ENSEIRB-MATMECA
Bordeaux-INP

2A Électronique
Projet numérique
EN202

Pokémon Silicium
Compte-rendu

Élèves :
Maxime LETEMPLE
Samuel BARIS

Enseignant :
Christophe JÉGO

19 janvier 2023

Table des matières

Table des figures

1 Introduction

L'objectif de ce projet a été de créer un jeu de combat au tour par tour inspiré de Pokemon et d'implémenter son architecture sur une carte FPGA NEXYS 4. L'affichage se fera sur un écran grace à un module compleméntaire implémentant un connecteur VGA et la navigation dans le jeu se fera à l'aide d'un joystick, lui aussi ajouté en tant que module complémentaire.

2 Cahier des charges

Nous nous interressons donc aux phases de combat du jeux pokemon. Ces phases sont des combats au tour par tour où deux dresseurs s'affrontent à l'aide de plusieurs pokemon. Les pokemon ont accès à plusieurs attaques qu'ils peuvent utilisés afin de baisser la barre de vie du pokemon adverse. Un pokemon dont la barre de vie atteint 0 est "mort" et ne peut donc plus être utilisé. La particularité du jeu pokemon est que chaque pokemon et chacune de leurs attaques ont des statistiques différentes et un type (les pokemon peuvent avoir deux types mais les attaques ne peuvent avoir qu'un type). Ces types caractérisent les forces et les faiblesses des pokemon et des attaques vis-à-vis des autres types, organisées comme le pierre-papier-ciseaux pour équilibrer les combats.

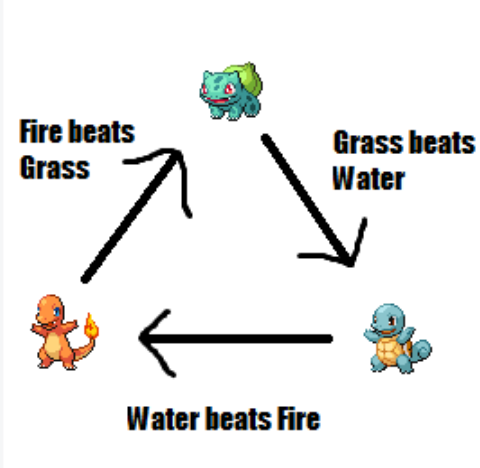

FIGURE 1 – Exemple d'un triangle faiblesse/force des types

Le cahier des charges des différentes fonctionnalités que nous voulions intégrer à notre achitecture est le suivant :

- Affichage du jeu sur un écran VGA
- Utilisation d'un joystick pour naviguer dans les menus
- Possibilité de jouer à deux, chacun son tour, quatre pokemon par joueur
- Prise en compte de la table de type qui régit les forces et faiblesses de chaque pokemon
- Possibilité pour chaque joueur d'attaquer ou de changer de pokemon à chaque tour

3 Décomposition de l'architecture

Ce projet est décomposé en plusieurs grand blocs pour simplifier son apréhension. Dans un premier temps, le module gest_buttons traite les signaux reçus du joystick pour les envoyer dans le module game_logic. Ce module se charge de toute la logique de jeu : il gère la boucle du jeu(ordre dans lequel les actions sont effectuées), le calcul de la vie des pokémons et de leurs attaques. Les signaux nécessaire à l'affichage sont ensuite envoyés à l'interface jeu-affichage qui traduit l'état dans lequel le jeu est (vie des pokémons, état du jeu, pokémons sélectionnés, etc. Cette interface traduit ces signaux en addresses qui sont envoyées au module d'affichage, qui ensuite modifie la bitmap contenue dans l'interface VGA trouvée sur le site de Yannick Bornat, qui se charge de l'affichage.

