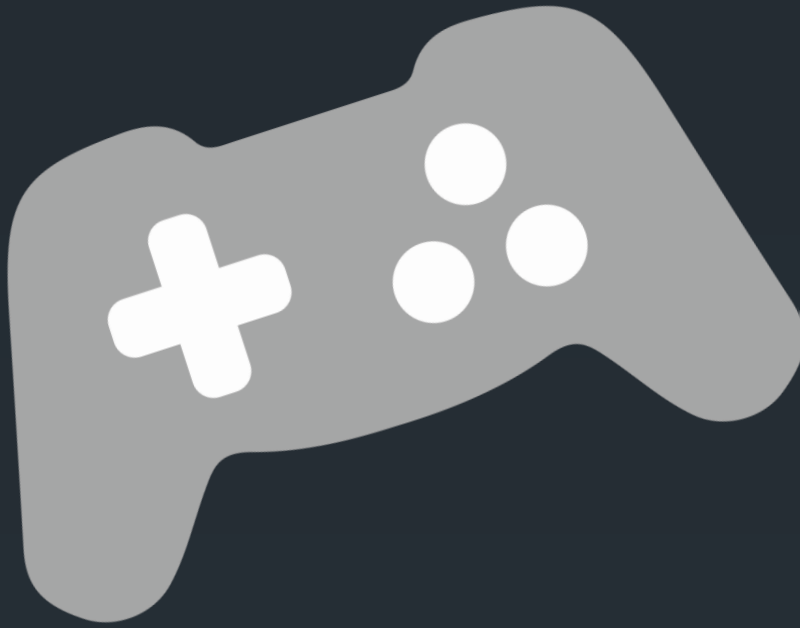




Manette Modulable

Groupe de projet n°35 : Manette de jeux vidéo pour personne en situation d'handicap.

Transley Gracias et Camille Saâd

Informatique Microélectronique Automatique

Polytech Lille _ 4^{ème} Année

Janvier – Mai 2018

Encadrants :

Alexandre Boé, Xavier Redon et
Thomas Vantroys

Remerciements

Nous tenons à remercier dans un premier temps nos tuteurs de projet, Messieurs Alexandre Boé, Xavier Redon et Thomas Vantroys pour nous avoir proposé et permis de faire ce projet lié à la santé. Nous les remercions également pour les conseils et le partage de connaissances qu'ils nous ont transmis au cours de cette 4^{ème} année.

Dans un second temps, nous souhaitons remercier Monsieur Thierry Flaman pour le temps et le partage de connaissances techniques, ses conseils, qu'il nous a accordé dans le cadre des réalisations circuits imprimés.

Enfin, nous remercions Le centre de rééducation Marc Sautelet pour avoir émis l'idée de ce projet l'an dernier à la demande d'un jeune patient hémiparétique.

Contenu

Remerciements	1
Table des illustrations	3
Introduction	4
1. Analyse du Projet : Recherches et développement.....	5
1.1. Un projet applicable pour le centre Marc-Sautelet	5
1.2. Cahier des charges	5
1.3. Fonctionnement	6
2. Réalisation des modules de la manette	7
2.1. Réalisation électronique : manette modulable	7
2.2. Communication entre composants : récupération des états	8
2.3. Partie conception	9
2.4. Module NRF52810 : un futur possible entre les modules	9
3. Serveur interface : Récupération des données en SPI	11
3.1. Communication SPI : principe de fonctionnement	11
3.2. SPI chez l'Atmega328P-PU (bouton.ino et joystick.ino).....	12
3.3. SPI chez la Raspberry Pi Zero W (raspi2devices.c)	12
3.4. SPI avec 2 modules : la différenciation des octets reçus	13
4. Une interface de gestion accessible.....	15
4.1. Configuration de l'application Android	15
4.2. Interfaces faciles d'utilisation	16
4.3. Interface facile d'utilisation	16
5. Connexion avec la console	17
5.1. Programmation de la Raspberry Pi Zéro W	17
5.2. Description des trames USB	17
5.3. Description des trames Bluetooth.....	20
Conclusion.....	21
Annexes.....	22
Document technique Raspberry Pi Zero W	22
Document technique Pins SPI Atmega328P-PU	22
Schéma de branchement prototype SPI sur nos modules.....	23
Schéma nouvelles valeurs du joystick.....	23

Table des illustrations

Figure 1 Bêta Corne du système	5
Figure 2 Prototype filaire avec deux modules : un bouton et un joystick	6
Figure 3 circuit électronique Atmega328p-pu	7
Figure 4 PCB bouton	8
Figure 5 Carte circuit électronique Joystick	8
Figure 7 Exemple réalisation de boîte module	9
Figure 6 capuchon gâchette	9
Figure 8 Circuit électronique NRF52810.....	10
Figure 9 Exemple code configuration application.....	15
Figure 10 Configuration Raspberry pour l'application	16
Figure 11 Présentation application Android	16
Figure 12 Description trames USB	17
Figure 13 Script clavier sortie OTG.....	18
Figure 14 Report de sortie pour envoyer les données des capteurs au pc.....	19
Figure 15 Trame Ps4 Bluetooth.....	20

Introduction

Dans un contexte mondial où les nouvelles technologies sont omniprésentes, les jeux vidéo sont un loisir pour tout âge et dans tout milieu social. Cependant, les manettes ne sont pas adaptées pour tous. En effet, les personnes à mobilité réduite, et plus particulièrement celles ayant un handicap touchant les membres supérieurs du corps, ont du mal à s'adapter et à jouer avec les manettes actuelles. Les quatre plateformes des jeux vidéo en concurrence sur le marché ne proposent que très rarement des alternatives concrètes à la manette classique. Pour nuancer, on peut trouver la Wii une manette utilisable à une main mais cette console ne donne pas accès à tous les types de jeux sur le marché. Playstation et Xbox proposent néanmoins des accessoires permettant de jouer avec ou sans les mains (Move, Kinect). Mais cela reste uniquement pour des jeux bien ciblés. On peut alors se demander quel système pourrait être créé pour adapter ces manettes aux handicaps moteurs.

Dans le cadre de notre quatrième année en formation IMA¹ à l'école d'ingénieurs Polytech Lille, nous nous sommes penchés sur la question. Suite à une demande émise l'an dernier par le centre de rééducation Marc-Sautelet, Villeneuve d'Ascq, et l'un de ses patients, un adolescent hémiplégique², nous avons recherché à concevoir une manette PS4 adaptée au handicap de l'adolescent.

Nous avons choisi de réaliser une manette modulable. Le but est donc d'exploser la manette en plusieurs parties appelées « modules ». L'utilisateur pourra choisir sa propre configuration. Cette manette peut dans un premier temps être filaire puis être remplacée par un prototype Bluetooth. Premièrement, pour établir le cahier des charges techniques, nous avons recherché les produits et projets disponibles. Nous avons étudiés les possibilités pour répondre au cahier des charges fonctionnel. Dans un deuxième temps, notre travail s'est procédé en plusieurs étapes afin d'établir le chemin entre la console PS4 et notre Manodulable³, la manette modulable. Notre projet été articulé en plusieurs parties. En effet, nous avons pris le temps au début du projet pour faire les recherches pouvant orienter notre développement. Puis, nous avons commencé à réaliser les prototypes. Nous avons réalisé des cartes électroniques qui récupèrent les valeurs des différents boutons et les renvoient vers la Raspberry Pi Zéro. Nous avons programmé une application Android pour initialiser les choix de l'utilisateur. Pour finir, nous avons programmé la Raspberry Pi pour qu'elle envoie à la console les valeurs des boutons dans le bon format.

