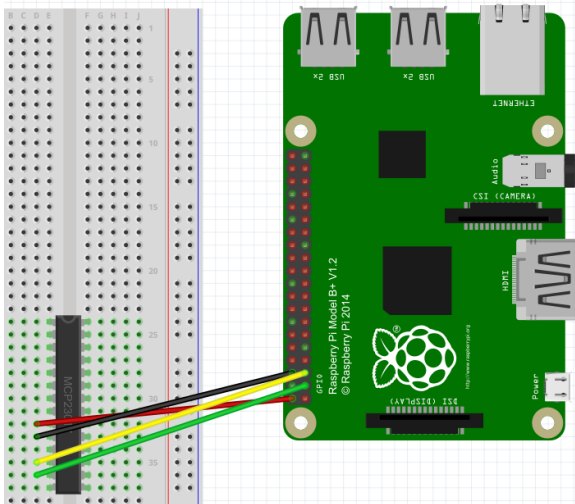


Extenseur 16 E/S I2C adressable FOURNIER Robin

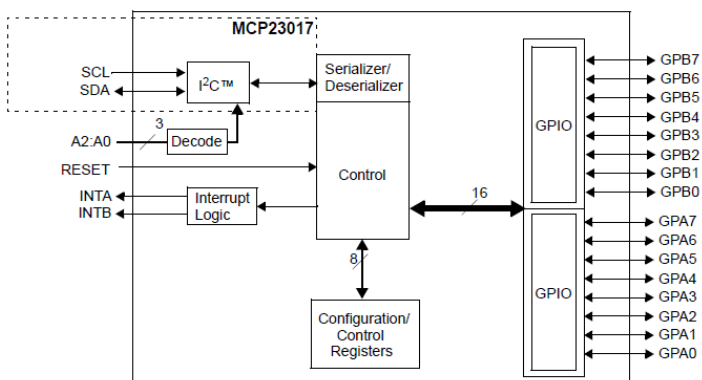
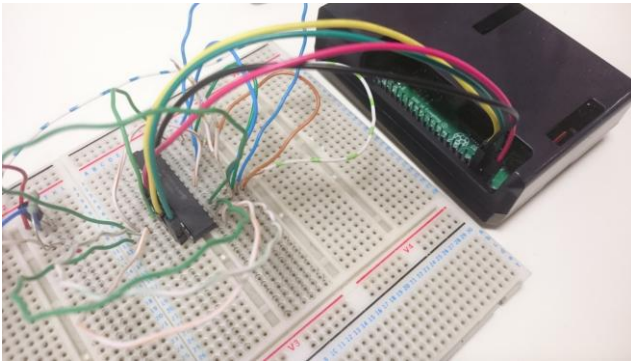


L'extenseur d'E/S est la partie connectée à la RaspBerry. On relie le MCP23017 aux 5V, GND, SDA, SCL de la RaspBerry. L'Extenseur E/S permet d'adresser les entrées / sorties, pour un 1 logique on a 5V et pour un 0 logique on a 0V. Le RESET est relié par défaut au 5V. Il y a 2^3 adresses disponible sur A0, A1, A2 ce qui fait 8 adresses.

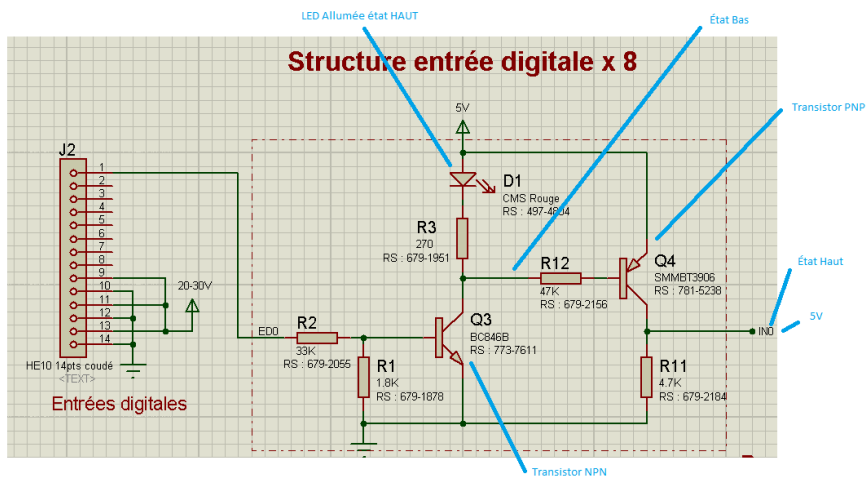
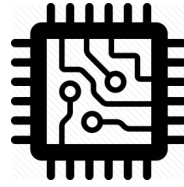
6.3 Extenseur 16 E/S I2C



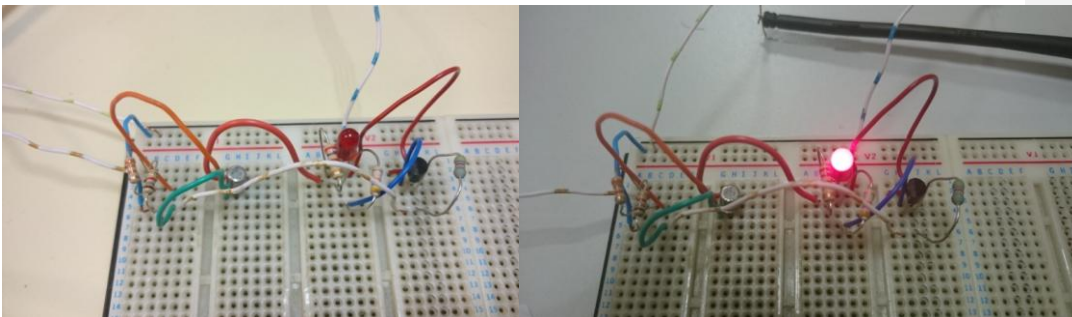
Voici comment est relié le MCP 23017 à la RaspBerry.



6.4 Structure entrées digitales

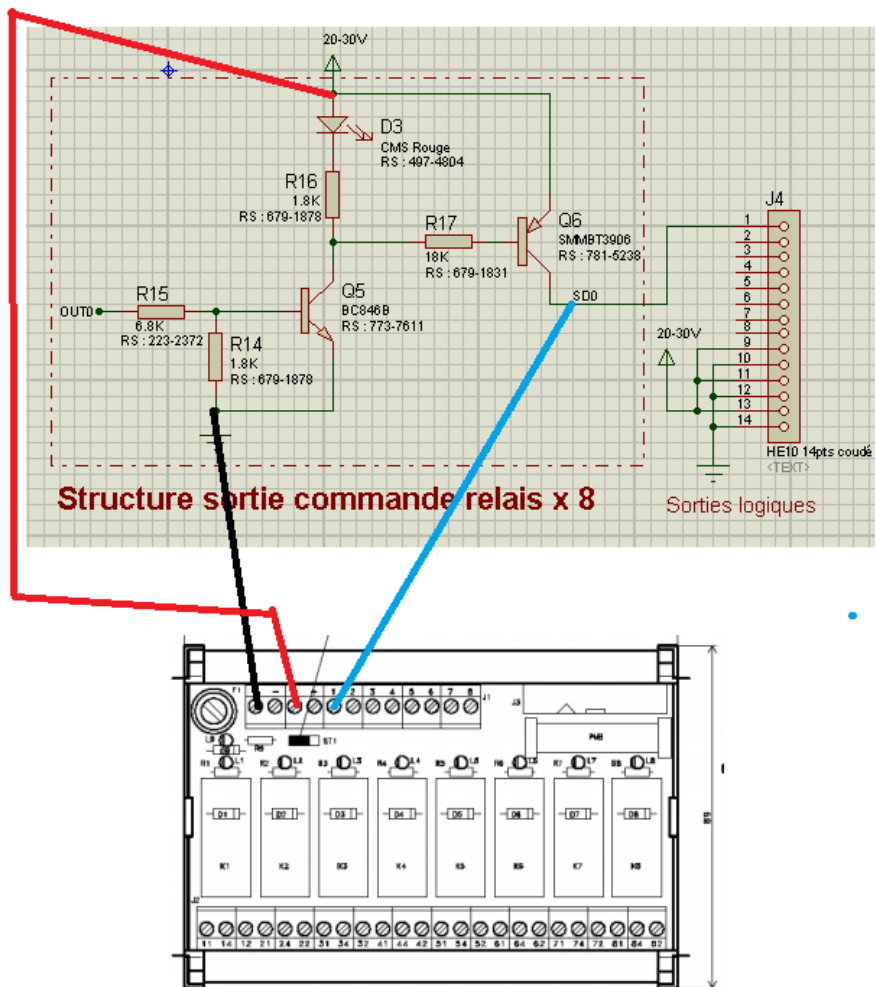
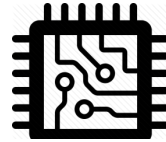