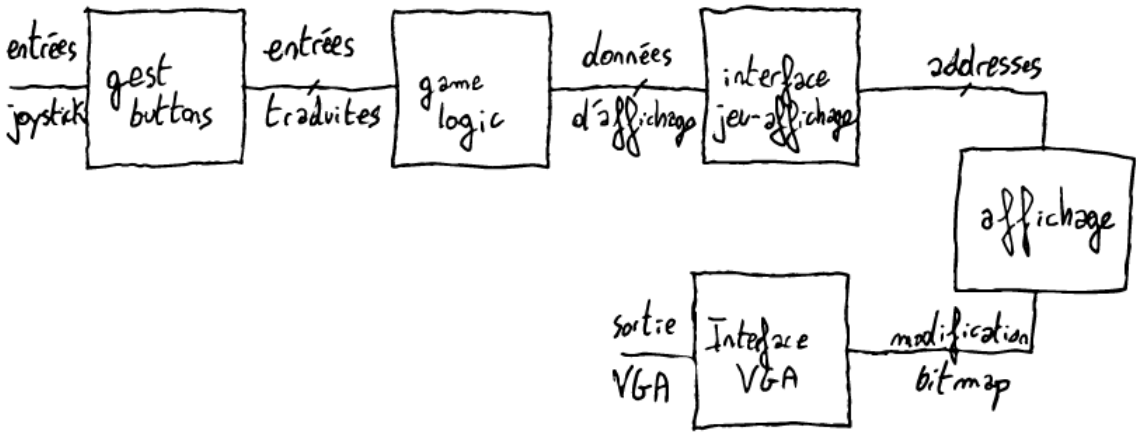

FIGURE 2 – Décomposition globale de l'architecture

4 Bloc d'entrée du module joystick Digilent

Le module du joystick (nommé PmodjSTK) a été repris tel quel des fichiers de démo sur la page du constructeur. Celui-ci récupère les données envoyées par le PMOD via une liaison série et les envoie dans un vecteur de 40 bits **data_joystick**. Ces données sont envoyées dans le module gest_buttons, qui en sortie envoie les signaux **left**, **right**, **up**, **down**, **fwd**, **bwd**. Les signaux directionnels sont obtenus en comparant les valeurs X et Y, qui sont les images de la tension des potentiomètres du joystick. Les signaux fwd et bwd sont issus des boutons BTN1 et BTN2 du module, mais sont traités pour n'avoir la valeur 1 que pendant un coup d'horloge après appui(via une FSM).

5 Bloc d'affichage

Le bloc d'affichage est constitué d'une FSM (gest_display) qui contrôle plusieurs modules d'affichage. Selon la valeur du signal **type_selector**, des multiplexeurs contrôlent la valeur de la sortie (donnée et adresse). Par exemple, si la FSM a envoyé l'addresse du pokémon allié, **type_selector** prend la valeur "001", et **data_out** prend la valeur de **data_out_pokemon**, idem pour **addr_display**.