¹ IMA : formation Informatique Microélectronique Automatique

² Hémiplégique : handicap caractérisé par une paralysie d'une ou plusieurs parties du corps et d'un seul côté.

³ Manodulable : Anagramme des mots Manette et Modulable (Marque non déposée)

1. Analyse du Projet : Recherches et développement

1.1. Un projet applicable pour le centre Marc-Sautelet

Le Centre Marc Sautelet est un établissement de soins de suite et de réadaptation pour enfants et adolescents de la métropole lilloise. L'établissement est spécialisé dans la prise en charge des conséquences fonctionnelles des affections telles que : orthopédiques, neurologiques, métaboliques, affections des brûlés.

Notre projet pourrait être utile pour un centre tel que Marc Sautelet. En effet, les jeux vidéo font partis de la vie quotidienne des enfants et nombreux sont les jeunes patients ne pouvant profiter de manière optimale de ce loisir C'est le cas d'un adolescent hémiparétique. Ayant une paralysie totale du côté droit de son corps, il ne peut utiliser sa main pour jouer. Une manette modulable permettrait alors de positionner les boutons à différents endroits accessibles du côté gauche (pied, épaule, main...). Il s'agit également de choisir des boutons plus ou moins grands selon le besoin. La manette n'est donc plus centralisée en une seule localisation ou ni réduite à une seule taille.

1.2. Cahier des charges

Chef de projet : Alexandre Boé
 Responsables : Xavier Redon, Thomas Vantroys
 Partenariat : Marc Sautelet (non établi)
 Elèves-ingénieurs : Camille Saâd, Transley Gracias

Sujet : Manette modulable pour personne à déficience moteur compatible PlayStation 4

Objectifs principal (1er POC) :

- Faire transmettre des informations issues des capteurs (ex:boutons poussoires) via des atméga328p-pu jusqu'à la Raspberry Pi (Rpi 3 ou Rpi Zero W) en SPI⁴.
- Envoyer les données reçues vers la PS4 en USB
- Réaliser une application faisant le lien entre la configuration de la manette et la correspondance possible avec la console

Objectifs secondaire (2e POC):

- Modifier les modules avec un microcontrôleur et une puce Bluetooth pour permettre une meilleure flexibilité, soit une meilleure ergonomie et une connexion sans fil).
- Connecter la Raspberry Pi 3 en Bluetooth à la PS4

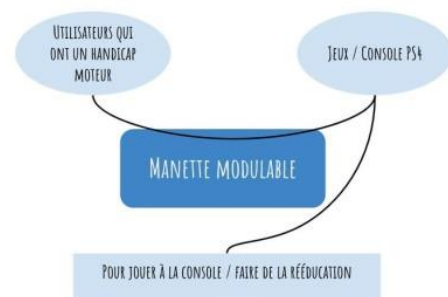


Figure 1 Bêta Corne du système

⁴ SPI : Serial Peripheral Interface, bus de données série

1.3. Fonctionnement

Dans notre projet, nous nous concentrerons sur la réalisation de 3 modules pour un prototype filaire : 1 bouton, 1 joystick et 1 gâchette.

Ainsi notre première preuve de concept sera composée de ces 3 capteurs (bouton, joystick, gâchette) accompagnés de leurs cartes Atmega358p-pu. En fonction de leur type, analogique ou numérique, les valeurs sont récupérées par les microcontrôleurs. Puis elles sont de nouveau récupérées via une liaison SPI (Serial Peripheral Interface) en mode 0⁵ par la Raspberry Pi Zero W. La raspberry est le maître et les Atmega328p-pu sont les esclaves. Lorsque l'utilisateur met en place la configuration physique de la manette, il doit passer par l'application pour indiquer quel type de bouton il souhaite utilisé. Il devra alors tout simplement créer un bouton et le déplacer dans l'emplacement voulu. Lorsque la configuration est validée, la Raspberry Pi peut enfin retransmettre l'ensemble des valeurs récupérées avec la bonne correspondance d'une vraie manette. Il nous faut alors traiter le bon format de ces valeurs.

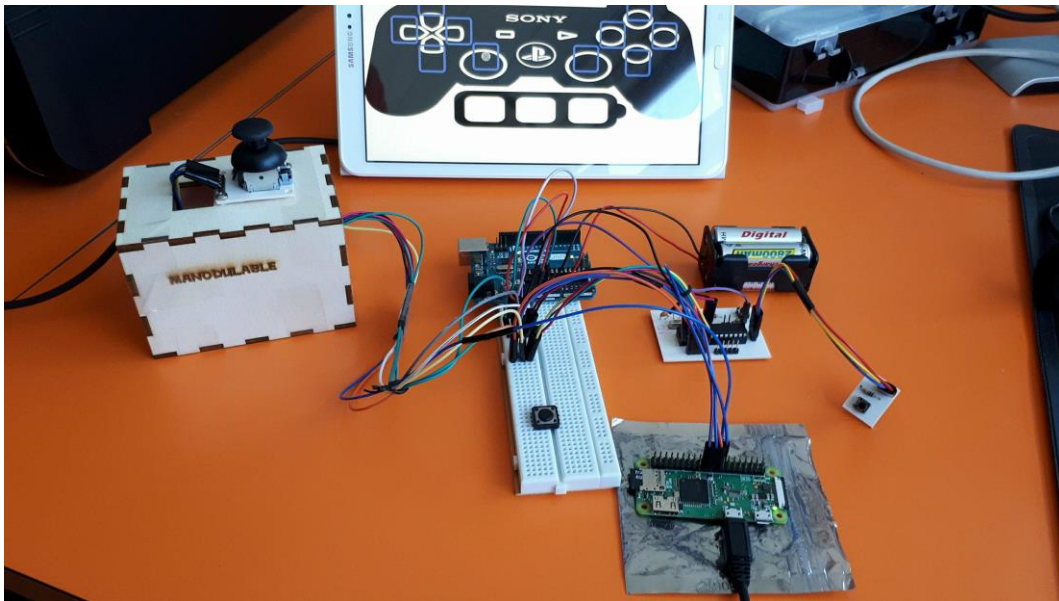


Figure 2 Prototype filaire avec deux modules : un bouton et un joystick

⁵ Mode 0 : En Spi il s'agit du mode par défaut. Bit poids fort CPOL (Clock Polarity) = 0 et le bit de poids faible CPHA (Clock Phase) = 0

2. Réalisation des modules de la manette

La modularité de notre manette se joue sur la partie électronique de notre projet en comportant plusieurs modules.

2.1. Réalisation électronique : manette modulable

Nous avons réalisé notre système de modules avec la composition suivante : 1 module = 1 carte Atmega328P-Pu + 1 carte capteur (bouton, gâchette ou joystick) + 1 alimentation. Ainsi, nous avons 3 types de circuits électroniques : Atmega328P-Pu, Push button et Joystick. Pour réaliser ces cartes, nous avons utilisé deux logiciels: Fritzing et Eagle.

