Guide du développeur

Projet Frankenstein

**Par**:

Maxime Goyette

David Michel Donais

**23 Avril 2017**

Pour le cours de projet IFT592 et IFT692

# 

[**Introduction**](#_o01uqmwwcsla) **2**

[**Logiciels pour la compilation**](#_spijh3m8tbj1) **2**

[Essentiel](#_ovs6rqc0lxjj) 2

[Émulateur graphique](#_6lu5e9rsuzct) 2

[Émulateur Raspberry Pi](#_oi61la4hr2ve) 3

[Outils pratiques](#_k3ue7pywjou9) 3

[**Hiérarchie du projet**](#_1nft2zc1624f) **3**

[**Environnement de développement**](#_wz8wsaxfbls0) **4**

[**Compilation**](#_7ricx96zrsr9) **4**

[**Émulateur**](#_q20fw13565kp) **5**

[Classes](#_f6xxhoc3spxo) 5

[**Circle**](#_nltdr579d73a) **6**

[**Bibliographies et sources**](#_qzysir5d4ft5) **6**

# 

# 

# Introduction

Ce guide vise à introduire les connaissances nécessaire à la prise en charge du projet Frankenstein. Plus particulièrement, il sera décrit comment configurer et compiler correctement le projet; les technologies utilisés, autant dans les exécutables de tests que dans la partie embarquée; le domaine de l’émulation du Nintendo Entertainment System, ainsi que des liens vers la documentation appropriée; et finalement l’architecture et une description des composantes de l’émulateur. Ce guide n’est pas exhaustif, mais fournit une introduction suffisante pour reprendre efficacement le projet.

# Logiciels pour la compilation

## Essentiel

* Meson - version 0.36.0 ou supérieure
* Python - version 3.4 ou supérieure
* Ninja - version 1.5 ou supérieure
* Compilateur avec support de c++11
* Système d’exploitation GNU/Linux
* git

Ces logiciels sont nécessaire afin de pouvoir, minimalement, compiler la bibliothèque d’émulation. On parle ici d’une compilation native pour l’environnement de développement, sans exécutable ni image pour le Raspberry Pi.

Meson est un programme de méta-configuration. Comme CMake, il génère des scripts de compilation. Ce dernier dépend de Python, qui l’exécute, et de Ninja. Ce dernier remplace le célèbre make dans les scripts générés.

## Émulateur graphique

* Bibliothèque SFML-dev (version 2.4 ou supérieure)

SFML est une bibliothèque pour la conception multimédia, avec un support natif de l’affichage, des entrées (clavier ou autre), et du son. Elle est utilisée pour l’émulation sur l’environnement de développement, ce qui permet de déverminer les comportements de l’émulateur.

## Émulateur Raspberry Pi

* gcc-arm-none-eabi (testé avec la version 6-2017-q1)
* wget
* xxd

Le compilateur gcc-arm-none-eabi génère un binaire destiné à l’architecture ARM, avec la convention d’appel ARM embarqué.

L’utilitaire wget, quant à lui, est nécessaire pour télécharger les fichiers binaires démarrant le Raspberry Pi. Le dernier outil, xxd, est utilisé afin d’émettre les données d’une cassette numérisée en tableau compréhensible pour le c++. Cette étape est nécessaire seulement pour l’émulateur du Raspberry Pi.

# Compilation

L’outil utilisé pour la création des exécutables et de l’image pour Raspberry Pi 3 se nomme Meson. Meson est un moteur de production (« build management system ») relativement récent. Il est très semblable à l’outil CMake, mais avec une syntaxe DSL accessible et une documentation en ligne compréhensible. Il possède aussi la notion de compilation sur une plateforme différente de l’hôte, intégré directement au DSL. Ce trait permet de compiler l’image ARM pour le Raspberry Pi et les exécutable natifs, d’une seule commande.

La première étape pour configurer le projet Meson est de créer un dossier de compilation (build directory), ici nommé **test-build**, ainsi qu’un dossier d’installation, ici nommé **install-dir**. Le premier est utilisé pour la compilation des objets, des exécutables, et fichiers temporaires, alors que le second est le dossier de destination où se retrouveront tous les fichiers et exécutables généré lors de la compilation. On y retrouvera les exécutables pour l’environnement GNU/Linux ainsi que les images et fichiers pour le Raspberry Pi.

La commande pour générer les scripts de compilation est la suivante, à exécuter dans le répertoire racine du projet:

meson **test-build** --buildtype release --prefix /home/username/Frankenstein/**install-dir**/ --cross-file subprojects/circle/rpi3-cross.txt

Les paramètres importants sont:

* **Test-build**, le répertoire de compilation
* --buildtype release, qui spécifie une compilation pour la performance (-O3)
* --prefix /home/username/Frankenstein/**install-dir**/, le chemin absolu vers le répertoire d’installation désiré
* --cross-file subprojects/circle/rpi3-cross.txt, spécifie le document contenant les spécifications de la trans-compilation. C’est dans ce fichier que sont définis, entre autres, le compilateur ARM à utiliser.