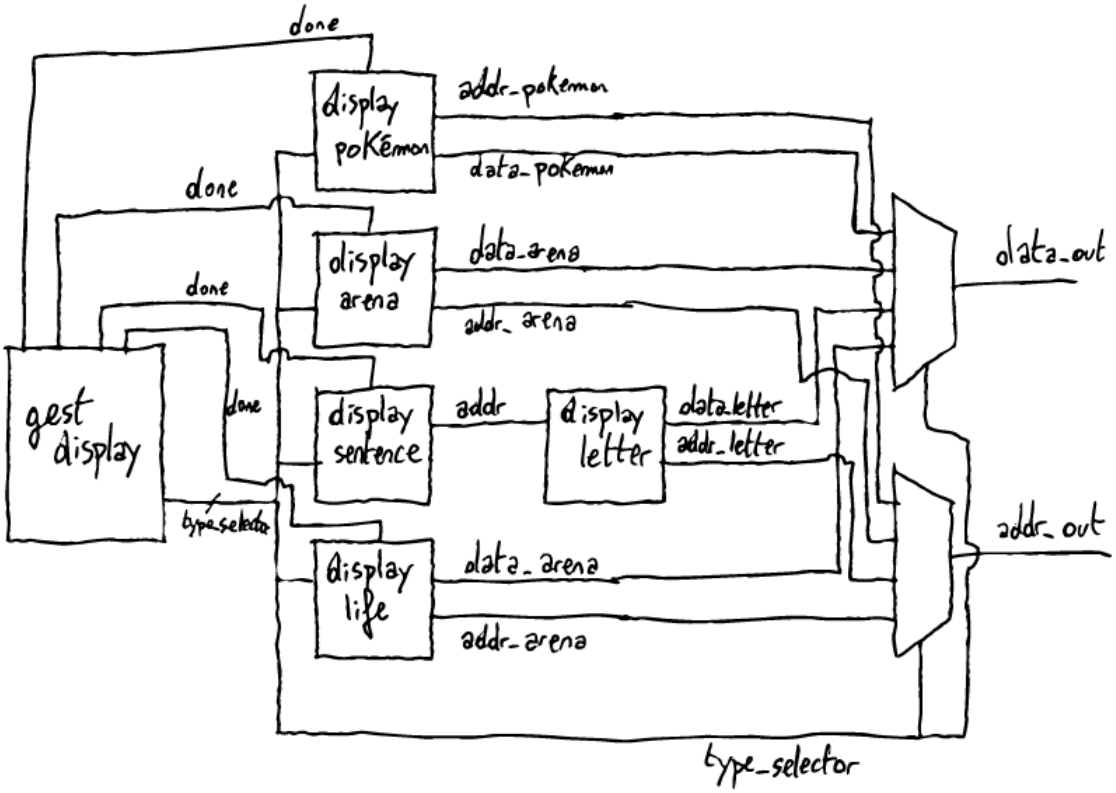

FIGURE 3 – Décomposition globale de l'architecture

5.1 FSM globale de l'affichage

La FSM est constituée de deux états pour chaque élément affiché. Au départ, elle attend un signal de **clock_enable**(pour s'assurer que l'affichage se fait à 30 images par secondes), puis elle commence à afficher chaque élément. Pour chacun, le signal **type_selector** prend la valeur nécessaire, puis les signaux nécessaires sont mis à jour (addresse de départ de l'affichage et addresse de départ dans la ROM des données).

5.2 Modules d'affichage de pokémons et de l'arène

Les modules d'affichage des pokémons et de l'arène sont sensiblement les mêmes. Pour les pokémons, la ROM contient des images de 64*64 pixels de 8 bits. l'affichage se fait par des compteurs de lignes et de colonnes. Il ne faut pas oublier d'ajouter 256, soit la largeur de l'écran à la fin de chaque ligne. Une fois que l'affichage est terminé le signal **done** passe à 1 pour avertir la FSM.
Le module d'affichage des pokémons doit gérer la transparence. La sortie de la ROM n'est donc pas la sortie du module. Il y a entre les deux un multiplexeur. Quand le pixel est vert ("00011100"), celui-ci est envoyé sur une addresse en dehors de l'écran. Sinon, il est envoyé à la bonne adresse.

5.3 Modules d'affichage des lettres

L'affichage des lettres est plus sensible que l'affichage d'images. Les phrases sont enregistrée dans la ROM du module **display_sentence**. Cette ROM contient les addresses

des lettres contenues dans la ROM du module **display_letter**. **display_sentence** se charge d'envoyer le signal à **display_letter** d'afficher les lettres une par une. La communication entre ces deux module se fait par le signal **letter_done**. Quand celui-ci vaut 1, la FSM de display_sentence passe à la lettre suivante, et ce jusqu'à ce que le compteur de lettres soit égal au signal **size_sentence**.

5.4 Modules d'affichage de la vie

L'affichage de la vie se fait par l'affichage de l'image(de la même manière que pour les pokémons et l'arène), puis de la barre de vie. La vie maximale de tous les pokémons est de 100 PV, et la longueur de la barre de vie est de 100 pixels. De cette manière, il est plus simple de l'afficher. Il suffit d'ajouter un compteur, qui dessine un pixel rouge au bon emplacement jusqu'à ce que le compteur soit égal à la vie actuelle du pokémon.

6 Architecture de la logique de jeu

6.1 Schéma et description globale de l'architecture

L'architecture de la partie logique de jeu est la suivante :

