MyLab2 GB
Emulateur de Gameboy sur MyLab2

Orphée Antoniadis

Projet de semestre - Prof. Fabien Vannel

Hepia ITI 3ème année

Semestre d'automne 2017-2018

Table des matières

1 Introduction

1.1 Objectif

L'objectif principal de ce projet était de développer un émulateur de la console de jeux vidéos de Nintendo appelée Gameboy pour la carte de d'extension MyLab2 développée à l'hepia. Un emulateur est un logiciel permettant d'imiter le comportement physique d'un matériel informatique. Le but ici est donc d'imiter le comportement du processeur de la Gameboy avec tous ces périphériques.

Il a fallut dans un premier temps comprendre le fonctionnement de la Gameboy. Le projet a donc commencé par un travail de recherche et de documentation. Etant donné que Nintendo n'a jamais rendu public la datasheet de sa console, toutes les informations récoltées ont été obtenues après un travail de reverse engineering effectuée par la communauté. Heureusement, la communauté de développeurs d'émulateurs est très active, de nombreux forums et blogs existent ce qui a simplifié la recherche d'informations. Toutes les sources que j'ai trouvé seront listées à la fin de se document et pourront faire office de base de donnée complète pour quiconque voulant en apprendre plus sur la console ou bien même voulant développer son propre émulateur.

1.2 Méthode

1.3 Schéma

2 MyLab2

2.1 Introduction

La MyLab2 est une carte d'extension pour les kits de développement des microcontrolleurs LPC1769 et LPC4337 de NXP. Cette carte a été développée à hepia, au laboratoire de systèmes numériques. Elle vient se superposer aux cartes de NXP via des connecteurs broches.

Le LPC1769 de NXP est un microcontrolleur ARM Cortex-M3 ayant une fréquence d'horloge pouvant aller jusqu'à 120MHz. Ce dernier propose une mémoire flash de 512kB (mais seulement la moitié est programmable avec la version gratuite de LPCXpresso) ainsi qu'une SRAM de 64kB. A noter que les 64kB de SRAM sont discontinus et qu'il y a en réalité deux banques de RAM de 32kB chacune. Le LPC4337 de NXP est un microcontrolleur ARM Cortex-M4 ayant une fréquence d'horloge pouvant aller jusqu'à 204MHz mais inclut aussi un coprocesseur ARM Cortex-M0 pouvant aller à la même fréquence d'horloge. La SRAM de ce dernier fait 136kB.

Pour ce projet, le microcontrolleur utilisé sera le LPC1769 pour un soucis pratique. En effet, nous avons déjà eu à utiliser cette carte pour le cours de microcontrolleurs et périphériques de deuxième année ce qui fait que j'avais déjà une grande partie de la librairie de gestion des périphériques de la carte d'extension prête. C'était un gain de temps non négligeable qui m'a permis de me concentrer sur l'émulation de la Gameboy.

Le projet pourrait tout de même être repris et porté sur le LPC4337 afin d'avoir de meilleures performances étant donné que sa fréquence d'horloge est deux fois plus grande, qu'il dispose de deux processeurs permettant une parallélisation des taches et qu'il y a plus de RAM.

2.2 Pérphériques utilisés

2.2.1 Boutons

Les boutons A et B de la MyLab2 sont de simples pins GPIO. Le LPC1769 propose des interruptions (EINT3) sur les ports 0 et 2 du GPIO. Ici, le bouton A est sur la pin 10 du port 2 (P2.10) et le bouton B est sur la pin 19 du port 0 (P0.19). L'interruption peut se faire sur le flanc montant ou descendant (ou les deux). Pour l'activer, il faut mettre le bit correspondant à 1 dans le registre `IntEnF` (flanc descendant) ou `IntEnR` (flanc montant). Lors d'une interruption, il faut regarder laquelle a eu lieu dans le registre `IntStatF`/`IntStatR` puis la quittancer en mettant le bon bit à 1 dans le registre `IntClr`.