VDD	État Led	IN0
<12.5V	LOW	0V
>12.5V	HIGH	5V



Les 8 entrées digitales PNP 20 – 30V sont situées sur J2. Lorsque la tension VDD est supérieure à 12.5V Q3 commute et D1 s'allume permettant d'avoir la tension 5V aux bornes IN0.

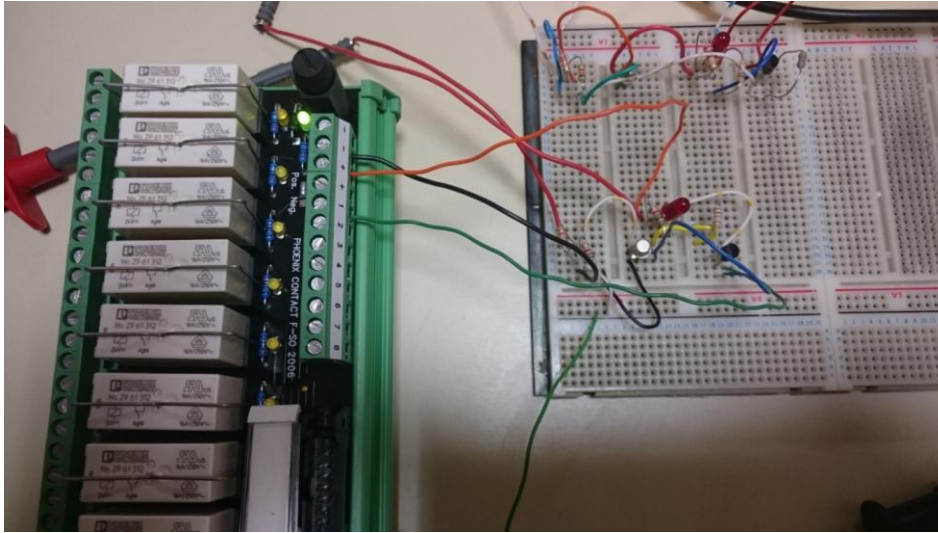
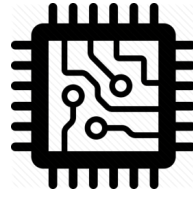
6.5 Structure sortie commande relais



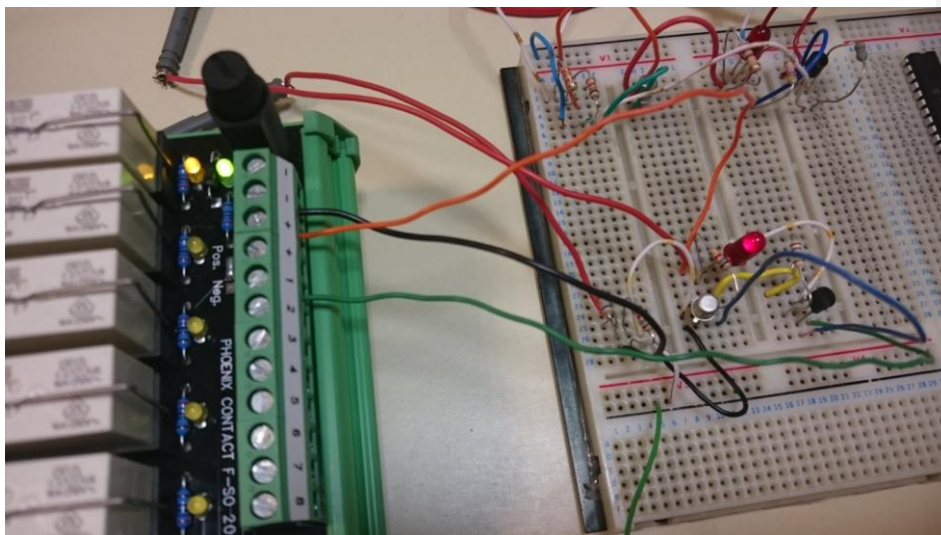
Cette structure est celle reliée au relai.

VDD	OUT 0	État Led
>24V	>2.5V	HIGH

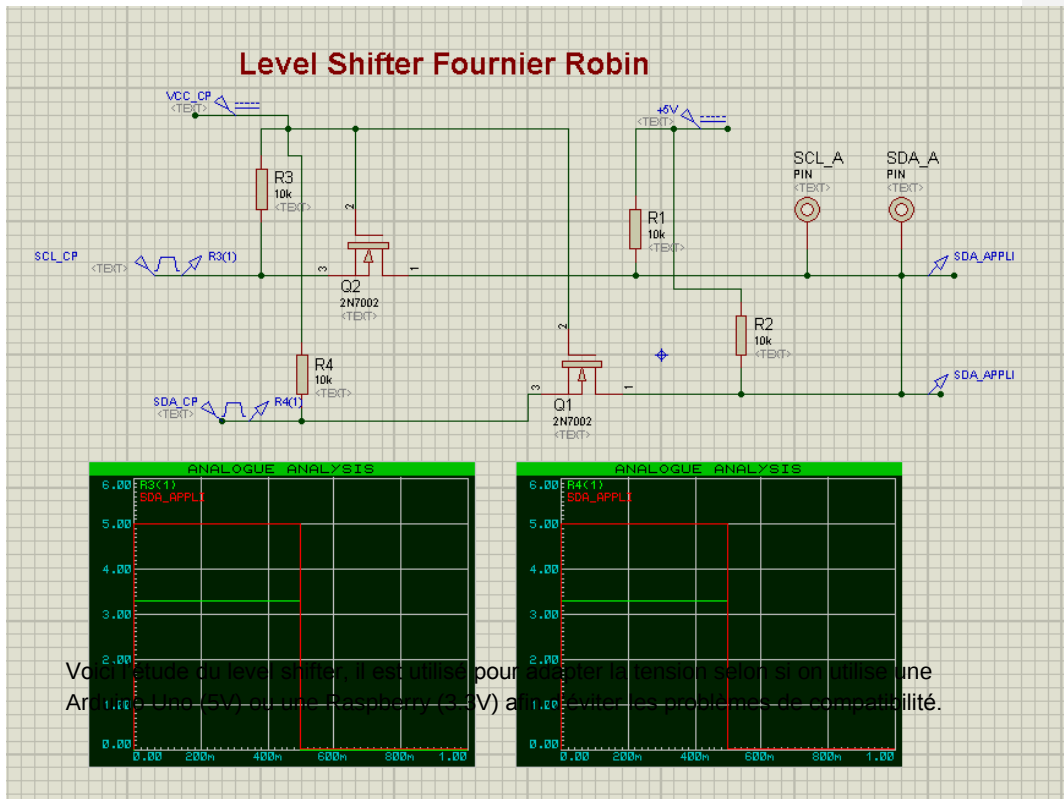
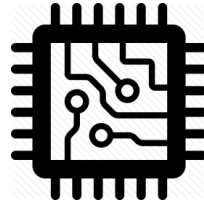
6.5 Structure sortie commande relais