Une fois généré, il suffit de se déplacer dans le répertoire de compilation et de lancer **ninja install**, qui compilera le projet entier et déplacera les exécutables et fichiers requis dans **install-dir**.

# Exécutable

Une fois compilé, il suffit de suivre les instructions du guide de l’utilisateur pour exécuter l’émulateur sur le Raspberry Pi. Si tout fonctionne, le jeu tel qu’installé dans rom\_static\_data.cpp devrait démarré. Si un problème a eu lieu au chargement de l’image, le Raspberry Pi restera sur un écran arc-en-ciel.

Il est aussi possible d’utiliser **term\_emulator** et **sfml\_emulator** pour valider le comportement de l’émulateur. Ces derniers génèrent une trace de l’exécution avec plusieurs informations utiles, tel que les valeurs des registres. Cela permet de suivre avec exactitude le déroulement de l’application, au défaut de générer plusieurs milliers de lignes de journal. Pour les démarrer, il suffit d’exécuter l’application, suivit d’un chemin vers une cassette numérisée. Si cette dernière est valide (c’est-à-dire, avec le Mapper 1), elle sera exécutée.

# Environnement de développement

Le développement et la compilation du projet ont été réalisé sur des systèmes d’exploitation GNU/Linux, plus précisément la distribution archlinux. Le projet devrait pouvoir être configuré sur un environnement similaire, pour peu que les dépendances soient suffisamment à jour.

Il est possible d’utiliser l’éditeur ou l’éditeur de développement intégré de son choix, tant que ce dernier supporte les projets où les commandes de compilation et d’exécution peuvent être spécifiées, ou encore les projets Meson, si disponible. D’expérience, il était facile de configurer le projet avec l’IDE *Netbeans*, dans sa dernière version, ainsi que *Gnome Builder* et *KDevelop*.

# Émulateur

## Hiérarchie du projet

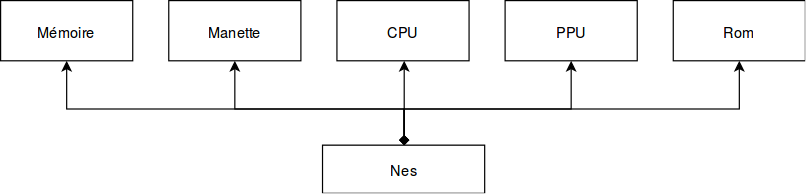
* **Frankenstein**
  + **application** 
    - Contient un émulateur graphique, **sfml\_emulator**, ainsi qu’un émulateur console, **term\_emulator**, utilisés pour le développement dans un environnement graphique GNU/Linux.
  + **emulator**
    - Contient les fichiers de développement de la bibliothèque de l’émulateur.
    - **include**
      * Contient les fichiers d’entête de la bibliothèque de l’émulateur.
    - **test**
      * Contient quelques tests unitaires pour l’unité central de traitement (CPU), utilisant la bibliothèque **gtest**.
      * **roms**
        + Contient des images de cassettes NES utilisées pour des tests d’intégrations.
  + **kernel**
    - Contient les fichiers de développement pour l’image s’exécutant sur le Raspberry Pi.
  + **Subproject**
    - Contient les bibliothèques externes utilisés dans le projet.
    - **Circle**
      * La bibliothèque Circle, par rsta2, est un mini système d’exploitation pour le Raspberry Pi.

## 

## 

## Classes

* Nes
  + Contient les instances de toutes les composantes et s’occupe de les initialiser. L’instance de Nes (this) est passé à chaque constructeur afin qu’une composante puisse accéder avec ses collègues. Par exemple, dans Cpu on peut accéder au Ppu en faisant nes->ppu.
* Memory
  + Classe qui gère les accès mémoire du CPU et les modes d’adressage des instructions simulées. Elle contrôle aussi les actions liées aux écritures et lectures aux adresses mémoires, tel que les communications aux autres composantes.
* Cpu
  + Exécute les instructions du NES à partir de la cassette numérisée (rom) copiée en mémoire pendant l’initialisation. Simule le comportement du CPU original et contient des variables équivalentes aux registres.
* Ppu
  + Picture Processing Unit. Sa tâche est de générer une image, 50-60 fois par seconde, et de l’afficher au téléviseur.
  + Effectue la routine logiquement équivalente à celle du PPU du NES. Le PPU peut recevoir des commandes de la part du CPU si certaines valeurs sont écrites dans des cellules mémoires spécifiques. La classe Memory s’occupe d’appeler les fonctions du PPU qui font le traitement approprié.
* Rom
  + Contient le programme et les données graphiques à simuler.
* Gamepad
  + Simule le comportement des lectures et des écritures aux adresses concernant les manettes : $4016 et $4017. Memory s’occupe de lier ce dernier au CPU.
* Mapper
  + Cette classe n’est pas utilisée. C’est une ébauche en prévision du désir d’implémenter plus de Mapper.



