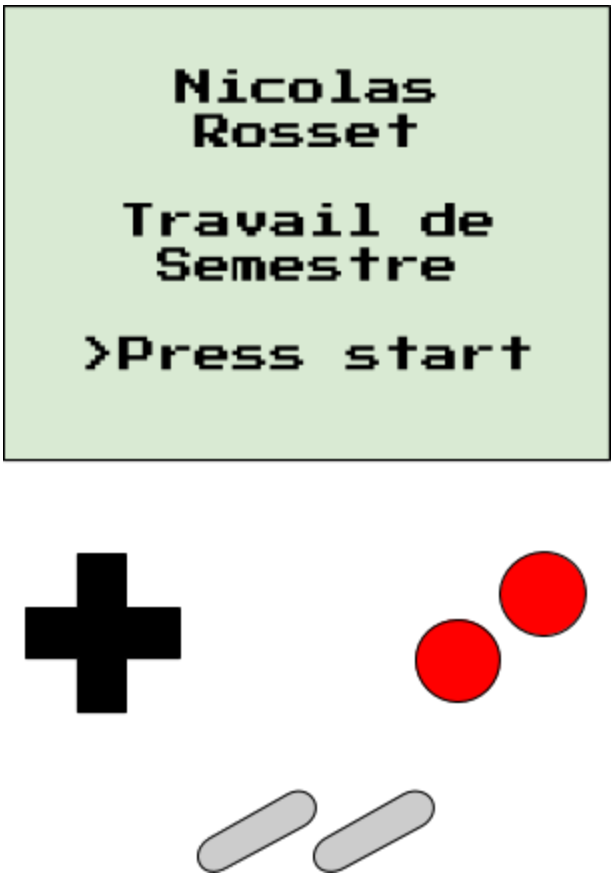
GameBoy

Sur Mylab2 et LPC4337



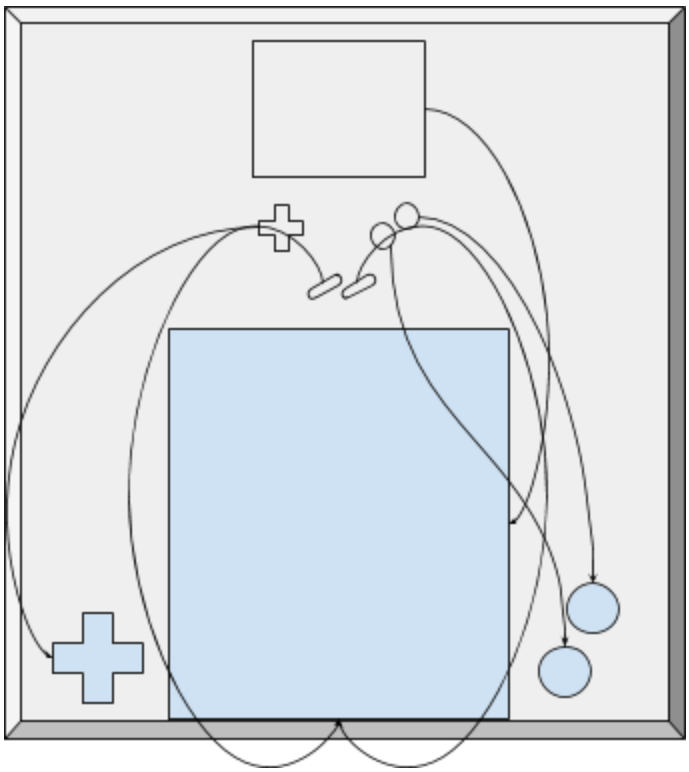
Introduction

Ce projet de semestre consistait à étudier la faisabilité d'un simulateur gameboy sur micro-contrôleur LPC4337 avec la carte d'extension faite à l'hepia : la MYLAB2 et d'éventuellement le tenter.

Ce dossier a pour but d'être utilisé afin de créer un simulateur Gameboy sur lpc4337. Il est donc une centralisation de sources, un recueil de recherches et aussi un cahier de laboratoire.

J'ai choisi ce projet car, comme beaucoup de gens j'ai été un utilisateur de la console gameboy à son époque et je suis très intéressé par l'utilisation de micro-contrôleur donc ce projet me semblait parfait.

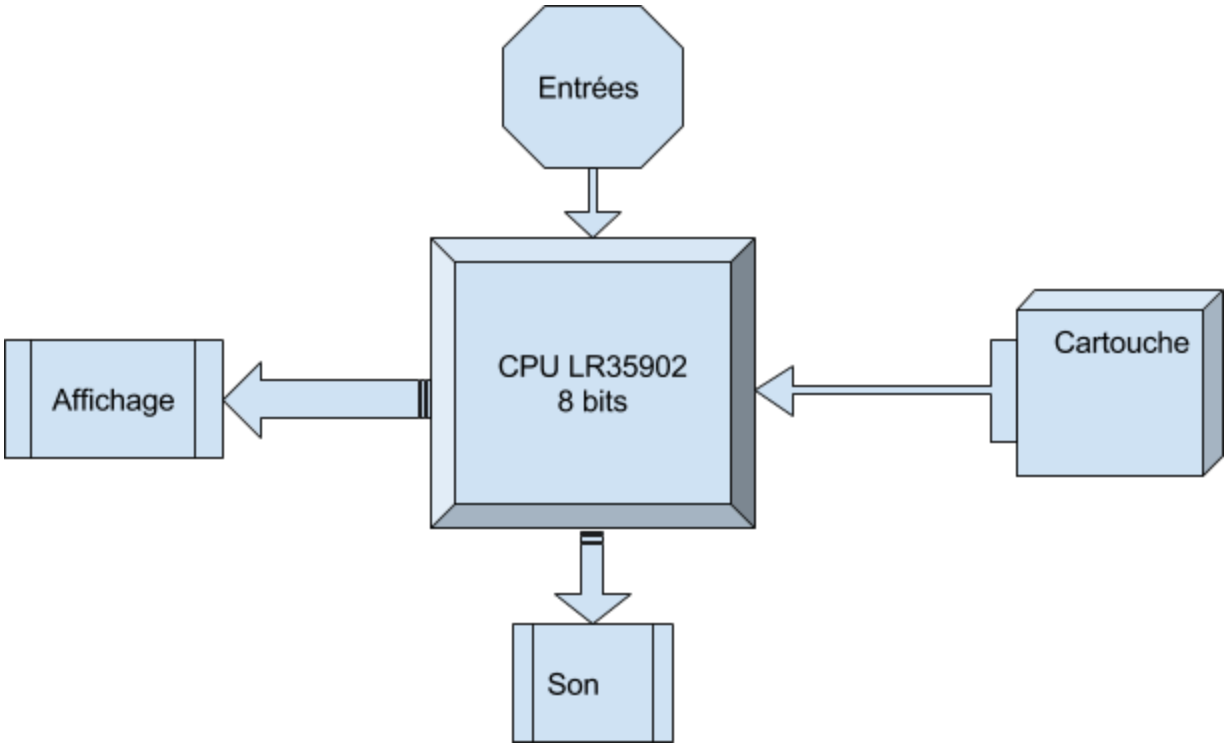
Ce rapport est séparé en deux parties, gameboy et micro-contrôleur. La première partie explique le fonctionnement de la gameboy, son architecture mémoire, ses entrées-sorties, etc… La seconde partie explique comment interfacer les parties de la gameboy avec le micro-contrôleur, l'émulation du processeur LR35902 et prend en compte la faisabilité de cette tâche.