1. Carte modules atmega328p-pu

Cette carte est le « centre d'intelligence » du module, elle permet de récupérer les états du capteur et de les retransmettre. La carte du microcontrôleur Atmega328P-Pu en standalone⁶, est accompagnée de connecteurs mâles pour les branchements nécessaires. Ces branchements concernent l'alimentation, le bouton ou le joystick/gâchette, la communication SPI, la communication série TxRx et reset.

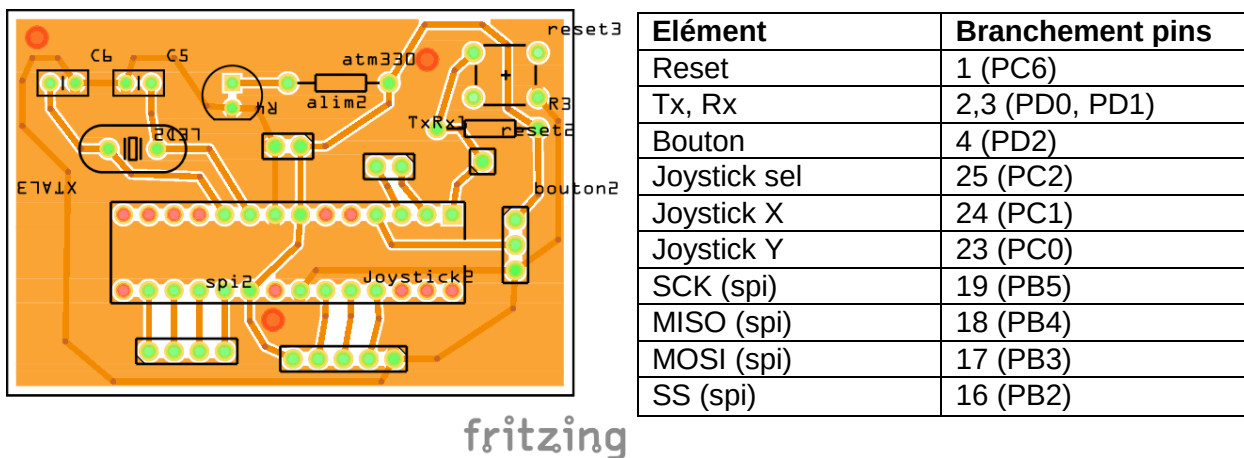


Figure 3 circuit électronique Atmega328p-pu

L'avantage de cette carte se trouve dans la possibilité de retirer l'Atmega328p-pu en cas de dysfonctionnement de celui-ci. De plus, nous sommes supposé ne plus avoir besoin de reprogrammer le microcontrôleur, cela est supposé définitif pour chaque carte. Une carte Atmega328p-pu pour un bouton poussoir restera donc une carte utilisable uniquement avec des boutons de ce type.

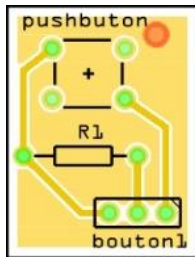
Cependant, cette carte est conçue de manière minimaliste avec des composants traversant. Elle aurait pu également être réalisée avec des composants CMS⁷ et accompagnée d'un

⁶Atmega Standalone : Atmega sans arduino, il peut être monté sur breadboard ou dans un circuit électronique

⁷ CMS : composant monté en surface (eq. SMD surface mounted device), nécessite de braser le composant à la surface d'une carte plutôt que passer par des broches à travers celle-ci.

microcontrôleur Atmega16u2 afin de programmer la carte sans passer par une Arduino UNO. Egalement dans les avantages d'une telle réalisation, nous pouvons citer le gain en surface et la réduction du nombre de câbles.

2. Carte élément bouton



fritzing

Figure 4 PCB bouton

Pour le bouton, nous avons réalisé une carte très simple comportant la résistance, un bouton poussoir et les broches pour connecter l'élément à la carte voulue.

3. Carte élément joystick/gâchette

Tout comme la carte du bouton, ce circuit électronique contient uniquement le joystick et la connexion à celui-ci. Il était nécessaire de réaliser ces PCB⁸ car les joysticks de notre matériel ne comportaient pas de Breakout Board⁹. Lors de cette réalisation, nous avons alors rencontré quelques difficultés. En effet, nous avons trouvé les schémas d'implémentation de notre joystick et deux possibilités s'offraient alors à nous : refaire au millimètre près les pins nous-même d'après les dimensions non normalisées données ou récupérer le fichier PCB .brd du constructeur et y modifier le routage selon nos besoins.

Nous avons commencé par la première possibilité mais nous nous sommes rapidement heurtés à des problèmes de précisions. La deuxième solution était donc plus adéquate malgré les difficultés rencontrées face à des paramétrages spécifiques déposés par le constructeur sur le fichier .brd. Pour résoudre, ces difficultés nous avons retiré le plan de masse et uniquement gravé les pistes qui nous intéressaient.

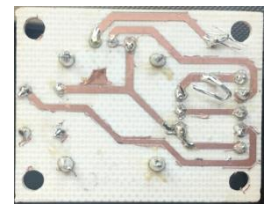


Figure 5 Carte circuit électronique Joystick

2.2. Communication entre composants : récupération des états

Avant toute programmation, il nous a fallu charger un amorceur sur chaque atmega328p-pu qu'on souhaite utiliser. Pour cela, il nous faut déjà programmer une Arduino UNO en tant que programmeur ISP. Cela afin qu'elle puisse programmer l'autre microprocesseur Atmega et y installer l'amorceur. Voici les étapes :

- Lancer l'IDE Arduino
- Ouvrir le programme à téléverser
- Dans «Outils» => «Type de carte» => sélectionner «ATmega328 on a breadboard 8 MHz internal clock»
- Dans «Outils» => «Programmeur» => sélectionner «Arduino as ISP»
- Dans «Croquis» => Téléverser avec un programmeur

⁸ PCB : de l'anglais « printed circuit board », soit circuit imprimé

⁹ Breakout board : circuit électronique permettant d'exposer sur des connecteurs standards les pattes d'une puce d'un format particulier.

Concernant le programme des microcontrôleurs permettant de lire les états des différents capteurs de module, nous l'avons réalisé sur Arduino IDE (fichiers .ino) afin réutiliser ces lignes de codes par la suite.

Cette récupération de l'information s'est faite en plusieurs étapes à l'aide de plusieurs cartes électroniques composées des microcontrôleurs Atmega328p-pu. Chaque carte modélise un « module » à laquelle on a branché un élément-manette : un bouton, une gâchette ou un joystick. Cette configuration s'est faite en gardant à l'esprit la modularité avec une simplification du câblage pouvant réduire le confort et la mobilité du joueur.

2.3. Partie conception



Figure 6 Exemple réalisation de boîte module

Pour répondre à un problème d'esthétique et de confort, nous pouvons réaliser une boîte par module. Celles-ci pourront alors être aussi bien posées sur une table, au sol ou prises dans les mains. On garde le côté modulable. Pour aller plus loin, on pourrait travailler sur l'ergonomie de la manette avec des spécialistes d'un centre de rééducation pour mieux répondre aux besoins des patients. Ces boîtes ont été réalisées en bois à la découpeuse laser par l'intermédiaire du

logiciel Inkscape.