2.2.2 Joystick

Le joystick de la MyLab2 est aussi relié à une pin GPIO mais contrairement aux boutons A et B, il est relié au port 1 du GPIO. Il n'y a pas d'interruptions sur le port 1 du GPIO, il faut donc vérifier l'état des pins du joystick de manière régulière. Pour se faire, un timer sera utilisé qui provoquera une interruption (TIMER0) toutes les 10ms. La routine d'interruption appelle la fonction `joystick_handler` qui prend comme argument une fonction de callback (un pointeur sur une fonction qui sera appelée si une des directions du joystick est appuyée), l'argument de cette dernière, et le mode de vérification (`POLLING` ou `TRIGGER`). En mode `POLLING`, la fonction de callback sera appelée qu'une seule fois si le joystick est maintenu appuyé contrairement au mode `TRIGGER` où la fonction sera appelée tant que le joystick n'est pas relâché. La fonction de callback doit avoir comme prototype :

```
void joystick_callback(uint8_t pos, uint8_t edge, void *arg)
```

avec `pos` qui est la position du joystick, `edge` qui est le flanc (montant ou descendant) et `arg` qui contient l'argument de la fonction (peut être à `NULL`).

2.2.3 Ecran LCD

La MyLab2 utilise le le bus de communication SPI pour communiquer avec l'écran. Afin d'initialiser le SPI, il faut d'abord sélectionner les bonnes pins dans les registres appropriés (`PINSEL0` et `PINSEL1`). En effet, les pins sont configurées par défaut sur des pins GPIO. Pour le LPC1769 ce sont donc les bits 31 :30 de `PINSEL0` et 5 :0 de `PINSEL1` qu'il faut modifier.

Il faut ensuite fixer la valeur de la fréquence de transmission. Un registre de configuration du SPI le permet. C'est le registre `SPCCR`. La valeur donné à ce registre va diviser la valeur de l'horloge du SPI. Pour atteindre une fréquence de transmission maximale, il faut mettre l'horloge du SPI à 100MHz en modifiant le registre `PCLKSEL0` puis fixer la valeur du registre `SPCCR` à 10 pour avoir une fréquence de transmission de 10MHz ($\frac{100}{10}$). Pour finir il faut juste activer la communication SPI en la mettant en master mode. Ceci se fait dans le registre `SPCR` (bit 5 à 1).

L'écriture et la lecture du bus SPI se fait à l'aide du même registre pour le LPC1769. Ce registre est le registre `SPDR`. Pour écrire, il faut fixer la valeur du registre aux données à envoyer puis attendre que le flag de confirmation d'envoi soit mis à 1. Ce flag est le bit 7 du registre `SPSR`.
Pour venir lire sur le bus SPI, il faut d'abord envoyer 0xFF (donc en utilisant la fonction d'écriture) puis venir lire sur ce même registre (buffer bi-directionnel).

L'écran LCD accepte 2 types de données, les instructions et les arguments d'instruction. La pin DC (pin GPIO 1.16) permet de contrôler ce qui va être envoyé. Le DC est à 0 lors de l'envoi d'une instruction et à 1 pour l'envoi d'un argument. La liste complète des instructions est disponible dans la documentation de l'écran LCD. Pour ce projet, seulement 3 de ces instructions sont utilisées. Celle pour séléctionner la colonne d'écriture en pixels (instruction 0x2A), celle pour séléctionner la ligne d'écriture en pixels (instruction 0x2B) et enfin, celle pour écrire dans la mémoire de l'écran (instruction 0x2C). La mémoire est l'ensemble des données qui composent l'intégralité des pixels de l'écran. Ici l'écran est de 240×320 et un pixel est codé sur 16 bits donc la mémoire serait ces $240 \times 320 \times 16$ bits. Pour allumer un pixel sur l'écran, il faut d'abord séléctionner la zone d'écriture en utilisant les instructions 0x2A et 0x2B, puis il faut écrire les données relatives à la couleur du pixel désirée en utilisant l'instruction 0x2C).

2.2.4 Dalle tactile

La MyLab2 communique avec la dalle tactile en utilisant le bus I^2C. La détection d'appuie sur la dalle tactile se fait quand à elle par interruption GPIO. Il faut donc activer d'abord ces interruptions (comme pour les boutons A et B) puis récupérer les informations voulues dans la routine d'interruption. Les informations sont par exemple les coordonnées de position et peuvent être lues dans les registres de la dalle tactile dont les adresses sont disponibles dans la documentation. Pour ce projet, il n'y a eu besoin que de récupérer des coordonnées de la position détectée. Pour récupérer ces informations, il faut d'abord envoyer (donc écrire sur le bus I^2C) l'adresse du registre à lire et ensuite venir lire dans ce registre.

A noter qu'il faut bien entendu initialiser le bus I^2C ce qui est fait relativement de la même manière que le bus SPI. Petite différence juste pour fixer la fréquence de transmission qui se fait par deux registres, les registres `I2SCLH` et `I2SCLL`. Ces registres correspondent respectivement à la fréquence de l'horloge à l'état haut et à l'état bas. La fréquence du bus I^2C en fast mode est de 400kHz et l'horloge périphérique est à 25MHz par défaut donc il faut mettre ces deux registres à 32 ($\frac{25000}{32+32} = 400$). Il faut pour finir activer les interruptions I^2C et créer l'algorithme représenté par un organigramme dans la documentation du LPC1769.