Gameboy

Introduction

La nintendo gameboy (de son nom complet) est une console de jeu portable qui a marqué une génération. Ce qui nous intéresse aujourd'hui ce n'est pas la bibliothèque de jeux de cette console ni son potentiel de divertissement, mais son aspect technique. Nous allons séparer cette machine en plusieurs blocs et tous les étudier.



CPU

Le processeur de la gameboy est un CPU LR35902 8 bits similaire au Z80, c'est un processeur CISC.

Registres

Il utilise 6 registres 16 bits et 4 d'entre eux peuvent être accédés en 2 registres 8 bits chacun.

AF	A : work reg - F : Flags
BC	BC & B - C : work regs
DE	DE & D - E : work regs
HL	HL & H - L : work regs
SP	Stack pointer
PC	Program counter

Les flags se trouvent dans le registre F, il y en a 4:

- zf - flag zéro
- n - flag addition soustraction
    - Spécifie si la dernière instruction est une addition/soustraction
- h - flag demi-retenue (half carry)
    - Spécifie une retenue mais pour les 4 premiers bits
- cy - flag retenue

Processeur

Comme on fait face à une processeur CISC, les instructions peuvent prendre différents temps d'horloge pour s'exécuter.

Chaque instruction prend un nombre de cycle d'horloge divisible par 4, certaines ressources spécifient donc les temps d'instructions déjà divisées, par soucis de simplicité et fidélité, il n'y a pas de pré-division dans ce document.

Certaines instructions prennent un ou deux paramètres qui se trouvent à/aux adresse(s) suivant l'instruction.

Jeu d'instructions

Load
LD - Load depuis la mémoire

Incrémentation/décrémentation
INC - Incrémente un registre
DEC - Décrémente un registre

Pile mémoire
PUSH - Stocke le contenu d'un registre dans la pile et décrémente l'adresse de cette dernière de deux (SP)
POP - Stocke dans un registre la dernière valeur stockée dans la pile et incrémente l'adresse de cette dernière de deux (SP)

Arithmétique
ADD - Addition de deux registres
ADC - Addition de deux registres et du flag de retenue (cy)
SUB - Soustraction de deux registres
SBC - Soustraction de deux registres et du flag de retenue (cy)

Opération logique
AND - Opération bit à bit ET
XOR - Opération bit à bit OU exclusif
OR - Opération bit à bit OU
CPL - Inversion de chaque bit

Comparaison
CP - Soustraction sans résultat, mais met à jour les flags

Ajustement décimal
DAA - Ajuste le registre A pour avoir une représentation BCD

Rotation et décalage de bits
RL - Rotation sur la gauche
SL - Décalage sur la gauche
SWAP - Inverse l'ordre des bits
SR - Décalage sur la droite

Opération sur un bit
BIT - Teste un bit (met à jour des flags par rapport au bit)
SET - Met un bit à 1
RES - Met un bit à 0

Commandes du processeur
CCF - Inverse le bit de carry
SCF - Met le flag de carry à 1
NOP - Pas d'opération
HALT - Pause
STOP - Pause
DI - Désactive les interruptions
EI - Active les interruptions

Sauts d'instructions
JP - Saut absolu
JR - Saut relatif
CALL - Saute mais stocke le Program Counter (PC) dans la pile (un Branch and Link)
RET - Retour (retour à l'adresse stockée dans la pile (attention à l'ordre des instructions))
RST - Un call mais dans la table des resets

Mémoire

Matériel

Le processeur de la gameboy possède 2 sram 8kB, une sram pour la vidéo et une autre multi-usage.

Entre autre, elle peut aussi accéder à une mémoire ram interne et une mémoire rom interne à une cartouche de jeu

Les registres OAM sont réservés aux attributs des sprites.

Map

Début	Fin	Description
0000	3FFF	Rom de la cartouche
4000	7FFF	Rom de la cartouche échangeable par MBC
8000	9FFF	VRAM interne, SRAM dédié à l'affichage
A000	BFFF	RAM externe se trouvant dans la cartouche
C000	DFFF	RAM interne, SRAM multi-usage de la gameboy
FE00	FE9F	OAM, mémoire dédiée aux sprites
FF00	FF7F	Registres E/S
FF80	FFFE	HRAM
FFFF	FFFF	Registre interrupt enable