Nous avons également dû faire la différence entre les gâchettes et les joysticks. En effet, les manettes de jeux vidéo présentent des

boutons R et L, il s'agit de boutons sensibles à la pression appliquée, on les appelle des gâchettes et sont alors analogiques. Pour modéliser, ces boutons, nous avons choisi d'utiliser un joystick dont on bloquera un axe. Il a alors inclus le système suivant aux boîtes précédentes.

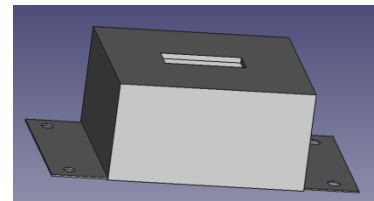


Figure 7 capuchon gâchette

2.4. Module NRF52810 : un futur possible entre les modules

Lors ce projet, nous avons essentiellement travaillé à la réalisation du prototype filaire. Cependant, ce projet portant sur de la recherche et du développement nous avons également cherché à préparer le second prototype de manette qui pourrait communiquer en connexion Bluetooth. Cette connexion bluetooth entre les modules et la raspberry pi permettrait plus de flexibilité. Il s'agit alors d'une solution idéale à la conception de notre manette.

Nous avons trouvé comme première solution la possibilité d'utiliser des composants NRF52810 de Nordic. Ceux-ci ont la capacité d'être un microcontrôleur et un module bluetooth en même temps.

La carte électronique nrf52810 a dû être réalisée entièrement à l'aide des documents techniques du constructeur. En fait, ayant uniquement un composant de surface, il nous faut le brasé sur une carte et y ajouter les condensateurs, quartz ou résistances nécessaires

pour avoir un module complet et utilisable. Pour faciliter la programmation de ce composant nous avons également ajouté des pins de branchement pour des connecteurs JTAG, ICSP ou encore SPI. Nous avons rencontré de nombreuses difficultés sur ce circuit électronique. Il nous a fallu la recommencer plusieurs fois sur Altium. Ces difficultés concernaient la dimension des composants et le routage de précision.

Nous avons réussi à développer et graver la carte une première fois, mais les pistes et l'empreinte prévues pour le nrf52810 étaient mal dimensionnées. En effet, il nous faut alors graver des pistes d'une taille largement inférieures à 6mil. Le 8mil est la taille minimale pour la production à Polytech, nous avons regardé pour envoyer les cartes à faire ailleurs mais le 6mil n'est pas beaucoup conseillé. Enfin, il n'était pas judicieux de continuer avec le composant microcontrôleur-bluetooth Nrf52810 qui prend du temps, donc la carte est difficilement réalisable et en plus d'être longue à programmer.

Nous avons décidé d'avancer du projet de laisser l'idée des modules avec les nrf52810 de côté. Il serait même plus intéressant de les remplacer par des modules sans fils des Xbee, des modules Rf ou directement des modules Bluetooth plus accessibles.

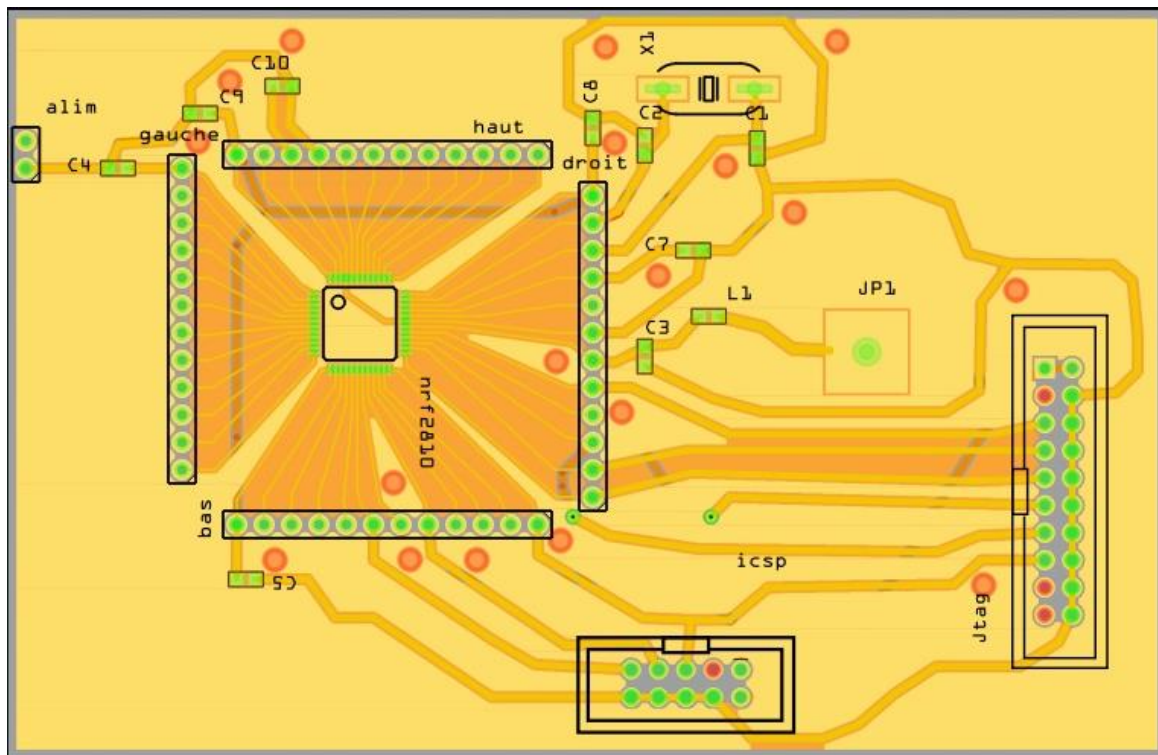


Figure 8 Circuit électronique NRF52810

3. Serveur interface : Récupération des données en SPI

La Raspberry pi joue un rôle de serveur interface entre la console, l'application et les capteurs. En effet, elle reçoit les états des capteurs des différents modules et traite celle-ci. Puis elle les envoie à l'application qui permet de configurer la manette selon les modules disponibles utilisés et les choix du joueur. Cela fait, la Raspberry pi doit communiquer les données à la console selon la correspondance aux boutons officiels.

Nous avons réalisé différents programmes travaillant ensemble, ils sont appliqués soit à un module soit à la raspberry pi.

3.1. Communication SPI : principe de fonctionnement

Il existe différentes possibilités pour transmettre des informations entre un atmega328p-pu et une Raspberry Pi. L'une d'elles est la réalisation d'une connexion SPI (Serial Peripheral Interface).

Une liaison SPI est un bus de données série synchrone qui échange en full-duplex¹⁰ entre un maître et un ou plusieurs esclaves. Le maître contrôle la communication, dans notre cas il s'agit de la Raspberry Pi. Les esclaves, soit nos modules Atmega328P-PU, peuvent coexister sur le même bus. La différenciation entre ceux-ci se fait alors par une sélection du destinataire avec une ligne appelée Slave Select (SS) ou Chip Select (CS). Par exemple, sur la Raspberry Pi, nous utilisons la librairie `/dev/spidev.h`, qui nous permet de choisir les périphériques avec `/dev/spidev0.0` (CS0) ou `/dev/spidev0.1` (CS1).