3 Gameboy

3.1 Introduction

La Gameboy est une console de jeux vidéos portable 8 bits développée et fabriquée par la firme japonaise Nintendo. La console a été mise en vente en 1989 et a connu un franc succès à travers le monde jusqu'à la fin de sa production en 2003. Nous allons dans cette partie étudier les caractéristiques et les périphériques de la console.

Le processeur de la Gameboy est le LR35902, processeur semblable au Zilog Z80 et cadensé à 4,19MHz. La mémoire de la console est de 64kB. Elle possède quatre boutons, A, B, START et SELECT, une croix directionnelle et une fente pour insérer les cartouches de jeux qui était située sur le haut, à l'arrière de la console. Elle est aussi dotée de hauts parleurs et d'un port série pour la communication entre deux consoles. Le son et la communication série ne seront pas emulés pour ce projet mais une amélioration future pourrait les rajouter. Divisons donc toutes ces parties en blocs.

3.2 CPU

3.2.1 Architecture

Comme dit précédemment, le processeur de la Gameboy est semblable au Zilog Z80. C'est en fait un hybride entre le Zilog Z80 et le Intel 8080. Le Intel 8080 a été conçu pour être compatible avec le Zilog Z80 ce qui veut dire que les instructions du 8080 sont aussi présentes dans le Z80. Le nom de ce processeur hybride est Sharp LR35902.

Le processeur de la Gameboy est un processeur 8 bits avec 16 bits de bus d'adresse. Il utilise 6 registres de 16 bits dont 4 pouvant être séparés en deux registres de 8 bits. Il donne aussi accès à 256 instructions (2^8) plus 256 instructions supplémentaires et ses 16 bits de bus d'adresse permet d'adresser 64kB de mémoire. Regardons maintenant plus en détail les regsitres du LR35902.

3.2.2 Registres

Ci-dessous, un tableau explicatif des 6 registres du processeur de la Gameboy.

Registres	15 :8	7 :0
AF	A : registre	F : flags
BC	B : registre	C : registre
DE	D : registre	E : registre
HL	H : registre	L : registre
SP	Stack pointer	
PC	Program counter	

Les regsitres, A, BC, DE et HL sont les registres de travail donc utilisés pour toutes les opérations du programme. Le registre F contient les flags du processeur. Il y a 4 flags différents.

- Z (Zero flag), qui indique si le résultat de la dernière opération était 0
- N (Negative flag), qui indique si le résultat de la dernière opération était négatif
- H (Halfcarry flag), qui indique s'il y a eu une retenue sur les 4 premiers bits
- C (Carry flag), qui indique s'il y a eu une retenue

Note : le Halfcarry flag était en général utilisé pour les affichages en décimal sur l'écran (par exemple le score du joueur).

3.2.3 Instructions

Nous avons vu plus haut que le processeur de la Gameboy donnait accès à 256×2 instructions alors qu'une instruction était sur 8 bits. L'instruction 0xCB était en fait une instruction préfix. Toute instruction donnée après ce prefix faisait partie des 256 instructions supplémentaires. Nous verrons un exemple d'instruction plus bas. Regardons d'abord le jeu d'instructions de la Gameboy et divisons les en catégories.

Instructions arithmériques

- INC, incrémentation du registre
- DEC, décrémentation du registre
- ADD, addition de deux registres
- SUB, soustraction de deux registres
- ADC, addition de deux registres et du carry flag
- SBC, soustraction de deux registres et du carry flag

Opérations logiques

- AND, ET bit à bit
- OR, OU bit à bit
- XOR, OU exclusif bit à bit

Opérations sur les bits

- RL, rotation à gauche
- RR, rotation à droite
- SL, décalage à gauche
- SR, décalage à droite
- SWAP, inversion des bits
- BIT, teste un bit
- SET, set un bit
- RES, reset un bit

Sauts

- JP, saut absolu à une adresse
- JR, saut relatif
- CALL, saut et stockage du program counter sur la pile (appel de fonction)
- RET, saut à l'adresse stockée dans la pile (retour de fonction)

Opérations avec la pile

- PUSH, stockage d'un registre dans la pile
- POP, récupération de la dernière donnée empilée

Commandes processeur

- NOP, pas d'opération
- HALT, arrête le processeur jusqu'à la prochaine interruption
- STOP, arrête le processeur
- EI, active une interruption
- DI, désactive une interruption

Autres instructions

- LOAD, stockage d'une donnée depuis ou dans la mémoire
- CP, comparaison de deux registres
- DAA, conversion du registre A en décimal

Exemples d'instructions

- 0x11 0x10 0x10 → LD DE, 0x1010
- 0xCB 0xC0 → SET 0, B

3.3 Mémoire

3.3.1 Memory map

Nous avons vu à plusieurs reprises que la Gameboy avait une mémoire de 64kB. Voyons plus en détail la structure de cette mémoire.