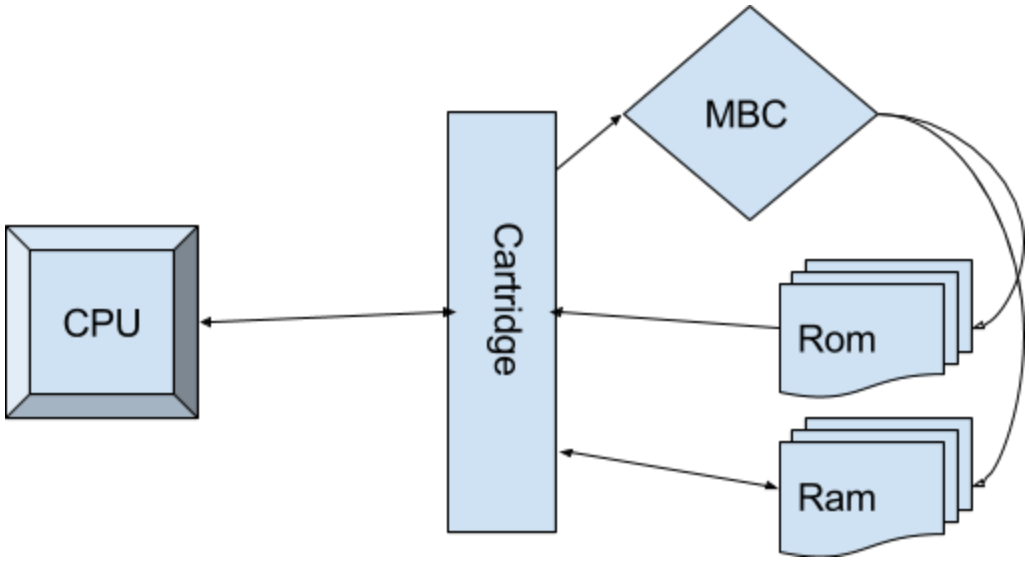
E000 - FDFF et FEA0 - FEFF ne sont pas utilisées

Cartouche (Cartridge)

Une cartouche gameboy est un périphérique externe standardisé pour gameboy, il s'agit d'un assemblage de plusieurs mémoires, ROM et RAM.

Le code d'un jeu entier se trouve tout naturellement dans la cartouche et la gameboy peut même y sauvegarder des données (pour garder une partie en mémoire par exemple).

Le processeur peut faire des loads en prenant en paramètre, une adresse sur 16 bits, ce qui limite considérablement la taille mémoire possible, pour y remédier, les cartouches de jeu implémentent un MBC, memory bank controller.
Un memory bank controller permet d'échanger des mémoires, on peut aussi parler de mémoire paginée. Ainsi même avec une plage d'adresse réduite on peut quand même accéder à une grande quantité de mémoire.



Gestion de la cartouche

Entrées (Inputs)

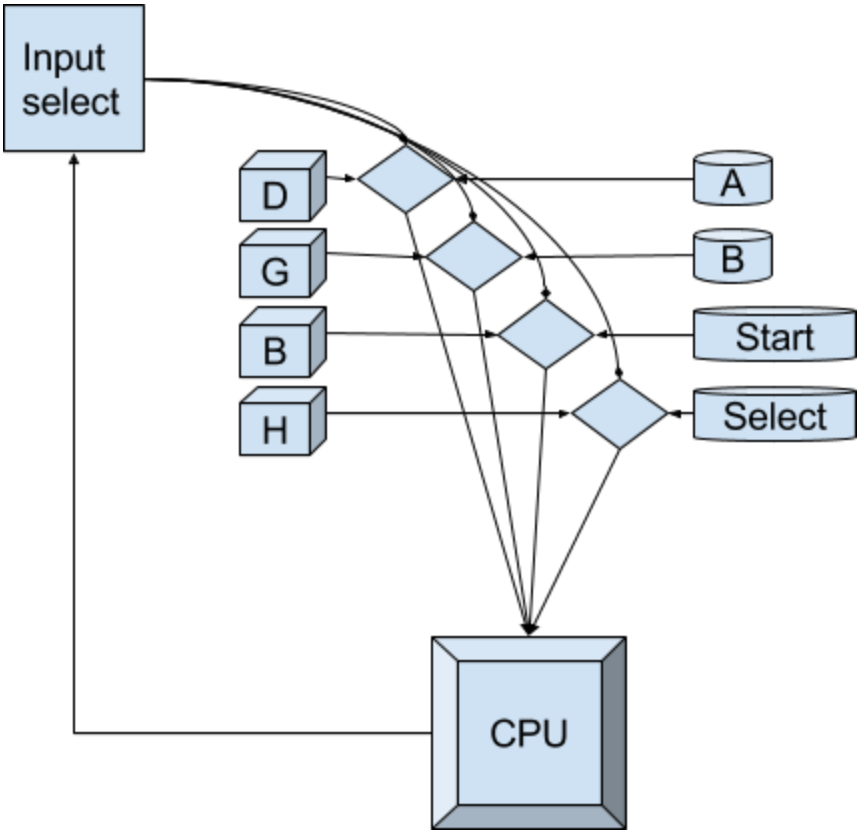
Une gameboy a 8 entrées basiques
Un joystick 4 positions et 4 boutons appelés : A, B, Start, Select.

Une pression d'un bouton déclenche une interruption, le programme peut ensuite vérifier quel bouton a été pressé.

Un seul registre permet de vérifier l'état de chaque bouton : FF00

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
N/A	N/A	Boutons	Joystick	Bas/Start	Haut/Select	Gauche/B	Droite/A

On voit qu'il faut d'abord set le bit 5 ou le bit 4 à 1 pour ensuite pouvoir vérifier l'état des entrées



Gestion des entrées

Affichage

L'écran de la console est un écran lcd d'une résolution de 160*144 pixels.
Il est possible d'afficher 4 degrés de gris différents.