Pour que le relai commute et que la Led s'allume il faut une tension nominale $V_{DD} = 24V$ et $OUT0 = 2.5V$



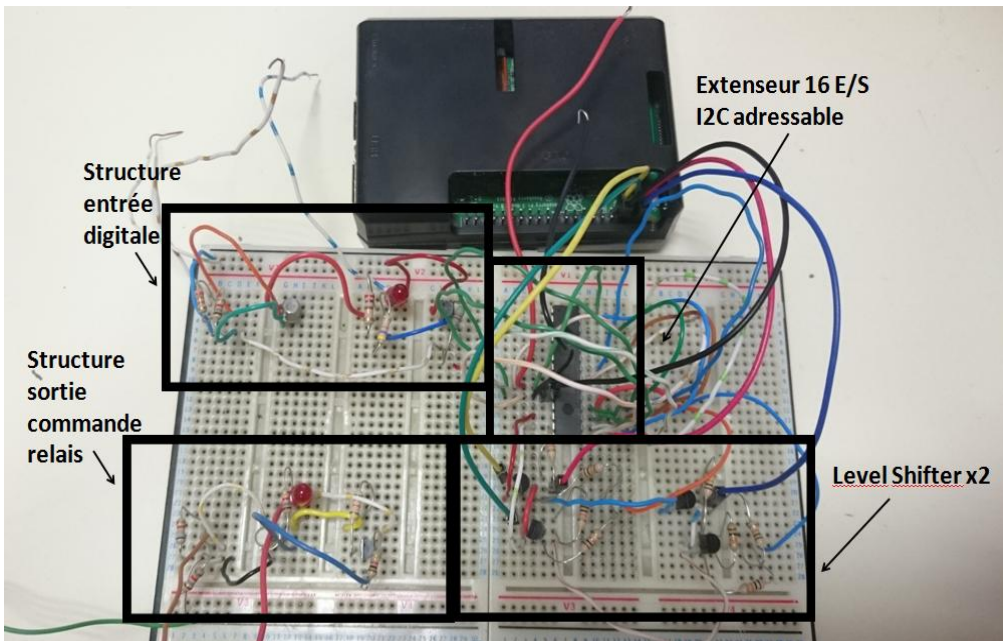
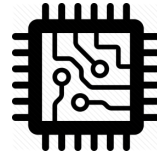
6.6 Level Shifter



Level Shifter

INPUT 5V — ? — OUTPUT 3.3V

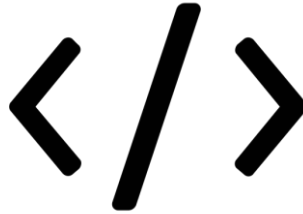
6.7 Résumé



Voici le montage complet en phase de test terminé.

L'ensemble du schéma fourni par le professeur a été validé. La Régulation 5V qui n'a pas été encore été abordée est simulée par une alimentation extérieure 5V.

6.8 Essais et Vérifications des Adresses



Programme permettant de voir en direct les changements d'états des bits lorsque l'on câble au 5V ou GND sur la carte TEST.

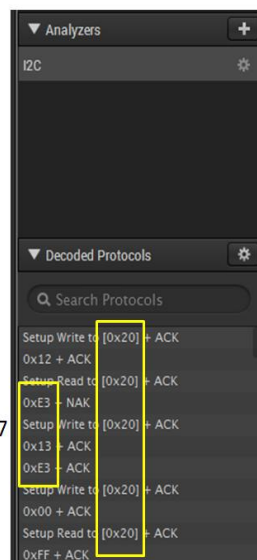
```
i2cIn[0] = 227
i2cOut[1] = 227

API IO Fournier
Simulation niveau entrées / sorties
Octet lus :
i2cIn[0] (initialisation) = 255
-----
i2cIn[0] = 227
i2cOut[1] = 227

API IO Fournier
Simulation niveau entrées / sorties
Octet lus :
i2cIn[0] (initialisation) = 255
-----
i2cIn[0] = 227
i2cOut[1] = 227

API IO Fournier
Simulation niveau entrées / sorties
Octet lus :
i2cIn[0] (initialisation) = 255
```

0xE3 = 227



0x20 Adresse
correspondante
au programme

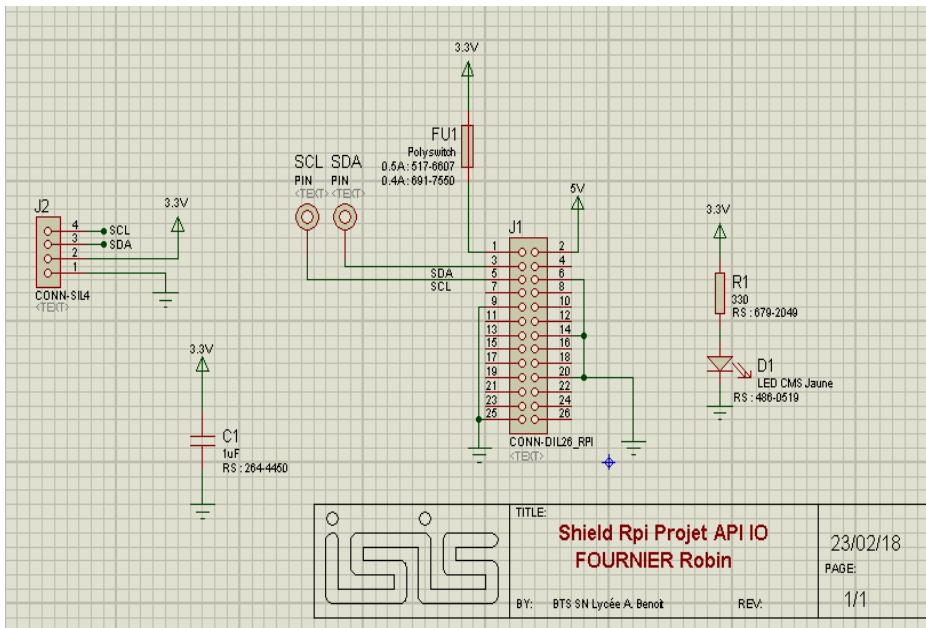
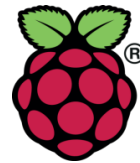
Voici la capture de donnée récupérée avec Saleae.

0x20 est l'adresse définie par défaut dans le programme fourni par le professeur.