Adresses	Nom	Description
0000h – 3FFFh	ROM0	ROM non-échangeable
4000h – 7FFFh	ROMX	ROM échangeable
8000h – 9FFFh	VRAM	Video RAM
A000h – BFFFh	SRAM	RAM externe
C000h – CFFFh	WRAM0	Work RAM
D000h – DFFFh	WRAMX	Work RAM échangeable
E000h – FDFFh	ECHO	Echo de la WRAM
FE00h – FE9Fh	OAM	Sprites
FEA0h – FEFFh	UNUSED	-
FF00h – FF7Fh	I/O Registers	-
FF80h – FFFEh	HRAM	RAM interne du CPU
FFFFh	IE Register	Registre d'activation des interruptions

3.3.2 Cartouche

La cartouche est un périphérique externe sur la Gameboy originale. Tout le code qui est executé par la Gameboy est contenu dans la cartouche. Le jeu est mappé dans les 32 premiers kB de la mémoire de la console. Le problème qui se pose ici est que les jeux ne peuvent faire une taille que de 32kB si on ne prend en compte que cette information. Pour résoudre ce problème, la Gameboy met à disposition un mécanisme d'échanges de blocs mémoires entre sa mémoire et la mémoire du jeu (appelé MBC).

Ces échanges se font aux adresses allant de 0x4000 à 0x7FFF. Il existe de nombreux types de MBC dont la liste est disponible sur internet. Afin de savoir à quel type de MBC nous avons à faire, il faut lire à l'adresse 0x147. Il faut écrire à une adresse entre 0x2000 et 0x4000 pour changer de banque de ROM et écrire entre 0x4000 et 0x6000 pour changer de banque de RAM. L'échange de banques de RAM n'est pas activé par défaut. Le jeu doit d'abord écrire 0xA à une adresse entre 0x0000 et 0x2000. A noté que toutes ces écritues ne sont pas effectives, elle permettent juste de gérer les banques de mémoire. Aucune donnée n'est écrite en ROM.

3.3.3 Bootstrap

Dans la partie précédente, nous avons vu que les 32 premiers kB du jeu étaient mappés au début de la mémoire de la console. Le seul moment ou ce n'est pas le cas est durant le démarrage de la console. Lors du démarrage de la console, le processeur exécute un petit bout de code allant de 0x0 à 0x100. Ce bout de code est stocké en dur dans la console et permet d'initialiser les registres et la mémoire de la Gameboy. Le code de bootstrap n'a jamais été donné par Nintendo et n'a été que récemment décodé par la communauté de développeurs. Ce code est donc mappé dans les 256 premiers bytes de la console, le program counter commence à 0 puis après avoir executé l'ensemble des instructions, il arrive à 0x100. L'éxécution du bootstrap était représenté par un scroll du logo de Nintendo sur l'écran. Ensuite, Le bootstrap est retiré et le jeu peu commencer son éxécution. A noter que le code du jeu commence donc à l'adresse 0x100. Les 256 premiers bytes contiennent les routines d'interruption.

3.3.4 DMA

Le registre DMA (0xFF46) permet d'activer le transfert DMA, entre tout adresse (entre 0x0 et 0xF19F) et l'OAM (mémoire contenant les sprites). Le transfert commence quand l'adresse source st écrite dans le registre DMA. Etant donné que le registre ne fait que 8 bits, l'adresse source est multipliée par 0x100

et donne la réelle adresse de départ du transfert.

Une fois que le transfert DMA commence, les données commençant à l'adresse écrite dans le registre DMA sont écrites dans la région OAM de la mémoire. Il y a 160 bytes écrits lors de ce transfert (taille de la région OAM). L'avantage du DMA est de permettre de copier l'intégralité des données de sprites en une seule instruction au lieu d'avoir à faire 160 instructions à la suite.