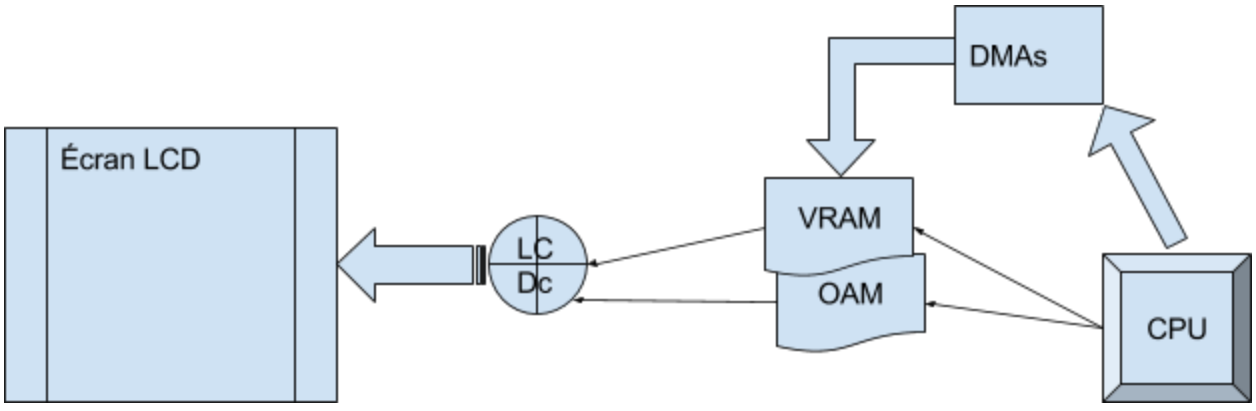
Entre autre, la gameboy possède un moteur dédié pour gérer les sprites.

L'écran LCD de la gameboy a un fonctionnement particulier.
Le contrôleur de l'écran va osciller périodiquement entre les quatre états

État 0	Les mémoires VRAM et OAM peuvent être modifiées par le CPU
État 1	Les mémoires VRAM et OAM peuvent être modifiées par le CPU, l'écran peut aussi être désactivé
État 2	Le contrôleur LCD lit la mémoire OAM, elle est inaccessible par le CPU
État 3	Le contrôleur LCD lit la mémoire OAM et VRAM, elles sont inaccessibles par le CPU

Ces restrictions sont dûs à une limitation matérielle, en effet si on essaie de désactiver l'écran en dehors de l'état 1, il peut être abîmé.

Pour remédier à ceci, la gameboy possède un DMA (h-dma) qui prend en compte ces limitations et peuvent donc copier dans la VRAM ou OAM en toute sécurité. Il existe aussi un general purpose DMA mais il ne prend pas en compte les limitations, à n'utiliser que dans les états 0 ou 1.



Gestion de l'affichage

Son

Il y a 4 canaux de son, les 4 canaux sont ensuite redirigés ensemble dans deux sélecteurs pour être envoyés dans les hauts-parleurs.

Canal 1	Tonalité et balayage
Canal 2	Tonalité
Canal 3	Signal ondulatoire
Canal 4	Bruit

Ces 4 canaux servent des utilités différentes, en effet on voit que deux d'entre eux ne peuvent pas faire des "jolis sons" (C3 et C4), mais gardons à l'esprit que c'est une vieille console.

Le canal 1 est utilisé pour émettre un son, il permet aussi un balayage de fréquence.
Le canal 2 est utilisé pour émettre un son
Le canal 3 lit dans une ram pour émettre un signal
Le canal 4 émet un bruit blanc

Un accès au registre FF25 permet de choisir 2 de ces 4 canaux pour être sortis sur les hauts-parleurs.

On peut ensuite aussi modifier le volume de chaque sélecteur.

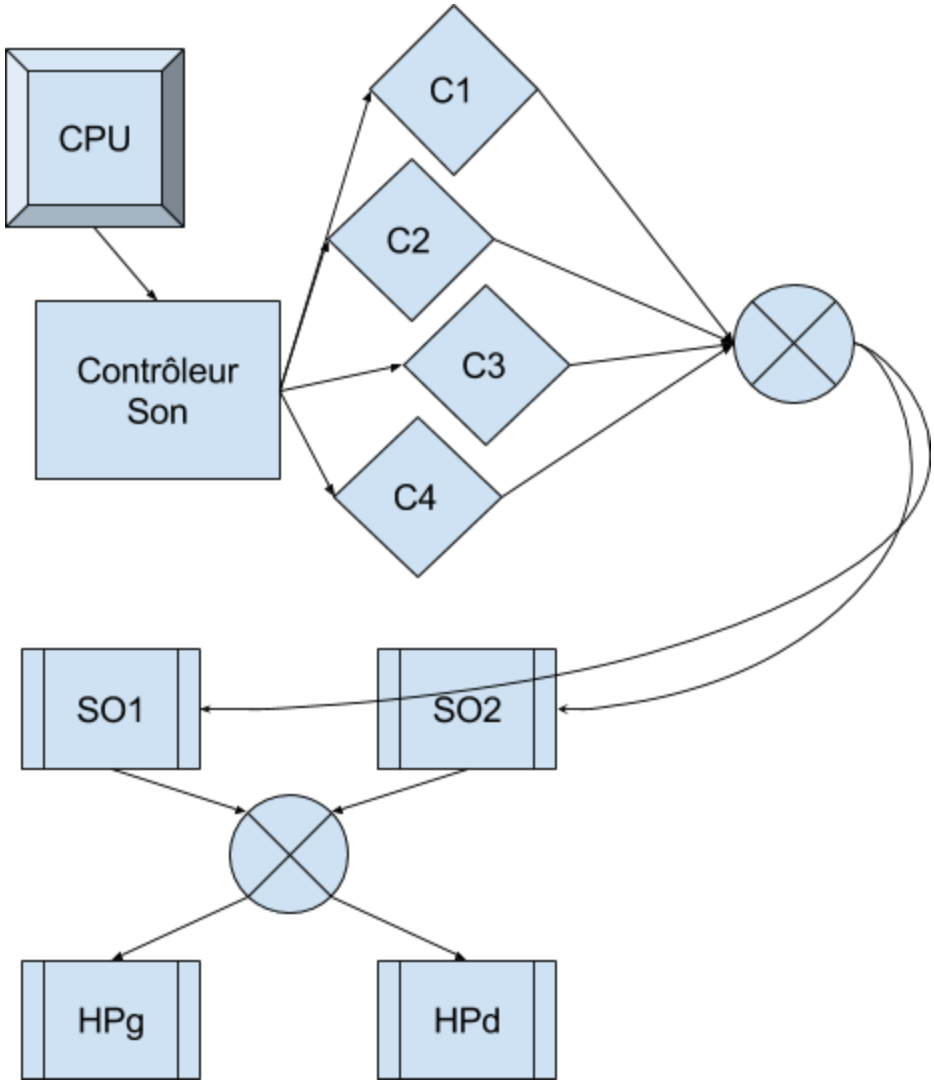


Schéma du système gérant le son

LPC4337

Introduction

Cette partie du rapport s'intéresse à toute la partie micro-contrôleur du projet.