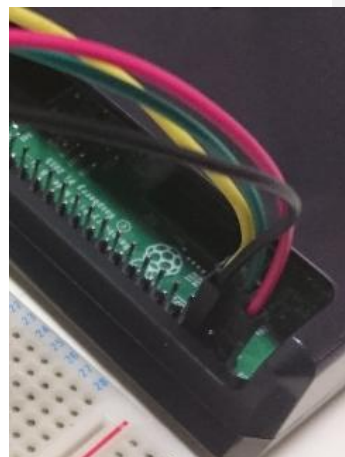
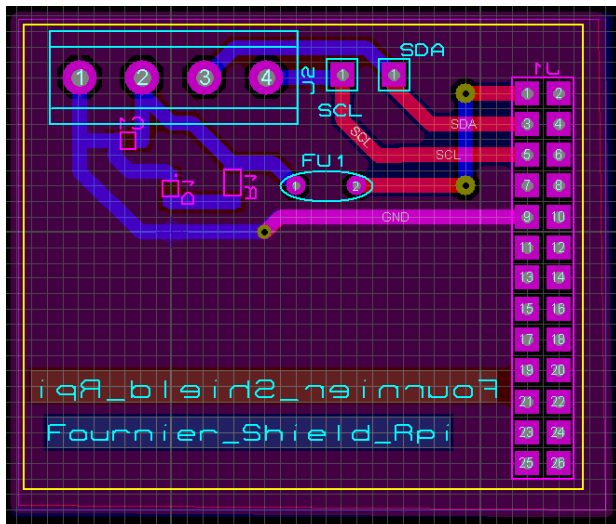
0xE3 est le résultat des entrées / sorties simulées en câblant pour un niveau logique 1 au 5V et pour un niveau logique 0 au GND.

En convertissant le résultat en binaire on obtient bien en direct les modifications sur les E/S en modifiant la connectique.

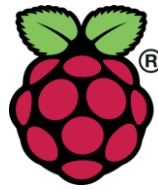
6.9 Shield RPI



La tâche suivante était de router et de fabriquer ensuite un shield pour la RaspBerry permettant un câblage plus rapide et plus facile.

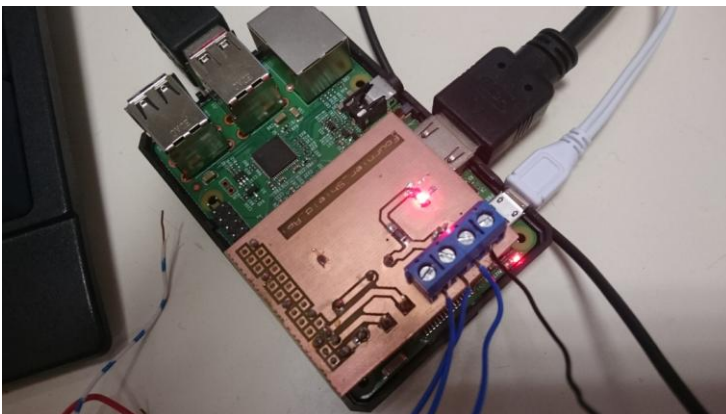


6.9 Shield RPI



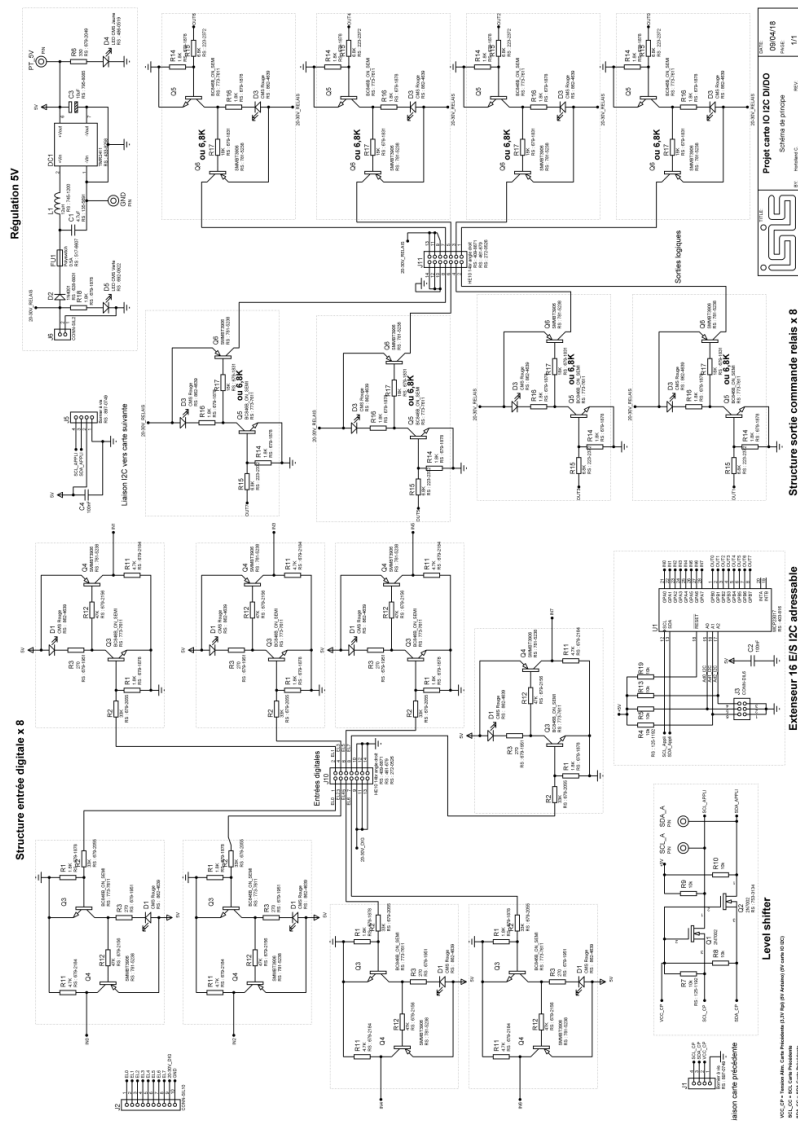
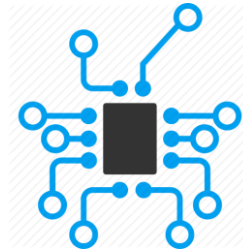
Voilà à quoi ressemble le Shield RPI une fois mis en œuvre.

Il permet de relier le SDA, SCL, 5V et GND de la carte test à la RPI .



6.10 Schéma de principe Entier Carte DI/DO

Avant Utilisation de Sous-Circuits



6.10 Schéma de principe Entier Carte DI/DO

Après Utilisation de Sous-Circuits



Mon schéma de principe après la multiplication des 8 entrées et 8 sorties numériques nécessaire au routage de la carte est devenu beaucoup trop encombrant et brouillon.

Afin de résoudre ce problème, j'ai trouvé la solution d'utiliser des sous-circuits sur ISIS Proteus.