3.4 Interruptions

3.4.1 Activer une interruption

La Gameboy est capable de générer 5 types d'interruptions pour certains de ses périphériques. Ces interruptions pouvaient être contrôlées depuis deux registres contenus dans la mémoire. Le registre IE (0xFFFF) et le registre IF (0xFF0F). Les 5 premiers bits de ces deux registres représentaient chacun une interruption.

Bit	Interruption	Adresse
0	V-Blank	40h
1	LCD	48h
2	Timer	50h
3	Serial	58h
4	Joypad	60h

Ici, l'adresse est l'adresse dans la mémoire qui contient la routine d'interruption (comme expliqué en 3.3.2). Le registre IE (Interrupt Enable) est modifié par le jeu. Le registre IF (Interrupt Flag) quand à lui est modifié par les périphériques lorsqu'une interruption est demandée.

La dernière chose à savoir sur la gestion des interruptions est le "interrupt master". Il n'est pas contenu dans la mémoire de la console et est un simple booléen que les jeux mettent à 1 ou à 0. Si il est à 1, aucune interruption ne pourra être traité. C'est donc un interrupteur qui est mis à 0 ou à 1 par les instructions EI, DI ou RETI.

3.4.2 Demander une interruption

Comme dit précédemment, un périphérique demande une interruption en modifiant le bon bit dans le registre IF. Lorsqu'une interruption est demandée, elle n'est pas forcemment traitée directement par le CPU. Tant que l'interruption n'est pas traitée, le flag d'interruption doit resté à 1. De plus, lorsque un bit du registe IF est mis à 1 par un périphérique que le même bit dans le registre IE est aussi à 1 et que le CPU exécute l'instruction EI, le CPU ne sautera pas à la routine d'interruption directement mais au prochain cycle d'horloge. A noter que cette particularité vaut pour l'instruction EI mais pas pour l'instruction RETI. Cette dernière retourne à la dernière adresse mise dans la pile mais active les interruptions au coup d'horloge actuel.

Exemple avec EI :
```
di
ld a, 4h
ld [FFFFh], a
ld [FF0Fh], a
ei
inc a // cette instruction va être exécutée avant la routine d'interruption
inc a // cette instruction va être exécutée après la routine d'interruption
```

Exemple avec RETI :
```
di
ld a, 4h
ld [FFFFh], a
ld [FF0Fh], a
reti
inc a // cette instruction va être exécutée après la routine d'interruption
```

```
inc a
```

3.4.3 Traiter une interruption

Nous avons pu voir qu'une interruption est traitée que si les bits correspondant sont actifs dans les registres IE et IF mais aussi que le "interrupt master" (IME) est à 1. La program counter est mis sur la pile puis il prend la valeur de l'adresse de la routine d'interruption demandée, IME est mis à 0 et le bit correspondant à l'interruption est mis à 0 dans le registre IF. Les routines d'interruption terminent normalement par une instruction RET ou RETI. Dans le cas de l'instruction RET, les interruption ne sont pas réactivées. Par contre, RETI les réactive (IME à 1) ce qui permet à une autre interruption d'être traitée (si par exemple un autre périphérique avait demandé une interruption en même temps que le premier). Les priorités de traitement d'une interruption est dans l'ordre du tableau ci-dessus. Le bit de poids faible a donc la plus grande priorité et ainsi de suite.

3.4.4 Comportement de HALT

L'instruction HALT est une instruction particulière car elle arrête le processeur jusqu'à qu'une interruption a lieu. Son comportement varie en fonction de l'état des registes IE, IF et du IME. L'instruction a en fait 3 comportements différents.

- IME = 1
  - L'instruction HALT est exécutée normalement, le CPU s'arrête jusqu'à ce que deux bits correspondants soient mis à 1 dans IE et IF. Une fois fait, le CPU repart et saute directement à la routine d'interruption.
- IME = 0
  - Si deux bits correspondants ne sont pas encore à 1 dans IE et IF, HALT s'exécute normalement, comme lorsque IME = 1 sauf qu'une fois le CPU reveillé, l'exécution du programme continue (sans routine d'interruption).
  - Si deux bits correspondants sont déjà à 1 dans IE et IF, le bug de HALT se produit. L'instruction HALT n'est pas exécutée et le processeur n'incrémente pas le program counter. L'instruction après l'instruction HALT est donc exécutée deux fois.

3.5 Timer

3.5.1 Timer controller

3.5.2 Divider register

3.6 GPU

3.6.1 Contrôleur LCD

3.6.2 Affichage

3.7 Entrées

4 Émulateur

5 Conclusion