Le lpc4337 est un micro-contrôleur à deux cœurs ayant une fréquence d'horloge pouvant atteindre 204Mhz. Il a été choisi pour ce projet dû à ces deux cœurs par soucis de performance, en effet émuler un autre processeur est une grosse tâche et nous allons voir pourquoi.

Multi-Cœurs

Le projet est séparé dans les deux cœurs, un Cortex-M4 et un Cortex-M0+

La partie M4 a les tâches suivantes :

- Émulation du processeur gameboy
- Gestion des entrées Joystick & boutons
- Lecture de la carte Micro-SD

L'émulation du processeur est une tâche extrêmement lourde. La gestion des entrées est plutôt simple et la lecture de la carte ne prend pas beaucoup de ressources.

La partie M0+ quant à elle :

- Gestion de l'écran
- Gestion du son

La gestion de l'écran est très lourde si on veut atteindre un bon taux de rafraîchissement.
La gestion du son elle est compliquée à mettre en place mais ne devrait pas poser trop de problèmes de dimensionnement.

Émulation

La partie principale d'un simulateur gameboy, l'émulation du processeur.

Pour émuler un processeur il faut préparer plusieurs choses.

Mémoire

Une partie simple, il suffit de préparer des variables pour remplacer les registres et le micro-contrôleur aura assez de RAM (136 kB de SRAM contre 16 kB pour une Gameboy) pour gérer cela aisément.

Pour les registres on utilise une structure-union-structure

```
struct registers {
	struct {
		union {
			struct {
				unsigned char a;
				unsigned char f;
			};
			unsigned short af;
		};
	};
```

Ceci nous permettra de gérer les registres en mode 8 bits comme en mode 16 bits
Pour la mémoire nous allons créer un tableau de shorts, l'adresse sera utilisée comme index du tableau.

Tous les accès mémoires seront redirigés vers le tableau cité plus haut et les accès aux registres seront redirigés vers notre structure.

Processeur

Pour le processeur, la tâche est plus ardue.
Nous allons suivre le principe de fetch -> decode -> execute :

- Fetch
  - On récupère
- Decode
  - On lit toute l'instruction et la compare à notre jeu d'instruction
- Execute
  - On lance une fonction émulant l'instruction

Théoriquement, pour chaque instruction du processeur il faut avoir une fonction qui y correspond, on a donc un tableau d'instructions, cela prend de la place, mais permet de vite trouver les correspondances

Échantillon de notre tableau[1]

```
const struct instruction instructions[256] = {
        { "NOP", 0, nop },                          // 0x00
        { "LD BC, 0x%04X", 2, ld_bc_nn },           // 0x01
        { "LD (BC), A", 0, ld_bcp_a },              // 0x02
        { "INC BC", 0, inc_bc },                    // 0x03
        { "INC B", 0, inc_b },                      // 0x04
        { "DEC B", 0, dec_b },                      // 0x05
        { "LD B, 0x%02X", 1, ld_b_n },              // 0x06
        { "RLCA", 0, rlca },                        // 0x07
        { "LD (0x%04X), SP", 2, ld_nnp_sp },        // 0x08
        { "ADD HL, BC", 0, add_hl_bc },             // 0x09
        { "LD A, (BC)", 0, ld_a_bcp },              // 0x0a
        { "DEC BC", 0, dec_bc },                    // 0x0b
        { "INC C", 0, inc_c },                      // 0x0c
        { "DEC C", 0, dec_c },                      // 0x0d
        { "LD C, 0x%02X", 1, ld_c_n },              // 0x0e
        { "RRCA", 0, rrca },                        // 0x0f
        { "STOP", 1, stop },                        // 0x10
        { "LD DE, 0x%04X", 2, ld_de_nn },           // 0x11
        { "LD (DE), A", 0, ld_dep_a },              // 0x12
        { "INC DE", 0, inc_de },                    // 0x13
        { "INC D", 0, inc_d },                      // 0x14
        { "DEC D", 0, dec_d },                      // 0x15
        { "LD D, 0x%02X", 1, ld_d_n },              // 0x16
        { "RLA", 0, rla },                          // 0x17
        { "JR 0x%02X", 1, jr_n },                   // 0x18
        { "ADD HL, DE", 0, add_hl_de },             // 0x19
        { "LD A, (DE)", 0, ld_a_dep },              // 0x1a
```

Pour chaque instruction, nous avons son assembleur, son nombre de paramètre et la fonction à appeler.

[1] Tableau d'instructions de Cinoop

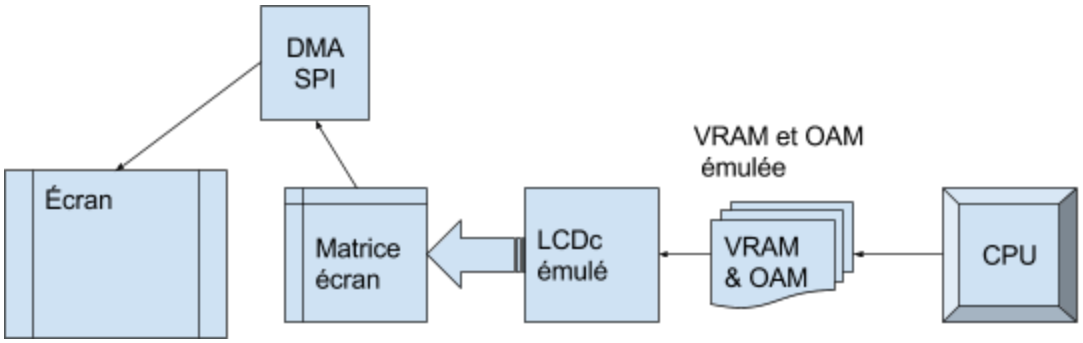
Périphériques utilisés

Nous utilisons plusieurs périphériques pour que ce projet puisse fonctionner, après tout il faut simuler le système entier de la gameboy.