Cette solution permet de

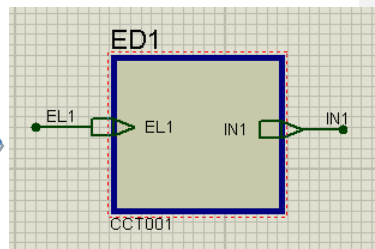
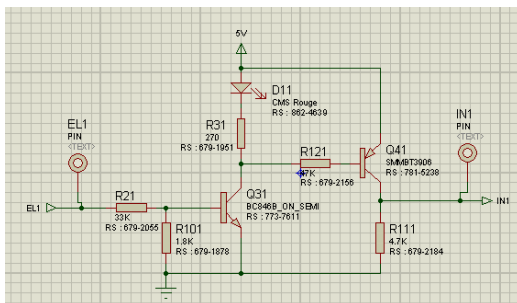
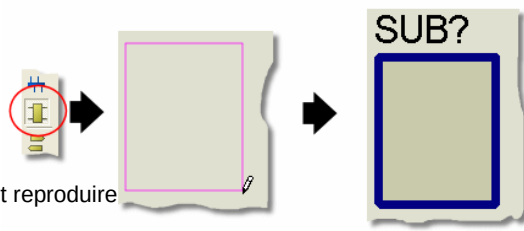
réduire considérablement le

volume que prennent les 8 E/S sur

le schéma. Il suffit de créer un sous-circuit et reproduire

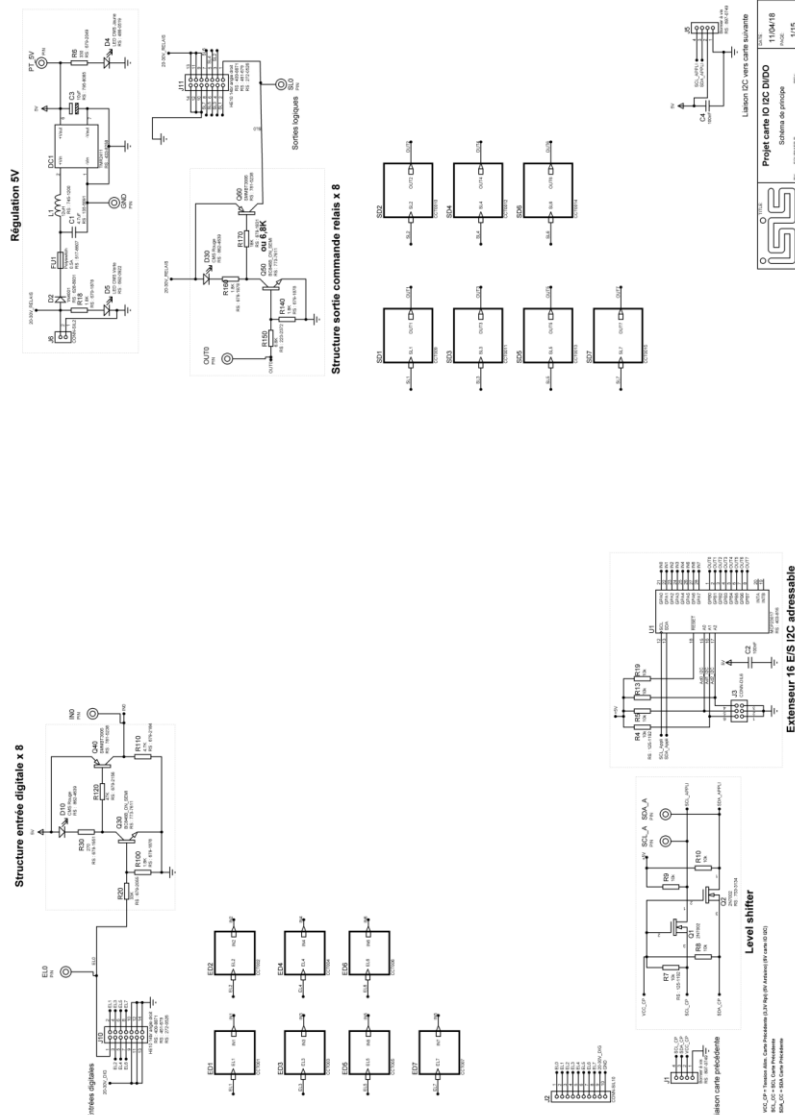
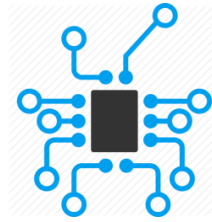
la structure et à l'insérer dans

le sous circuit, ensuite placer une entrée input et sortie output sur le bloc sous circuit et de les nommer avec des labels.



6.10 Schéma de principe Entier Carte DI/DO

Après Utilisation de Sous-Circuits



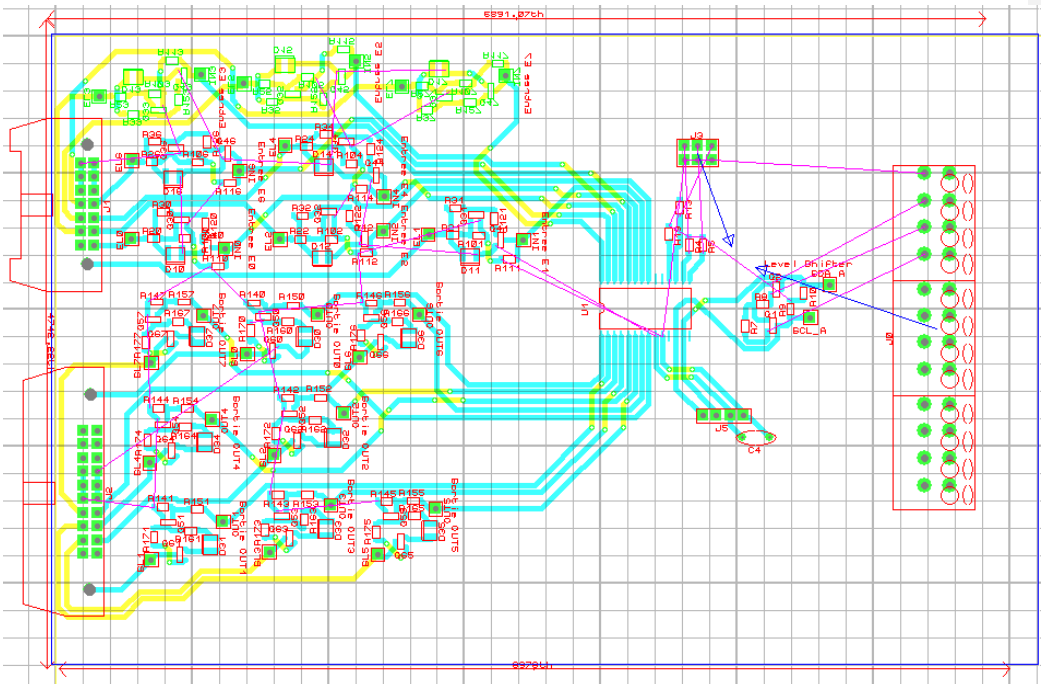
6.11 Routage Entier Carte DI/DO



La tâche finale de mon contrat est
de router mon schéma de principe.