L'interface SPI utilise quatre signaux logiques :

- SCLK — Serial Clock, Horloge (généré par le maître)
- MOSI — Master Output, Slave Input (généré par le maître)
- MISO — Master Input, Slave Output (généré par l'esclave)
- SS — Slave Select, Actif à l'état bas (généré par le maître) (CS0 et CS1)

On peut retrouver en annexes les schematics de branchement de la Raspberry et de l'Atmega328P-P, également notre montage en « daisy chain ». Ce montage est plus adéquate pour un système multi-esclave où nous avons une seule clock pour tout synchroniser

La transmission se fait octet par octet. Le maître génère l'horloge et sélectionne l'esclave avec qui il veut communiquer par l'utilisation du signal SS et l'esclave répond e aux requêtes du maître. A chaque coup d'horloge le maître et l'esclave s'échangent un bit. Après huit coups d'horloges le maître a transmis un octet à l'esclave et vice versa. La vitesse de l'horloge est réglée selon des caractéristiques propres de nos périphériques (dans notre cas 400kHz). C'est sous ce principe de fonctionnement que notre RPi et nos modules communiquent.

¹⁰ Full-duplex : en double sens, maître vers l'esclave et esclave vers le maître.

3.2. SPI chez l'Atmega328P-PU (bouton.ino et joystick.ino)

Le principe étant le suivant : La Rpi envoie une requête à l'Arduino, il s'agit d'un tableau de 5 octets : `tx[]={0x0A ; 0x00 ; 0x00 ; 0x00 ; 0x00}`. L'arduino, une fois qu'elle reçoit 0x0A va vérifier régulièrement les états des modules. Puis, elle stocke ces valeurs dans un tableau qu'elle renverra à la RPi, octet par octet. On doit donc faire un programme SPI par type de module. En effet, selon le capteur bouton, joystick ou gâchette, le programme ne lira pas les mêmes états.

Dans le programme sous arduino IDE, qu'importe la lecture des états, une partie ne change pas pour les Atmega, il s'agit de la partie : l'envoi du tableau de données. Pour cela, il faut dans un premier temps configurer l'envoi avec l'atmega328p en esclave. On utilise la librairie Arduino SPI.h. On a alors directement les fonctions nécessaires pour intervenir sur les registres souhaités : SPCR (spi control register), SPSR(spi status) et SPDR (spi Data register). SPDR est important, c'est lui que nous devons lire ou écrire.

Le setup de notre programme se trouve en annexe.

On utilise les interruptions afin de bien lire la requête et renvoyer uniquement quand celle-ci a été reçue avec `ISR(SPI_STC_vect)`.

C'est dans la boucle que l'on va récupérer l'état du capteur, que l'on stocke dans un tableau que l'on envoie indice par indice à l'aide de la fonction `SPI.transfer()` ; et en écrivant également le même octet envoyé dans SPDR.

3.3. SPI chez la Raspberry Pi Zero W (raspi2devices.c)

Premièrement pour programmer en C, il nous faut choisir les bonnes libraires :

```
#include <stdint.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <getopt.h>
#include <fcntl.h>
#include <sys/ioctl.h> //pour la communication
#include <linux/types.h>
#include <linux/spi/spidev.h> //pour SPI
```

Le code pour la raspberry pi est composé en plusieurs fonctions permettant l'échange SPI en 4 étapes : l'initialisation de la communication, l'envoi de la requête (tableau : `tx[]={0x0A, 0x00,0x00,0x00}`) la réception des données en retour et le stockage de celles-ci.

Voici la liste des fonctions utilisées :

```
static void pabort(const char *s) : impression erreur
```

uint8_t transfer(int fd): fonction réalisant la communication, elle retourne un tableau des octets reçus par spi (buffer rx).

static void print_usage(const char *prog) : impression des options

static void parse_opts(int argc, char *argv[]) : options possibles

void setup_ioctl(int fd) : configuration de la communication

void recuperation(int fd) : recuperation des données, repartition des données dans les tableaux r1 ou r2 selon le device.

int main(int argc, char *argv[]) : ouvrir le device avec fd=open(device,O_RDWR) et traiter les données en faisant appel aux fonctions précédentes, c'est cette fonction qui boucle pour changer de device puis affiche les résultats.

Les variables globales nécessaire au programme sont les suivantes :

```
static const char *device1 = "/dev/spidev0.0"; //module1 bouton par exemple
static const char *device2="/dev/spidev0.1"; //module2 joystick par exemple
static const char *device=NULL;           //device prenant la valeur de device1 ou device2 à tour de rôle
static uint8_t mode;                      //mode spi mis à 0 par défaut = raspberry en maître
static uint8_t bits = 8;                  //Nombre de bits par mot, soit un octet envoyé
static uint32_t speed =400000;           //vitesse de la Clock en SPI.
static uint16_t delay;
static uint8_t recu[5];                   //tableau de stockage globale, rempli par la fonction récupération
static uint8_t r1[5];                     //tableau de stockage pour le device1, rempli dans le main
static uint8_t r2[5];                     //tableau de stockage pour le device2, rempli dans le main
```

Nous avons procédé petit à petit à plusieurs tests, dans un premier temps juste l'envoi et le retour identique d'une chaîne de caractères, l'envoi d'un caractère et la réception d'un autre, l'échange d'entiers pour arriver aux tests de la récupération des états pour un module. Cela fait, on a procédé aux mêmes tests avec deux modules, soit deux devices.

La RPi reçoit alors un octet à la fois. C'est octet sont les états des capteurs avec un identifiant. On les lit et on les stocke dans des tableaux d'octets liés aux devices. On pourra lire ces valeurs en hexadécimal tout comme en décimal

3.4. SPI avec 2 modules : la différenciation des octets reçus

Lorsqu'on a plusieurs modules, on téléverse un programme de récupération d'états (Programme arduino) sur chaque Atmega328p-pu selon le capteur choisi. Pour la raspberry pi, celle-ci ouvrira l'accès aux devices à tour de rôle. (/dev/spidev1 puis /dev/spidev2). Lors de cette communication, deux problèmes majeurs ont été rencontrés : les différenciations des octets reçus (quelle valeur ? pour quel module ? pour quel état ?) et des correspondances octet-device. Dans un premier temps, nous souhaitons réutiliser les valeurs reçues on souhaite donc les stocker dans des tableaux d'octets.

Cependant à chacun de ces tableau, à chaque boucle du main, les valeurs changent d'indice dans le tableau. Par exemple, nous aurons pour le joystickr1={X Y sel 0} puis {X sel 0 Y} puis {X Y 0 sel} ..etc changeait de position. Ce changement de position, rend le traitement des données par la suite impossible. Nous avons pensé à envoyer un caractère 'J' ou 'B' pour faire la différence mais l'ajout de ce caractère rendait la communication encore plus compliquée.

Pour résoudre ces problèmes, nous avons choisi de modifier directement les octets des états depuis les Atmega328p-pu, soit avant l'envoi par le bus SPI. Nous perdons de l'information suite à un décalage de 1 ou 3 sur la droite mais nous avons directement un identifiant Joystick/sel, Joystick/X, Joystick/Y ou encore Bouton ou pas bouton dans les octets transmis et reçus. Voici les correspondances choisies :

Module état reçu	Octet transmis vers la raspberry
Joystick	0x1XXXXXXXX
Bouton	0x0XXXXXXXX (soit on ne change pas pour le bouton l'octet reçu)
Joystick bouton	0x11XXXXXXXX
Joystick X	0x101XXXXXX
Joystick Y	0x110XXXXXX

Nous pouvons alors par la suite directement traiter les valeurs, et voir quel état des modules a changé. Par ailleurs, comme nous perdons en précision (décalage de 1 ou 3 bits sur la droite), les valeurs des états du joystick changent. Un schéma des nouvelles valeurs des états se trouve en annexes.