Un écran pour l'affichage, Un convertisseur CNA pour le son, des boutons et du tactile pour les entrées ainsi qu'une carte micro-sd pour jouer le rôle de la cartouche de jeu.

Écran

La partie écran se fait en deux couches, nous avons la couche gameboy qui imite le fonctionnement de la console originale gérant une matrice représentant l'écran et un système SPI recopiant cette matrice sur notre véritable écran.



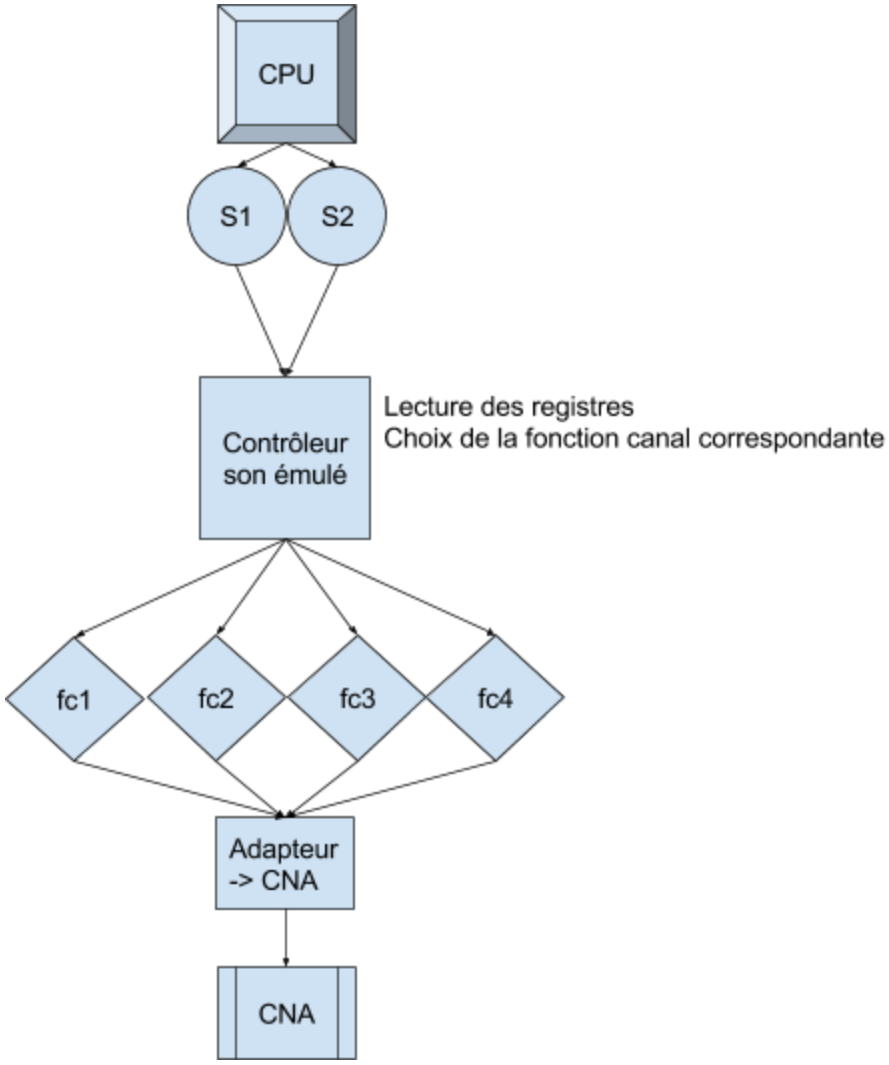
On fait ici cette gestion en deux étapes car la gameboy a une manière particulière de gérer son affichage notamment en séparant ses sprites du reste, il faut donc rester fidèle à ce fonctionnement pour ne pas avoir des affichages erronés.

L'écran de la mylab2 est un écran tactile couleurs avec une résolution bien plus grande que celle d'une gameboy, dans un premier temps il faut éviter d'agrandir l'écran gameboy, car les calculs sont complexes, mais on peut finir par le faire.

CNA

Le CNA est utilisé pour sortir du son. Le fonctionnement du système de son de la gameboy est bien différent et bien plus compliqué que les appareils de maintenant, nous allons devoir procéder en deux couches comme pour l'écran. Nous allons lire en temps réel, directement les sorties des sélecteurs, les adapter à notre CNA puis le transformer pour avoir un signal son pour nous.

C'est une tâche compliquée, il faut adapter le code pour chaque canal son de la gameboy, en effet nous avons vu qu'ils avaient tous des façons de fonctionner différentes (cf: Son).