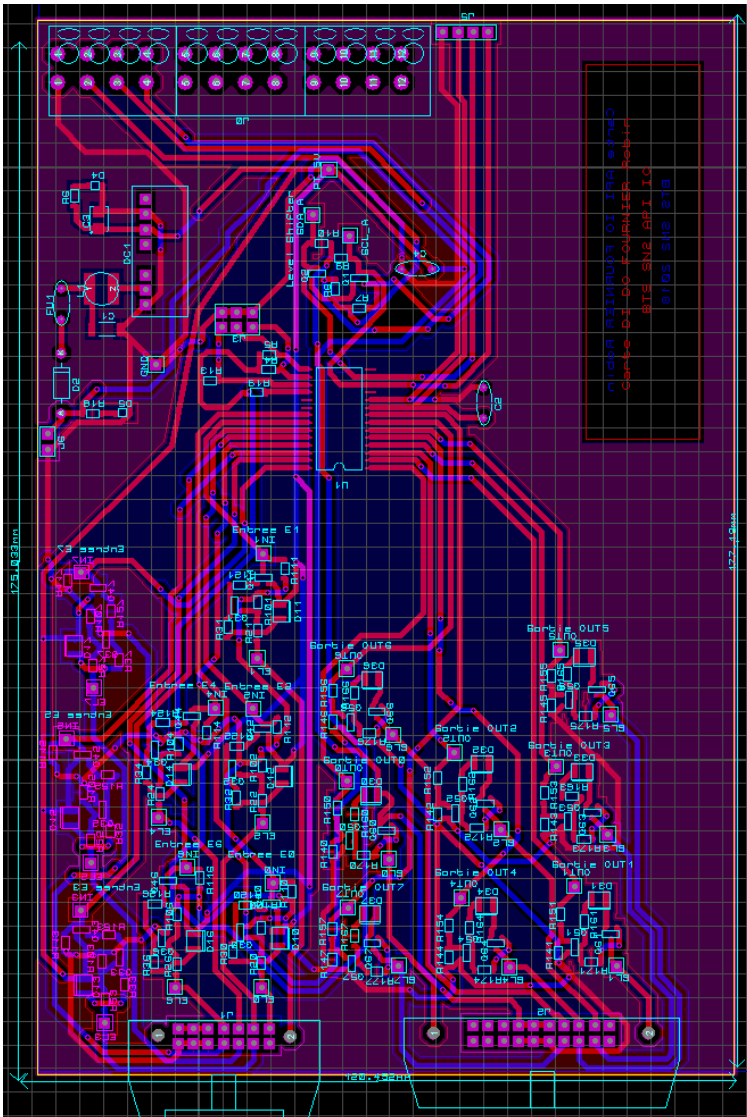
Pour se faire j'utilise le logiciel ARES. Le cahier des charges d'API m'impose de devoir faire une carte en longueur et d'une hauteur maximale de 12cm.

Je dois positionner 166 composants. Les connecteurs des entrées & sorties doivent être positionnés du même côté pour faciliter le branchement dans l'armoire industrielle.



État Routage carte le 18/05/18

6.12 Routage Entier Carte DI/D0



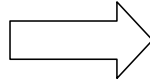
Routage terminé respectant les dimensions demandées par API. 12cm / 17.5cm

6.13 Problèmes rencontrés

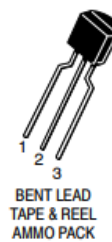
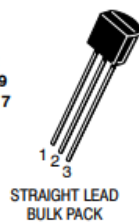
Problème de stock.

J'ai du remplacer pour certaine partie des composants.

Par exemple, j'ai du prendre l'équivalent du transistor 2N7002. Car il me fallait un composant traversant pour la breadboard.



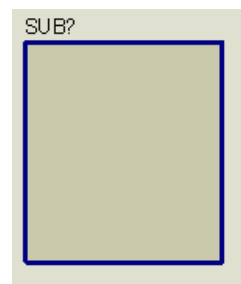
TO-92
CASE 29
STYLE 17



Manque de clareté et schéma structurel brouillon donc

Apprentissage d'une nouvelle fonctionnalité sur ISIS Proteus :

Les sous-circuits.



6.14 Partie Physique

Pour ma partie physique du projet, je me suis attribué

la tâche d'analyser la trame I²C de la communication entre le MCP23017 et la RPi.



Définition du Bus I²C :

Le bus I²C (Inter Integrated Circuit) fait partie des bus série : 3 fils pour faire tout passer. Il a été développé au début des années 1980, par **Philips**. Le bus I²C permet de faire communiquer entre eux des composants électroniques très divers grâce à seulement trois fils : un signal de données (SDA), un signal d'horloge (SCL), et un signal de référence électrique (masse).

Analyse Personnelle :

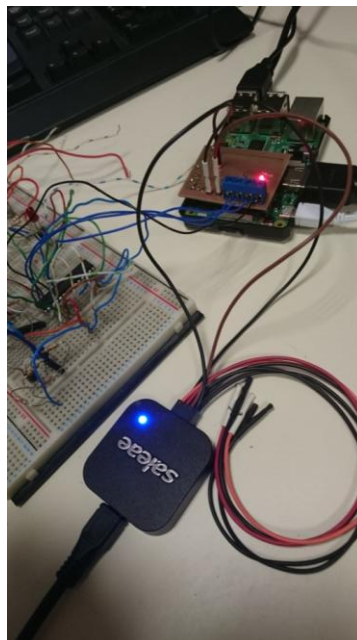
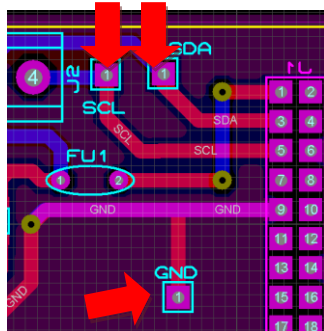
J'utilise Saleae pour analyser la trame.

le câblage de l'outil d'analyse est facilité

grâce aux points test ajoutés lors du

routage du

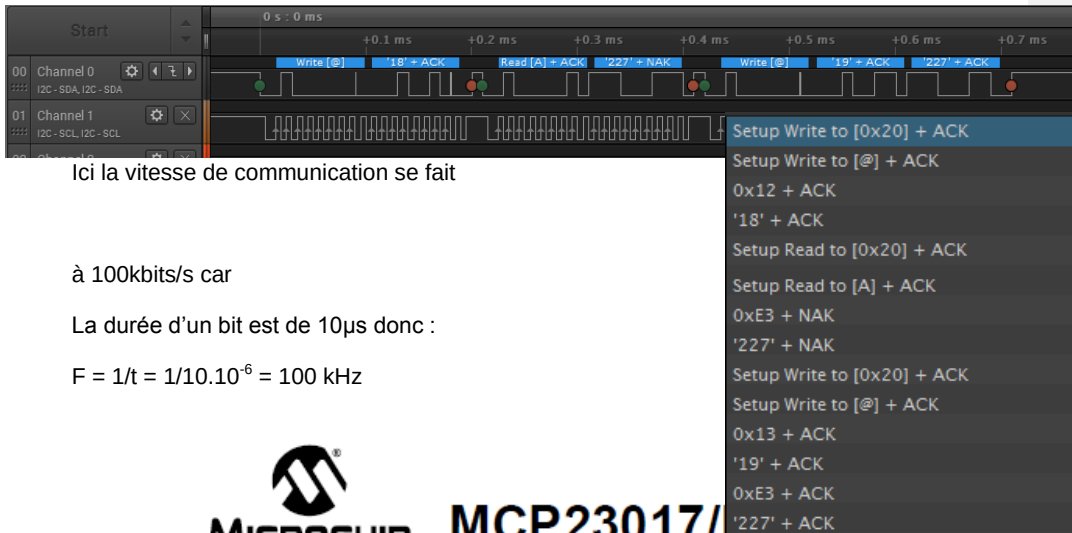
Shield RPi.



6.14 Partie Physique



Voici la trame analysée sur Saleae Logic Software,



Ici la vitesse de communication se fait

à 100kbits/s car

La durée d'un bit est de 10µs donc :

$$F = 1/t = 1/10 \cdot 10^{-6} = 100 \text{ kHz}$$

MICROCHIP MCP23017/MCP23017

16-Bit I/O Expander with Serial Interface

Features

- 16-bit remote bidirectional I/O port
- I/O pins default to input
- High-speed I²C™ interface (MCP23017)
 - 100 kHz
 - 400 kHz
 - 1.7 MHz

Package Types

PDIP, SOIC, SSOP

Le bus I²C est donc configuré en Standard Mode (100kbits).