4. Une interface de gestion accessible

Comme nous l'avions vu précédemment, le grand intérêt de la manette réside dans sa modularité. De ce fait, les boutons seront amenés à changer. En effet pour s'adapter à l'utilisateur, la manette pourra augmenter la taille d'un bouton mais aussi la sensibilité par exemple. L'application agit également à ce niveau. Effectivement, le téléphone et/ou la tablette Android permet de préparer le terrain à la Raspberry Pi pour transférer les informations à la console.

Ainsi, l'utilisateur devra choisir les boutons qu'il veut utiliser, l'application pourra savoir comment est configurée la manette avant même la connexion à la console. L'utilisateur enverra à la Raspberry Pi Zero W les informations sur les boutons nécessaires pour jouer. Par exemple, l'utilisateur pour mettre un joystick à la place des flèches directionnelles si cela s'adapte à son handicap. Autre exemple, une gâchette type joystick à la place des gâchettes classiques.

Il est prévu donc que les échanges d'informations se fassent par Bluetooth Socket pour recevoir les informations coté Raspberry et les envoyer coté Android.

4.1. Configuration de l'application Android

Coté Android, il est donc nécessaire de posséder un appareil ayant une communication Bluetooth. Pour la connexion en Bluetooth socket, les deux appareils doivent besoin de connaître leurs AdresseMac. Donc, l'application commence par chercher un l'appareil distant pour se connecter à celui-ci. Dans notre cas, la Raspberry Pi.

Dès que la connexion est établie, un thread permet de laisser l'application tournée avec en fond de tâche la connexion active.

Pour envoyer et recevoir les données, on a besoin de créer un nouveau thread. Puis, une dernière partie qui implémente l'envoi et la réception des données via ce therad.

```
public void write(byte[] buffer) {
    try {
        mmOutputStream.write(buffer);

        // mHandler is to show send message from device
        mHandler.obtainMessage(Constants.MESSAGE_WRITE, -1, -1, buffer);
        mHandler.sendToTarget();
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

Figure 9 Exemple code configuration application

Toutes ces parties sont implémentées sous forme de classes et méthodes dans l'application Java Android.

4.2. Interfaces faciles d'utilisation

Coté Raspberry, il faudra donc faire avoir une approche équivalente à celle-ci citée ci-dessus. Il faut tout d'abord, activer l'utilisation du Bluetooth sur la Raspberry Pi Zero W. Puis créer, la socket Bluetooth connectée avec la bonne adresse mac.

```
s = socket.socket(socket.AF_BLUETOOTH, socket.SOCK_STREAM, socket.BTPROTO_RFCOMM)
s.connect((serverMACAddress,port))
while 1:
    text = input()
    if text == "quit":
        break
    s.send(bytes(text, 'UTF-8'))
s.close()
```

Figure 10 Configuration Raspberry pour l'application

4.3. Interface facile d'utilisation

L'utilisateur pourra être : le patient, un parent/proche ou une assistante médicale. En effet, ce produit pourra être aussi une aide pour les personnes hospitalisées en créant un moyen de se divertir malgré le handicap ou dans un but rééducatif.

Pour aider et faciliter l'utilisation du produit, l'application est conçue de manière à être ergonomique et très facile d'utilisation. Tous les types de boutons devront être présents sur l'application avec des possibilités de mise à jour. Pour affecter un type de boutons à un emplacement correspondant à la vrai manette, il suffira de déplacer le bouton créé et de la placer à l'endroit souhaiter sur l'écran.

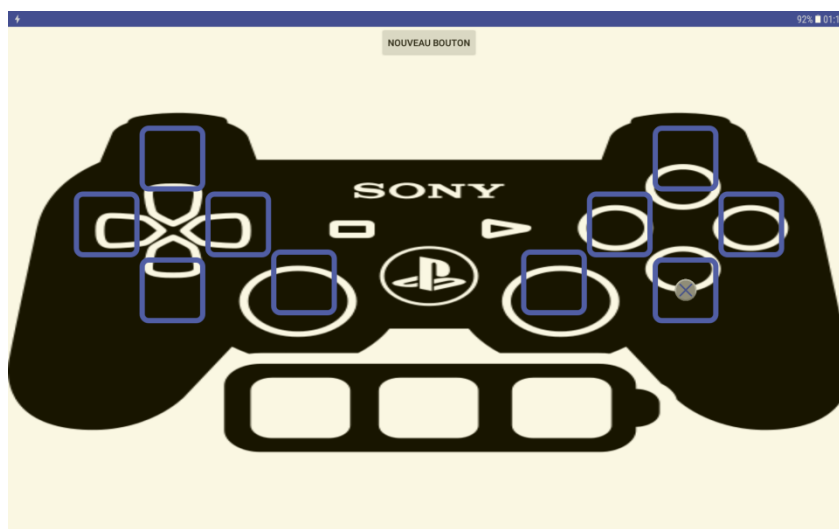


Figure 11 Présentation application Android

5. Connexion avec la console

La Raspberry Pi Zéro W permet de récupérer la configuration choisit par l'utilisateur. Mais, elle fait aussi office de serveur pour la console. En effet, la Raspberry Pi permet de récupérer les données reçues par la SPI. Ces données, il faut les passer dans le bon format avant de les envoyer à la console. Il nous faut traiter ces informations. Pour envoyer les données, la console PS4 nous permet deux modes de communication. Le premier via le port USB, le second via la connexion Bluetooth.

5.1. Programmation de la Raspberry Pi Zéro W

Tout d'abord, pour pouvoir programmer un périphérique USB sur le port OTG, il est nécessaire de faire des configurations préalables. Pour cela, on a besoin de ConfigFs qui est un système de fichiers virtuels. Il nous permet de créé depuis l'espace « utilisateur » des périphériques USB. Quelques réglages sont nécessaires :

- on doit rajouter `dtoverlay=dwc2` dans le fichier `boot/config.txt` qui permet de charger le driver `dwc2` pour la gestion de l'OTG qui se trouve dans le noyau
- on doit rajouter `dwc2` et `libcomposite` dans le fichier `data/etc/modules` pour initialiser `configFs`

Voici donc les premières étapes franchies en plus de la connexion Raspberry Pi Zéro.

5.2. Description des trames USB

Sachant que la console accepte les manettes via le port USB, on a trouvé des projets qui partagent du travail reverse engineering de la communication USB entre la manette et la console PS4. Ce travail nous donne la description de la manette avec les différentes valeurs comme `idVendors` ou `idProduct`, ou encore la vitesse de communication. On y trouve également la structure des « reports » et beaucoup d'informations nécessaires à une bonne communication entre les deux appareils.