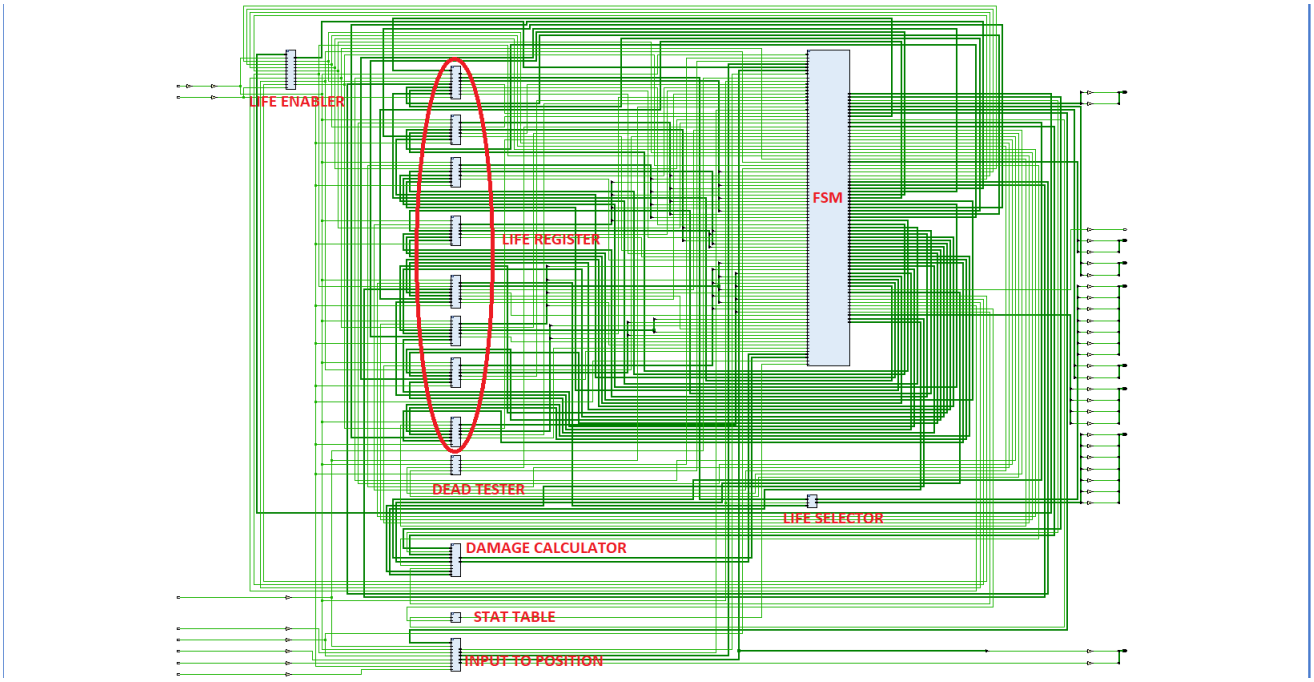

FIGURE 4 – Architecture de la logique de jeu

6.2 Description des blocs

6.2.1 Input to position

Ce bloc est une FSM qui va transmettre en sortie un vecteur de 2 bits indiquant la position du menu à laquelle on se trouve (en haut à gauche, en bas à droite ...). Les changements d'états de la FSM et donc du vecteur position se font en fonction de la

direction indiquer en entrée. Si on est en haut a gauche et que l'entrée up est à '1', il ne se passe rien car on est deja en haut. Si c'est l'entrée droite qui est à '1' alors la position change et on se trouve en haut à droite. La sortie position est relié à la FSM menu et va permettre la navigation dans les menus.

6.2.2 FSM_menu

Cette FSM est le coeur de la logique de jeu, elle contient 12 états qui correspondent chacun à une phase de jeu. Le vecteur stage prend une valeur différente à chaque états et va permettre au reste de l'architecture de savoir dans quelle phase de jeu nous sommes. La progression dans la FSM se fera principalement à l'aide des boutons fwd et bwd qui permettent respectivement d'avancer ou de revenir en arrière dans les menus mais aussi avec le std dead qui indique si un pokemon est mort sur le terrain. Les sorties de la FSM sont en plus de stage, le std_logic player qui indique quelle joueur joue, les vecteurs poke1 et poke2 qui indiquent quel pokemon est sur le terrain pour chaque joueur et le vecteur atk qui indique quelle attaque à été choisit par l'utilisateur qui joue.

Voici comment est décrite la FSM :

— On commence à l'état INIT où le joueur 1 joue et où les pokemon sur le terrain son ceux de base (identifié par "00"). En pressant fwd on passe à l'état suivant.