C'est une communication en série synchrone avec l'horloge commune SCL.

6.14 Partie Physique

Les transistors bipolaires NPN & PNP

Le transistor permet notamment de réaliser deux grandes fonctions de l'électronique :
l'amplification et la commutation de signaux (interrupteur électronique).

type NPN



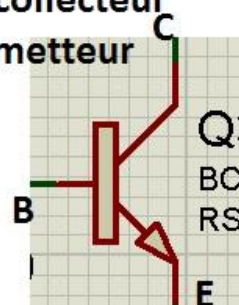
le courant est sortant de l'émetteur

type PNP

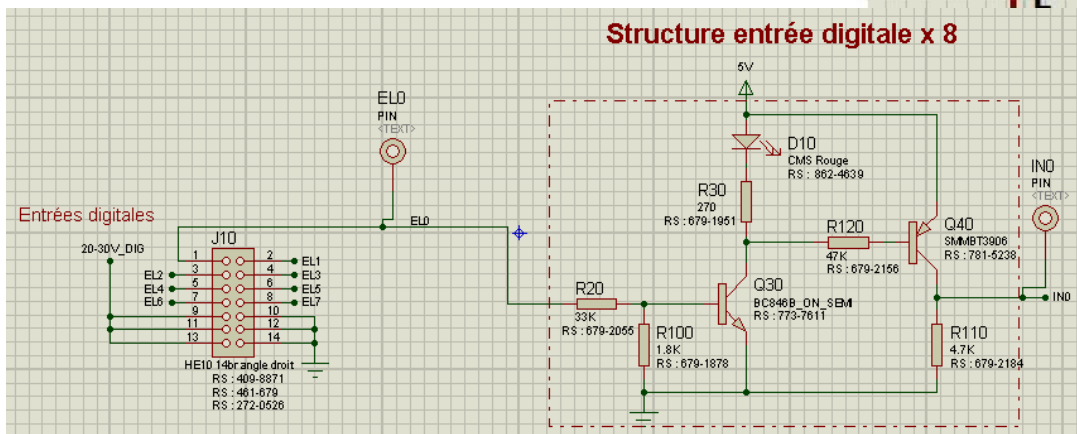


le courant est entrant dans l'émetteur

B -> la base
C -> le collecteur
E -> l'émetteur



Étude des états des transistors de ma structure
d'entrées digitales

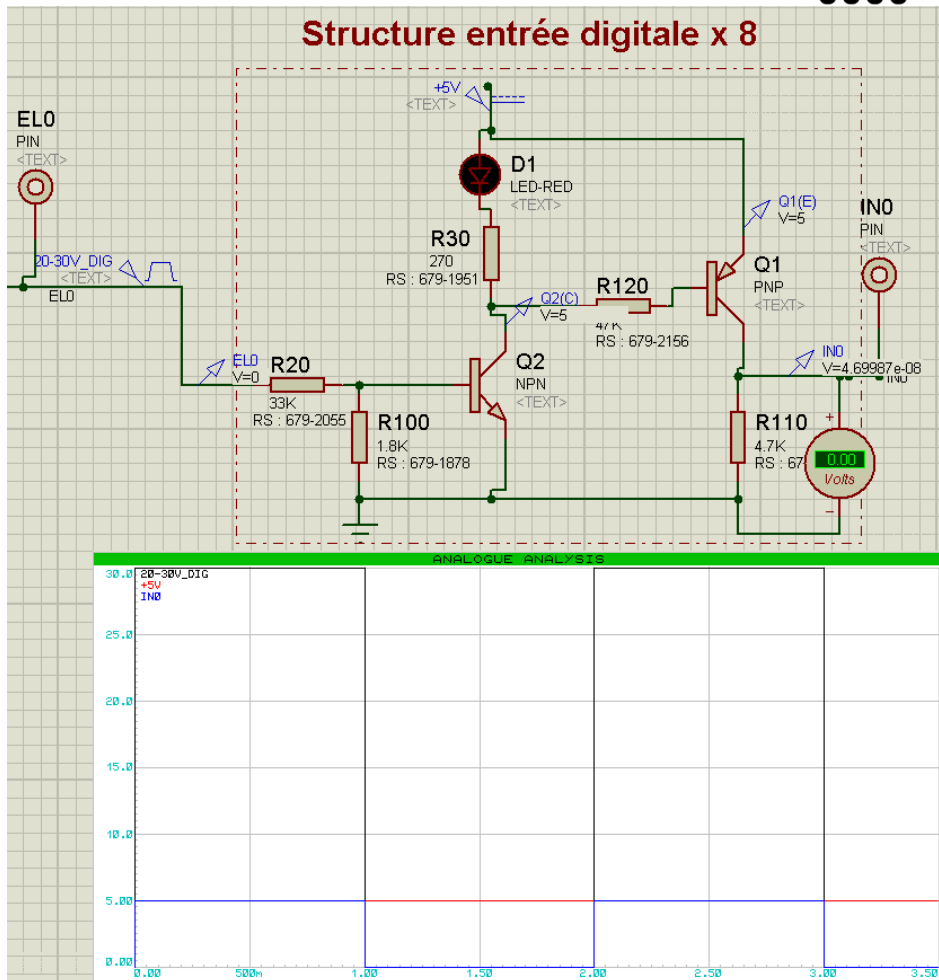


Q30 est un transistor de type NPN

Q40 est un transistor de type PNP

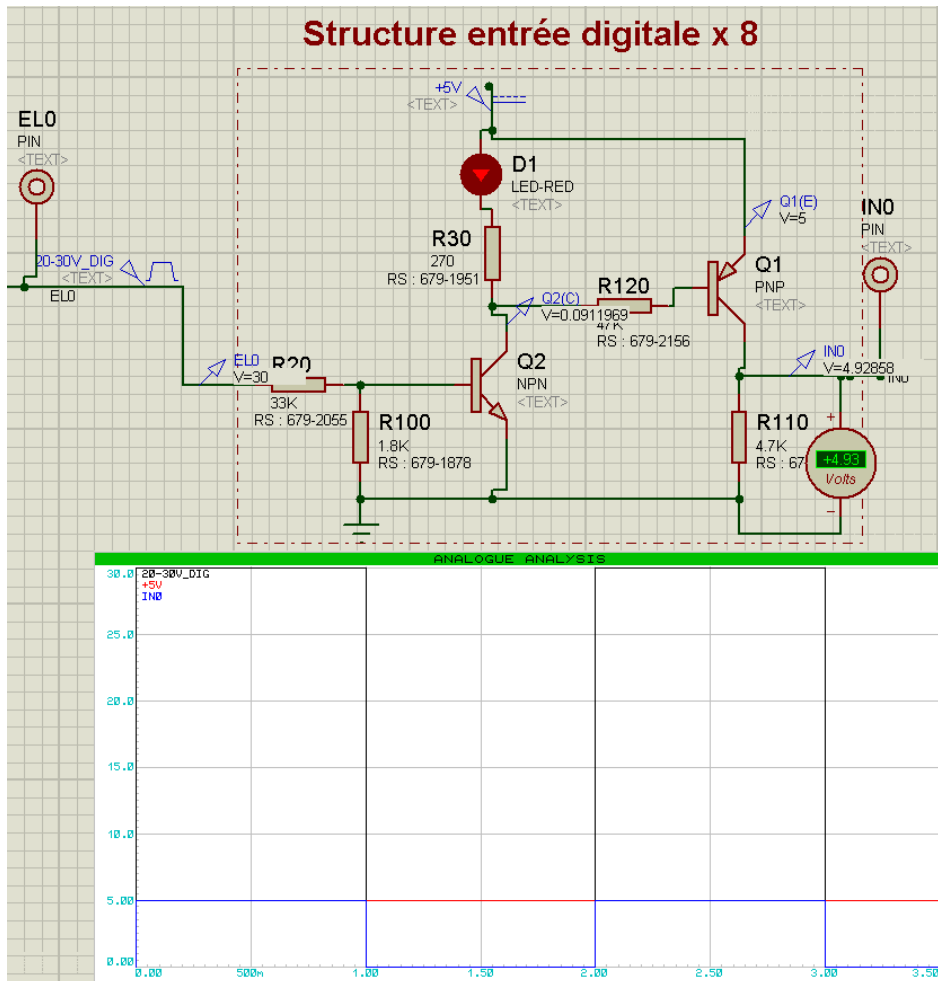
Lorsque la tension de EL0 = 20~30V Q30 est saturé et devient un interrupteur fermé, de ce fait la led D10 s'allume et Q40 devient aussi saturé et on obtient la tension de 5V à IN0.