```
0x12,      // bLength
0x01,      // bDescriptorType (Device)
0x00, 0x02, // bcdUSB 2.00
0x00,      // bDeviceClass (Use class information in the Interface Descriptors)
0x00,      // bDeviceSubClass
0x00,      // bDeviceProtocol
0x40,      // bMaxPacketSize0 64
0x4C, 0x05, // idVendor 0x054C
0xC4, 0x05, // idProduct 0x05C4
0x00, 0x01, // bcdDevice 1.00
0x01,      // iManufacturer (String Index)
0x02,      // iProduct (String Index)
0x00,      // iSerialNumber (String Index)
0x01,      // bNumConfigurations 1
```

Figure 12 Description trames USB

La spécification HID¹¹ est une classe de plus haut niveau que le protocole USB. Elle permet de créer simplement des périphériques USB. Pour nous familiariser avec la classe HID, nous avons commencé par implémenter un script qui crée un clavier en sortie du port OTG de la Raspberry Pi Zero W.

```
# Create gadget
mkdir /sys/kernel/config/usb_gadget/mykeyboard
cd /sys/kernel/config/usb_gadget/mykeyboard

# Add basic information
echo 0x0100 > bcdDevice # Version 1.0.0
echo 0x0200 > bcdUSB # USB 2.0
echo 0x00 > bDeviceClass
echo 0x00 > bDeviceProtocol
echo 0x00 > bDeviceSubClass
echo 0x08 > bMaxPacketSize0
echo 0x0104 > idProduct # Multifunction Composite Gadget
echo 0x1d6b > idVendor # Linux Foundation
```

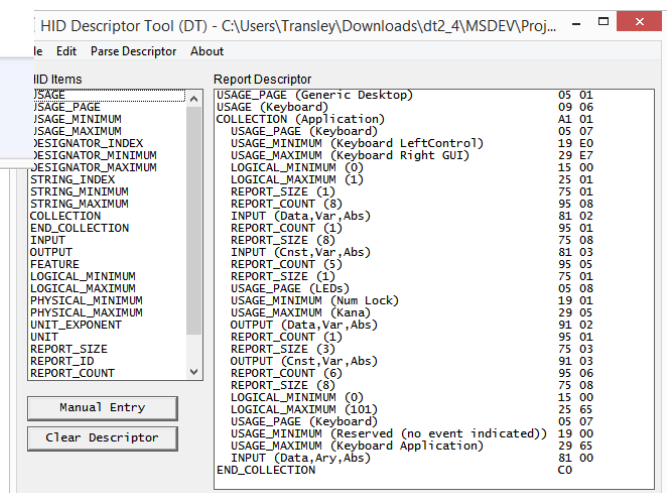
Figure 13 Script clavier sortie OTG

Grâce à ce script et l'utilisation de la commande `echo -ne`, on peut écrire sur un autre ordinateur via la communication USB. Nous avons utilisé un logiciel pour écrire le HID descriptor car il s'agit d'un codage particulier avec beaucoup de combinaisons possibles (HID Descriptor Tool).



On le voit sur cette capture d'écran l'ordinateur reconnaît une manette PS4 dans la version 4. Il s'agit bien de celle qu'on veut implémenter. Nous n'avons pas utilisé le HID descriptor de la manette PS4 (hormis `idVendor` et `idProduct`) car il était plus simple pour nous de tester avec un Joystick classique compatible avec Windows et Linux.

A partir de ce premier exemple nous avons donc commencé par implémenter le script d'un joystick pour PC avec les ID's d'une PS4.



```
root@zabeth16:~# lsusb
Bus 004 Device 003: ID 413c:2003 Dell Computer Corp. Keyboard
Bus 004 Device 017: ID 18a5:0302 Verbatim, Ltd Flash Drive
Bus 004 Device 016: ID 054c:05c4 Sony Corp. DualShock 4
Bus 004 Device 002: ID 8087:0024 Intel Corp. Integrated Rate Matching Hub
```

¹¹ HID

Figure 4: Example PID Set Effect Output Report

Byte	Bit							
	7	6	5	4	3	2	1	0
0	Report ID = 1							
1	ROM Flag	Effect Block Index						
2	Effect Type							
3	Duration							
5	Sample Period							
6	Gain							
7	Trigger Button							
8	Trigger Repeat Interval							
9	Pad						Y Axis Enable	X Axis Enable
10	X Axis Direction							
11	Y Axis Direction							
12 - 13	Type Specific Block Offset 1							
14 - 15	Type Specific Block Offset 2							

Pour finir sur la communication USB, après avoir récupéré les valeurs des boutons et initié le gadget USB, nous avons donc mis en place le transfert des données comme pour le clavier. Mais travaillant toujours dessus, nous avons pour l'instant eu très peu de résultat concluant mais très proche.

Pour passer sur la PS4 il faudrait compléter le HID descriptor et implanter celui de la PS4, de même pour les autres « report » qui permet initialisation de la manette. Il faudrait aussi prendre en compte le paramétrage initial lors d'une première connexion à la console. Pour l'envoi de « report », il se fait octet par octet avec une trame bien spécifique.

Data Format								
byte index	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
[0]	Report ID (always 0x01)							
[1]	Left Stick X (0 = left)							
[2]	Left Stick Y (0 = up)							
[3]	Right Stick X							
[4]	Right Stick Y							
[5]	TRI	CIR	X	SQR	D-PAD (hat format, 0x08 is released, 0=N, 1=NE, 2=E, 3=SE, 4=S, 5=SW, 6=W, 7=NW)			
[6]	R3	L3	OPT	SHARE	R2	L2	R1	L1
[7]	Counter (counts up by 1 per report)						T-PAD	PS
[8]	Left Trigger (0 = released, 0xFF = fully pressed)							
[9]	Right Trigger							
[10]	Unknown, seems to count downwards, non-random pattern							
[11]	Unknown, seems to count upwards by 3, but by 2 when [10] underflows							
[12]	Unknown yet, 0x03 or 0x04							
[13 - 14]	Possibly Gyro X (seems to be signed)							
[15 - 16]	Possibly Gyro Y							
[17 - 18]	Possibly Gyro Z							
[19 - 20]	Possibly Accel X							
[21 - 22]	Possibly Accel Y							
[23 - 24]	Possibly Accel Z							
[25 - 63]	TODO, work in progress							

Figure 14 Report de sortie pour envoyer les données des capteurs au pc

5.3. Description des trames Bluetooth

On avait commencé le projet en étudiant la possibilité de connecter la Raspberry Pi en Bluetooth. Nous avons été surpris lorsque nous avons trouvé des sites web exposer des travaux sur le reverse engineering de la manette PS4.

Nous avons donc commencé par essayer d'exploiter ces données. Malheureusement, nous avons rapidement été confrontés à un sérieux problème. En effet, le protocole est incomplet. Nous manquons également d'explications claires sur les détails de la trame. Les tableaux de valeurs hexadécimal comme si desous sont alors compliqué à décrire.

PS4 service attribute search response