# Documentation

## Circle

* <https://github.com/rsta2/circle>

## Raspberry Pi 3

### Schémas

* <https://www.raspberrypi.org/documentation/hardware/raspberrypi/schematics/RPI-3B-V1_2-SCHEMATIC-REDUCED.pdf>

### Manuels

* <https://www.raspberrypi.org/documentation/hardware/computemodule/RPI-CM-DATASHEET-V1_0.pdf>
* <http://infocenter.arm.com/help/topic/com.arm.doc.ddi0500d/DDI0500D_cortex_a53_r0p2_trm.pdf>
* <https://people-mozilla.org/~sstangl/arm/AArch64-Reference-Manual.pdf>

### Wikis

* <http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.subset.cortexa.a53/index.html>
* <https://www.raspberrypi.org/documentation/hardware/raspberrypi/README.md>
* <https://www.raspberrypi.org/documentation/>
* <http://elinux.org/RPi_Hardware>
* <http://elinux.org/VideoCore_IV_3D_Architecture_Reference_Guide_errata>
* <https://github.com/raspberrypi/firmware/wiki>

### Guides

* <http://stackoverflow.com/questions/19162072/installing-raspberry-pi-cross-compiler/19269715#19269715>
* <http://www.valvers.com/open-software/raspberry-pi/step01-bare-metal-programming-in-cpt1/>
* <https://arobenko.gitbooks.io/bare_metal_cpp/content/index.html>
* <http://www.cl.cam.ac.uk/projects/raspberrypi/tutorials/os/>
* <https://rpidev.wordpress.com/>

### Forums

* <http://raspberrypi.stackexchange.com/questions/10442/what-is-the-boot-sequence>
* <https://www.raspberrypi.org/forums/viewtopic.php?f=72&t=33652>

### Outils

* <https://github.com/raspberrypi/tools/tree/master/arm-bcm2708/arm-rpi-4.9.3-linux-gnueabihf>
* <https://sourceforge.net/projects/raspberrypiemulator/>
* <http://maazl.de/project/vc4asm/doc/index.html>

### Exemples

* <https://github.com/PeterLemon/RaspberryPi>
* <https://github.com/dwelch67/raspberrypi>
* <https://github.com/rsta2/circle>
* <https://github.com/rsta2/uspi>
* <https://github.com/ICTeam28/PiFox>
* <https://github.com/SharpCoder/rpi-kernel>
* <https://github.com/Terminus-IMRC/qpu-trivial-assembler>
* <https://github.com/Terminus-IMRC/mailbox>
* <https://github.com/Terminus-IMRC/mailboxinfo>
* <https://github.com/Terminus-IMRC/libvc4v3d>
* <https://github.com/Terminus-IMRC/libvc4vec>
* <https://github.com/Terminus-IMRC/qpuinfo>
* <https://github.com/Terminus-IMRC/vpu_cpuid>
* <https://github.com/novellus/Bare-Metal-Raspberry-Pi-Kernel>

## NES

### Manuels

* <http://nesdev.com/NESDoc.pdf>
* <http://nesdev.com/NES%20emulator%20development%20guide.txt>
* <http://www.slack.net/~ant/nes-emu/apu_ref.txt>
* <http://www.6502.org/tutorials/6502opcodes.html>
* <http://nesdev.com/6502.txt>
* <https://gist.github.com/adamveld12/d0398717145a2c8dedab>
* <http://www.thealmightyguru.com/Games/Hacking/Wiki/index.php/6502_Opcodes>
* <http://web.archive.org/web/20160306180425/http://www.pagetable.com/?p=410>
* <http://web.textfiles.com/games/nestech.txt>

### Wikis

* <https://en.m.wikipedia.org/wiki/MOS_Technology_6502>
* <http://wiki.nesdev.com/w/index.php/NES_reference_guide>
* <http://wiki.nesdev.com/w/index.php/Nesdev_Wiki>

### Guides

* <https://projects.drogon.net/nes-controller-on-the-raspberry-pi/>
* <http://blog.alexanderdickson.com/javascript-nes-emulator-part-1>
* <https://opcode-defined.quora.com/How-NES-Graphics-Work-Pattern-tables>
* <http://nesdev.com/NESTechFAQ.htm>
* <http://www.dustmop.io/blog/2015/04/28/nes-graphics-part-1/>

### Vidéos

* <https://www.youtube.com/watch?v=NLEMsw1SjDY>

### Exemples

* <https://github.com/fogleman/nes>