6.14 Partie Physique



Lorsque la tension de $ELO = 0V$ $Q2$ est bloqué et devient un interrupteur ouvert, de ce fait la led $D2$ est éteinte même si $VDD = 5V$ et $Q1$ devient aussi bloqué et on obtient la tension de $0V$ à $IN0$.

6.14 Partie Physique



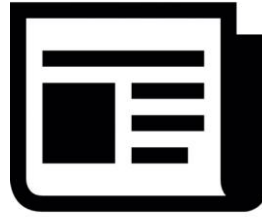
Lorsque la tension de ELO = 20~30V Q2 est saturé et devient un interrupteur fermé, de ce fait la led D2 s'allume et Q1 devient aussi saturé et on obtient la tension de 5V à IN0.

6.15 Journal de Bord

Journal de bord

API IO

FOURNIER ROBIN



18/01/18 – Exposé MR Defrance Explication Projet

19/01/18 – Isis et docs

24/01/18 – Isis et docs

25/01/18 – Montage sur breadboard structure entrée

26/01/18 – Montage sur breadboard structure entrée terminé + test

31/01/18 – Montage sur breadboard structure sortie logique terminé + test

01/02/18 – Correction montage structure entrée digitale + tableau logique structure sortie commande relai

15/02/18 – Câblage second level shifter et réunion projet

16/02/18 – Test 2nd level shifter sur raspberry, capture de trame SCL SDA sur LogicPort, début ARES Shield RPI

21/02/18 – Routage Shield RPI & Test Programme Macker

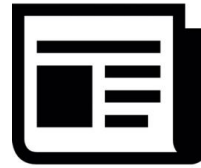
23/02/18 – Routage Double couche Shield RPI

14/03/18 – Dossier de Revue 1

15/03/18 – Dossier Revue 1

~54h depuis début du projet

6.15 Journal de Bord



21/03/18 – Revue de Projet 1

22/03/18 – Routage Shield Correction

28/03/18 – Début Soudure Shield RPI

29/03/18 – Fin Soudure Shield RPI, Modification Programme MCP23017

30/03/18 – Modification Dossier Revue

04/04/18 - Modification Shield RPI

05/04/18 – Validation & Test Shield RPI avec programme, nomenclature

06/04/18 – Schéma ISIS carte entière multiplication des structures

11/04/18 – Schéma de principe modification avec sous circuits E/S

12/04/18 – Partie physique & Routage Aménagement Structure entrée

13/04/18 -> 16/05/18 – Routage Carte Finale, Nomenclature, Revue avec entreprise API, Partie Physique Appliquée

18/05/18 – Dossier Revue 2

23/05/18 – Fin Routage Carte & Revue de Projet #2

24/05/18 – Dossier Revue, Nomenclature carte DI DO

Annexe Nomenclature

Composants Carte DI DO



QTÉ	PART-REFS	VALEUR	CODE
---	-----	-----	----
Resistances			

8	R4, R5, R7-R10, R13, R19	10k	
1	R6	330	
25	R18, R100-R107, R140-R147, R160-R167	1.8K	
8	R20-R27	33K	
8	R30-R37	270	
8	R110-R117	4.7K	
8	R120-R127	47K	
8	R150-R157	6.8K	
8	R170-R177	18K	
Capacitors			

1	C1	4.7uF	
2	C2, C4	100nF	
1	C3	10uF	
Circuits Intégrés			

1	U1	MCP23017	
Transistors			

2	Q1, Q2	2N7002	
16	Q30-Q37, Q50-Q57	BC846B_ON_SEMI	
16	Q40-Q47, Q60-Q67	SMMBT3906	
Diodes			

1	D2	1N4001	
1	D4	LED CMS Jaune	
1	D5	LED CMS Verte	
16	D10-D17, D30-D37	CMS Rouge	
Autres			

1	DC1	TMR2411	
36	EL0-EL7, GND, IN0-IN7, OUT0-OUT7, PT_5V, SCL_A, SDA_A, SL0-SL7	PIN	
1	FU1	Polyswitch	
1	J0	CONN-SIL10	
1	J1	CONN-DIL14_B	
1	J2	CONN-DIL20_RPI	
1	J3	CONN-DIL6	
1	J5	Bornier à vis	
1	J6	CONN-SIL2	
1	L1	12uH	



Carte DI DO Fournier Robin

	Nom	Date	Folio
Crée :	???	??/??/2012	1/3
Mis à jour :	???	??/??/2012	

Carte DI DO Fournier Robin										Client :		
Repère	Désignation du matériel	Valeur (Référence)	Tol.±	Equivalent	Fabricant @ Fournisseur	Boitier	Recommandation câblage	Qté	Condition	ref rapide "BS" (FA)	Prix U HT	Prix T HT
IN7/OUT0-OUT7/PIN								36				0,000
FU1	Poly-switch fusible réarmable	500mA – 0.25A			Littelfuse		(Ø 0.8mm)	1		517-6607	0.333	0.333
J0	CONN-SIL10						(Ø 0.8mm)	1				0,000
J1	CONN-DIL14_B						(Ø 0.8mm)	1				0,000
J2	CONN-DIL20_RPI											0,000
J3	CONN-DIL6											0,000
J5	Bornier à vis						(Ø 2 mm)	1				0,000
J6	CONN-SIL2						(Ø 2 mm)	1				
L1	Inductance de choc à mode commun				TRACPOWER			1		745-1200	2.200	2.200



Annexe Composants Shield RPI



Nomenclature API IO FOURNIER Robin						
Shield RPI						
Repère	Désignation	Valeur	Fabricant / Fournisseur	Quantité	Prix	
C1	Condensateur	1µF	RS	1	0,446	
J1	Embase femelle		RS	1	4,78	
J2	Bornier		RS	1	1,242	
R1	Résistance CMS	330ohm	RS	1	0,014	
FU1	Polyswitch	400mA – 0.2A	RS	1	0,358	
D1	LED CMS	Jaune	RS	1	0,137	
PIN	PIN		RS	3		