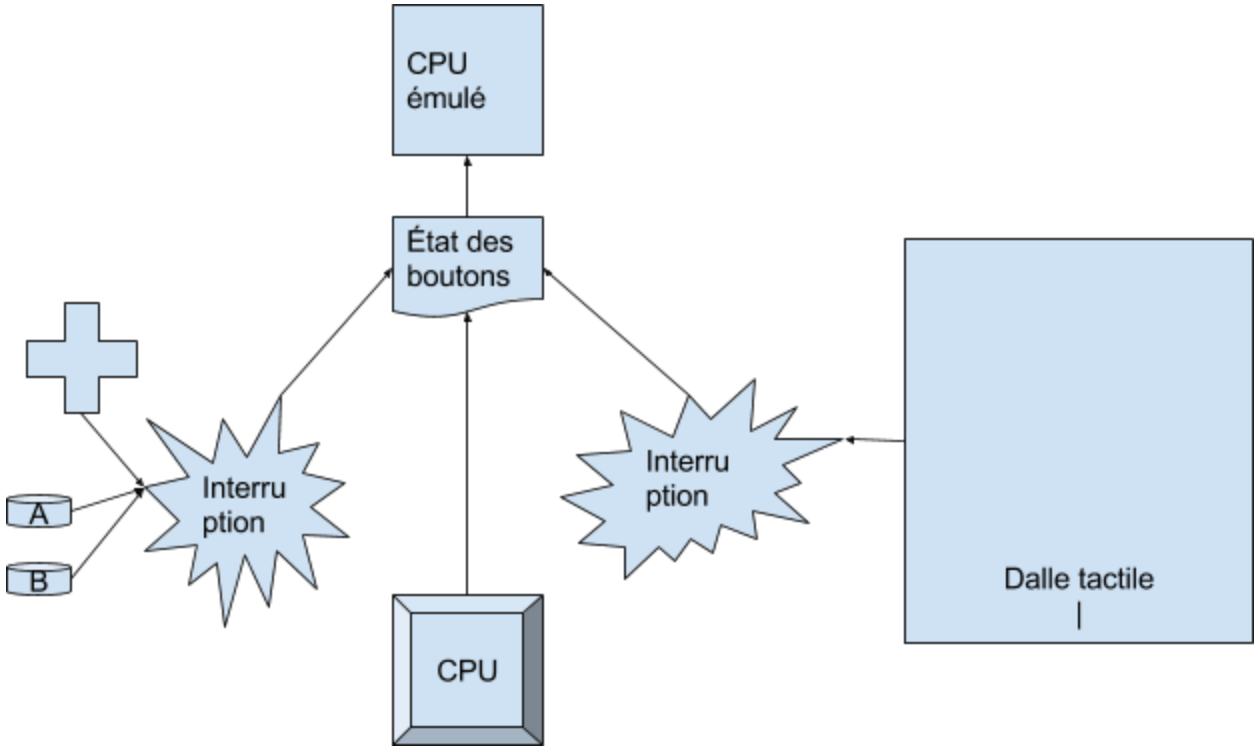


Dès que c'est fait, on peut sortir le son.

Boutons

Gérer les entrées devient plus facile. La gameboy fonctionne par interruption, et nous aussi. À chaque interruption GPIO correspondant à un bouton ou au joystick, on met juste à jour une variable dans laquelle on stocke l'état de nos entrées au moment de l'interruption, quand la gameboy fait son interruption, nous avons vu qu'elle va vérifier l'état des entrées, il suffit de lui montrer nos états à nous.

Cependant, la mylab 2 a un joystick certes, mais que 2 boutons contre 4 de la gameboy. Il manque les boutons Start et Select. Nous avons décidé d'émuler ces deux boutons par la dalle tactile, de faux boutons en quelque sorte. Principe similaire à avant, il suffit, après la pression sur la dalle tactile, de venir mettre à jour la variable.



Relativement simple à mettre en place pour les boutons par GPIO, pour la dalle tactile, elle est gérée en I²C, ce qui n'empêche pas la mise en place d'interruption non plus.

Micro-SD

Pour émuler la cartouche, il nous faut un espace de stockage externe, en effet une cartouche a beaucoup de mémoire et on ne peut pas la stocker dans la SRAM du micro-contrôleur.

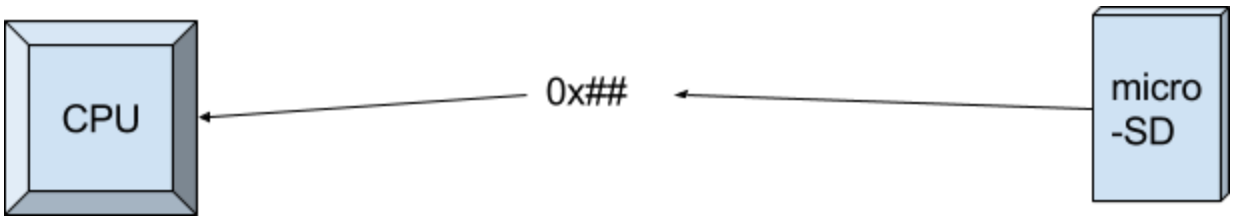
Pour remédier à ça nous utilisons une Micro-SD contrôlée en SPI. À partir de là deux choix s'offrent à nous

1. On utilise la Micro-SD comme un flux direct sans système de fichier
    a. Rapide
    b. Simple
    c. Presque natif pour le processeur émulé

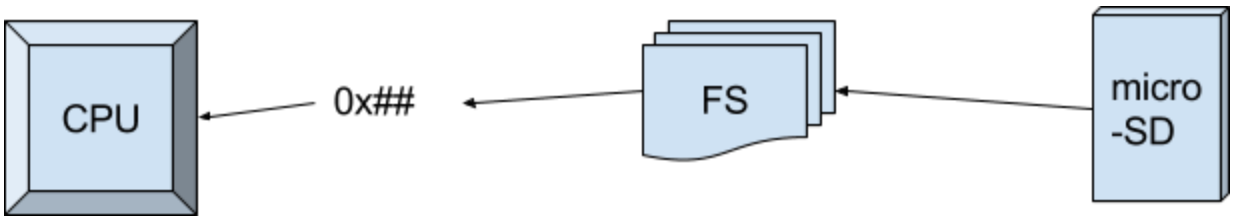
2. On implémente un système de fichier FAT
    a. Simple à gérer depuis un ordinateur
    b. Possibilité de gérer plusieurs jeux

Au départ nous partons sur le premier choix, ce sera plus facile à tester et plus rapide, ensuite éventuellement, implémenter un système FAT est une possibilité.
Avec un système FAT, nous pourrions gérer plusieurs jeux et proposer le choix à l'utilisateur. Cependant, il faudra utiliser la librairie correspondante, et nous devons d'abord nous occuper du bon fonctionnement de la base.

Dans les deux cas, nous lisons un octet, et on le donne à lire au processeur pour qu'il le traite. Bien entendu on a les cas ou le cpu voudra lire plus de données pour prendre les paramètres que certaines instructions pourraient demander.



Dans le cas du FAT, nous passons par un système de fichiers mais nous pouvons quand même accéder à notre jeu, il y a juste une couche supplémentaire



Travail de Semestre

État du travail

Le travail est, à son stade de travail de semestre, non-terminé.