```
0x07, 0x00, 0x01, 0x01, 0x53, 0x01, 0x50, 0x36, 0x01, 0x4d, 0x36, 0x00, 0x32, 0x09, 0x00, 0x00,
0x0a, 0x00, 0x01, 0x00, 0x05, 0x09, 0x00, 0x01, 0x35, 0x03, 0x19, 0x11, 0x0a, 0x09, 0x00, 0x04,
0x35, 0x10, 0x35, 0x06, 0x19, 0x01, 0x00, 0x09, 0x00, 0x19, 0x35, 0x06, 0x19, 0x00, 0x19, 0x09,
0x01, 0x02, 0x09, 0x00, 0x09, 0x35, 0x08, 0x35, 0x06, 0x19, 0x11, 0x0d, 0x09, 0x01, 0x02, 0x36,
0x00, 0x32, 0x09, 0x00, 0x00, 0x0a, 0x00, 0x01, 0x00, 0x06, 0x09, 0x00, 0x01, 0x35, 0x03, 0x19,
0x11, 0x0b, 0x09, 0x00, 0x04, 0x35, 0x10, 0x35, 0x06, 0x19, 0x01, 0x00, 0x09, 0x00, 0x19, 0x35,
0x06, 0x19, 0x00, 0x19, 0x09, 0x01, 0x02, 0x09, 0x00, 0x09, 0x35, 0x08, 0x35, 0x06, 0x19, 0x11,
0x0d, 0x09, 0x01, 0x02, 0x36, 0x00, 0x3b, 0x09, 0x00, 0x00, 0x0a, 0x00, 0x01, 0x00, 0x07, 0x09,
0x00, 0x01, 0x35, 0x06, 0x19, 0x11, 0x0e, 0x19, 0x11, 0x0f, 0x09, 0x00, 0x04, 0x35, 0x10, 0x35,
0x06, 0x19, 0x01, 0x00, 0x09, 0x00, 0x17, 0x35, 0x06, 0x19, 0x00, 0x17, 0x09, 0x01, 0x03, 0x09,
0x00, 0x09, 0x35, 0x08, 0x35, 0x06, 0x19, 0x11, 0x0e, 0x09, 0x01, 0x04, 0x09, 0x03, 0x11, 0x09,
0x00, 0x02, 0x36, 0x00, 0x4d, 0x09, 0x00, 0x00, 0x0a, 0x00, 0x01, 0x00, 0x08, 0x09, 0x00, 0x01,
0x35, 0x03, 0x19, 0x11, 0x0c, 0x09, 0x00, 0x04, 0x35, 0x10, 0x35, 0x06, 0x19, 0x01, 0x00, 0x09,
0x00, 0x17, 0x35, 0x06, 0x19, 0x00, 0x17, 0x09, 0x01, 0x03, 0x09, 0x00, 0x09, 0x35, 0x08, 0x35,
0x06, 0x19, 0x11, 0x0e, 0x09, 0x01, 0x04, 0x09, 0x00, 0x0d, 0x35, 0x10, 0x35, 0x06, 0x19, 0x01,
0x00, 0x09, 0x00, 0x1b, 0x35, 0x06, 0x19, 0x00, 0x17, 0x09, 0x01, 0x03, 0x09, 0x03, 0x11, 0x09,
0x00, 0x01, 0x36, 0x00, 0x52, 0x09, 0x00, 0x00, 0x0a, 0x00, 0x01, 0x00, 0x0a, 0x09, 0x00, 0x01,
0x35, 0x03, 0x19, 0x12, 0x00, 0x09, 0x00, 0x04, 0x35, 0x0d, 0x35, 0x06, 0x19, 0x01, 0x00, 0x09,
0x00, 0x01, 0x35, 0x03, 0x19, 0x00, 0x01, 0x09, 0x00, 0x09, 0x35, 0x08, 0x35, 0x06, 0x19, 0x12,
0x00, 0x09, 0x01, 0x03, 0x09, 0x02, 0x00, 0x09, 0x01, 0x03, 0x09, 0x02, 0x01, 0x09, 0x05, 0x4c,
0x09, 0x02, 0x02, 0x09, 0x08, 0x1f, 0x09, 0x02, 0x03, 0x09, 0x01, 0x00, 0x09, 0x02, 0x04, 0x28,
0x01, 0x09, 0x02, 0x05, 0x09, 0x00, 0x02, 0x00
```

Figure 15 Trame Ps4 Bluetooth

Conclusion

Ce projet de manette modulable s'est avéré plus chargé que nous le pensions. En effet, étant donné que le projet était très orienté recherche et développement, nous avons passé beaucoup de temps à chercher des solutions répondant au cahier des charges. Il nous a fallu également du temps afin de comprendre chaque nouveau langage ou méthode de réalisation que nous choisissons. Malgré tout, nous nous rendons compte que tous nos choix ne sont pas optimaux pour notre projet. Par exemple, d'un côté avoir la suite des trames Bluetooth et USB pour la PS4 et d'un autre côté remplacer les communications utilisées jusqu'ici par de meilleurs choix. La communication SPI pourrait être remplacée par du I2C.

Nous avons pu apprendre beaucoup des différentes parties de notre projet. Nous avons développé des connaissances et compétences abordées ou non dans notre formation IMA avec un projet concret. Nous avons surtout appris comment réellement travailler sur un gros projet.

Sur le point de vue personnel, nous nous sommes également découvert en tant que étudiant futurs ingénieurs. Nous avons appris plus sur notre manière de travailler en équipe ou seul.

Enfin, ce projet est un premier prototype d'un projet innovant et plusieurs améliorations seraient intéressantes à développées.

Schéma de branchement prototype SPI sur nos modules

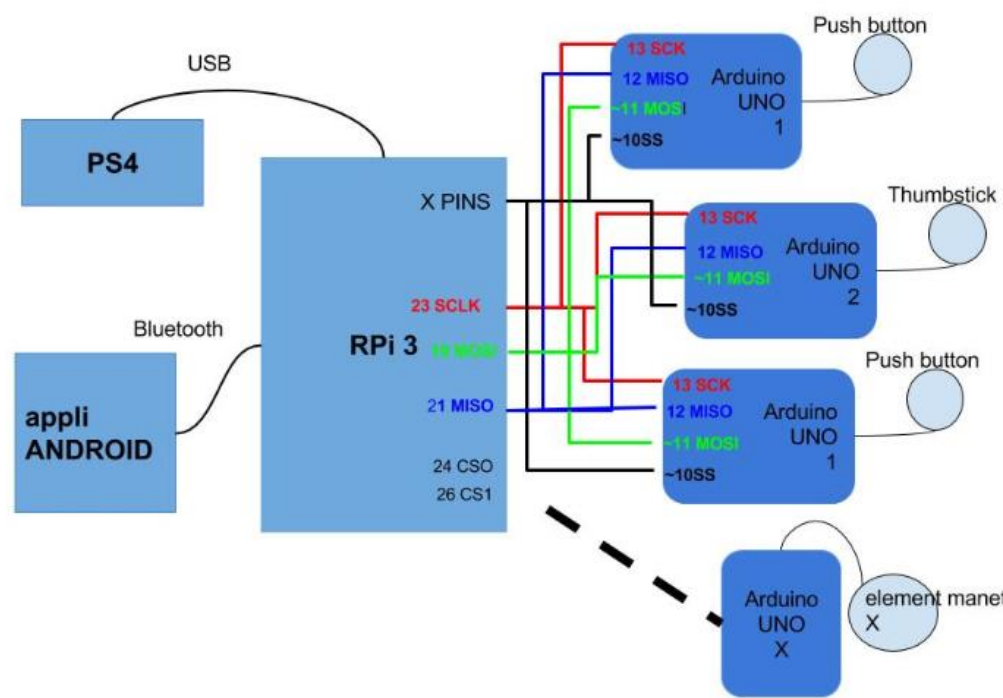


Schéma nouvelles valeurs du joystick