Nous avons pour l'instant :

- Périphériques
    - Communication basique avec l'écran
- Mémoire
    - Architecture mémoire mise en place
- Processeur
    - Jeu d'instructions mis en place, fonctions non disponibles
    - Émulation pas encore adaptée au micro-contrôleur mais fonctions de bases présentes

Pour la partie émulation, nous réutilisons une partie du code de Cinoop, un émulateur open-source pour windows, mais surtout les idées derrières.

Les périphériques n'ont malheureusement pas encore été implémentés, même l'écran qui est maintenant accessible, n'est pas encore optimisé pour que les jeux soient "jouables"

Maintenant que ce document est préparé, il reste à l'appliquer.

1. Gestion standard des périphériques
    a. Et les tests qui vont avec
2. Émulation du processeur
    a. Cadençement
3. Couches de compatibilité de l'écran
    a. Pour tester avec un programme gameboy : helloworld
4. Couche de compatibilité des entrées
5. Couche de compatibilité Micro-SD
6. Test complet d'un jeu entier gameboy
7. Couche de compatibilité Son
    a. On s'en occupe en dernier, car finalement un jeu peut être jouable sans le son
8. Paufinage
    a. Agrandissement de l'écran
    b. Système de fichier pour la micro-sd

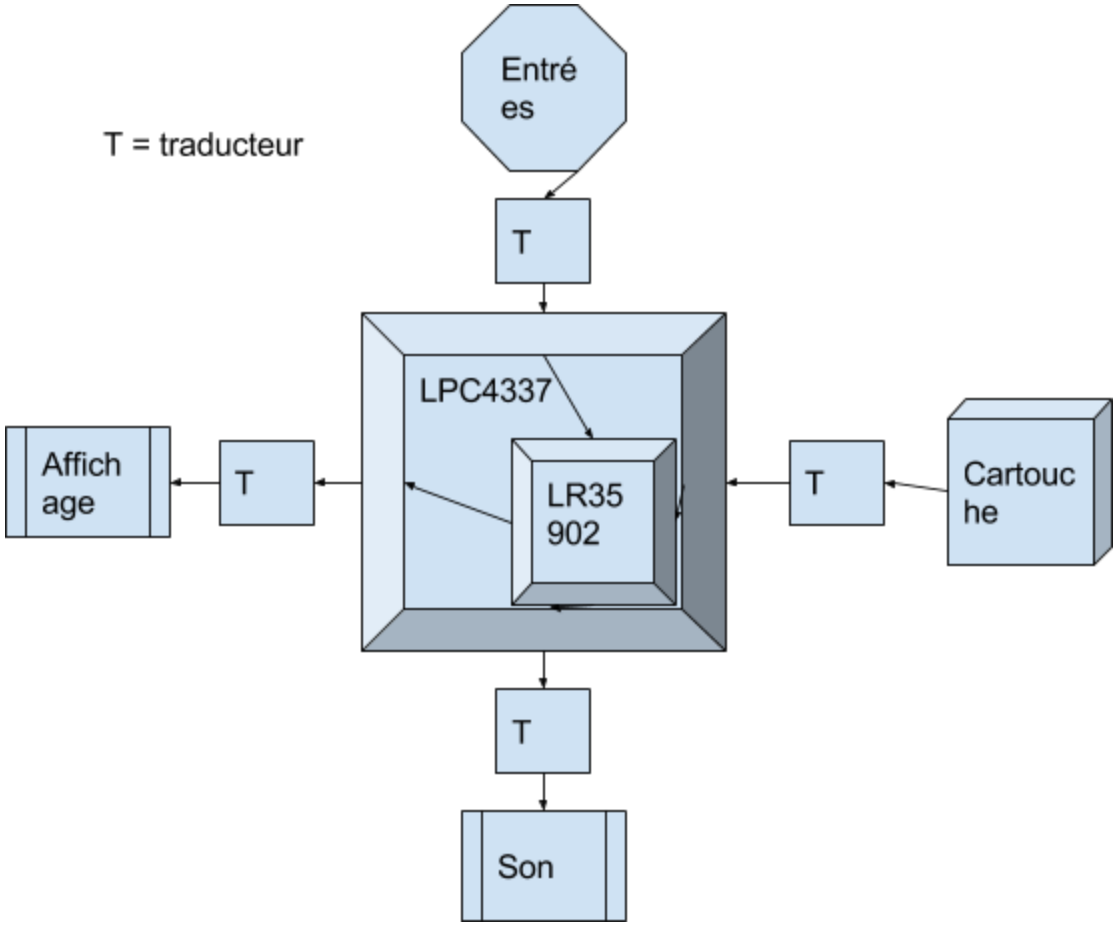
Éventuellement, pour pousser le projet encore plus loin, une gameboy couleur a un fonctionnement similaire à la version classique, on pourrait donc améliorer ce simulateur pour gérer la couleur aussi, mais ceci va demander des calculs de dimensionnements.

Conclusion

Il y a eu beaucoup de recherches, que ce soit de ressources ou de données techniques. Beaucoup d'essais sur machine, bien qu'ils soient seulement un aspect de debugging.

Tout ce travail n'a finalement pas abouti à un simulateur utilisable, ce qui n'est pas une surprise, mais nous avons maintenant toutes les cartes en main pour le faire.
Grâce à ce rapport, montrant comment fonctionne une gameboy et comment émuler son système sur notre micro-contrôleur, nous avons un chemin tout tracé permettant de faire cette console signé Hepia.

Que ce soit l'auteur de ce document ou quelqu'un d'autre qui crée ce simulateur, ce document sera d'une très grande utilité pour y aboutir.



Bibliographie

Jeu d’instructions :
http://www.pastraiser.com/cpu/gameboy/gameboy_opcodes.html

Wiki de développement :
http://gbdev.gg8.se/wiki/articles/Main_Page

Cinoop :
https://github.com/CTurt/Cinoop
https://cturt.github.io/cinoop.html

Javascript emulator
http://imrannazar.com/GameBoy-Emulation-in-JavaScript

La documentation LPC4337