— On est alors à l'état menu. En pressant fwd on passe à l'état suivant qui va dépendre de l'entrée position : les états bag et run sont des références au jeux pokemon mais ne sont pas implémenté et nous ramène directement au menu. Les deux "vrais" états sont donc attack_menu et pokemon_menu.

— Attack_menu : Etat où ce fait le choix de l'attaque envoyé. En pressant bwd on retourne au menu. En pressant fwd on passe à l'état suivant attack_text.

— Attack_text : La sortie atk prend la valeur de 'position' au moment où on a appuyé fwd. En pressant de nouveau fwd on passe à l'état suivant damage_calcul.

— Damage_calcul : Etat où ce fait le calcul des dégats. On passe sans condition à l'état life_calcul (cf module Damage calculator).

— Life_calcul : Etat où ce fait le calcul de la vie du pokemon ayant subit l'attaque. On passe sans condition à l'état attack_efficiency (cf module Enable life et Life register)

— Attack_efficiency : Etat utile à l'affichage mais il ne se passe rien ici. En pressant fwd on passe à l'état suivant switch.

— Switch : Etat où on change de tour, player prend son inverse pour valeur, c'est à l'autre joueur de jouer. Le tour suivant commence au menu si il n'y a pas de pokemon mort sur le terrain (std dead à '0'). Si il y a un pokemon mort, c'est celui du joueur qui vient de subir l'attaque, donc celui qui s'apprete à jouer. Avant d'aller au menu il doit donc passer par l'état pokemon_menu afin de changer de pokemon. Un signal interne newturn se met à '1' afin d'indiquer que le changement de pokemon fait par le joueur n'est pas un choix d'action fait pendant son tour mais à été imposé par le fait que son pokemon soit mort. Le joueur pourra donc effectuer son tour après avoir changé de pokemon.

— Pokemon_menu : Etat où ce fait le choix du pokemon à positionner sur le terrian. En pressant bwd on retourne au menu si il n'y a pas de pokemon mort sur le terrain.

Si il y en a un, on est obligé de changer de pokemon. En pressant fwd on passe à l'état suivant pokemon_switch.

— Pokemon_menu : La sortie poke1 ou poke2 (suivant le joueur qui est en train de jouer) prend la valeur de 'position' au moment où on a appuyé fwd. En pressant de nouveau fwd on passe à l'état suivant dead_test.

— Dead_test : Dans cette état, on regarde si le pokemon qui vient d'être mis sur le terrain est mort ou non (std dead). Si il est mort alors on retourne à l'état pokemon_menu afin de choisir un autre pokemon. Si il n'est pas mort le jeu peut poursuivre. On teste alors la valeur de newturn (cf état Switch) pour savoir si l'état suivant est menu (on est alors au début du tour) ou switch (Fin du tour).

6.2.3 Stat table

Ce bloc asynchrone est celui qui stocke toute les stats et les types de chaques pokemon et de leurs attaques. Le bloc renvoit en sortie la stat de defense du pokemon subissant l'attaque, ses types, la stat d'attaque de l'attaquant et de son attaque, le type de son attaque et si son attaque est stabé (Si l'attaque est du même type que le pokemon qui la lance celle ci est plus puissante). Le choix de ses parametres se fait en fonction du joueur qui est en train de jouer, des pokemons sur le terrain (vecteur de 2 bits poke1 et poke2) et de l'attaque lancé (vecteur de 2 bits atk correspondant à la valeur du vecteur position au moment du choix de l'attaque, cf FSM).

6.2.4 Type table

Ce bloc asynchrone transpose dans notre architecture la table des types suivantes représentant les forces et faiblesses de chaque pokemon :

FIGURE 5 – Table des types

Le module recoit le type de l'attaque et le type du defenseur, effectue une disjonction de cas pour chaque possibilité possible et attribue une valeur au vecteur de 2 bits type_mult qui va permettre dans le module de calcul des degats de savoir si un multiplicateur doit être appliqué. Le defenseur pouvant avoir deux types, il y a deux instances de ce bloc dans notre architecture.

6.2.5 Damage calculator

Comme son nom l'indique, c'est dans ce module que se fait le calcul des dégats lors de la phase jeu dédié à cette action. Le module récupère les stats de defense et d'attaque provenant du module "Stat table" (des unsigned) et effectue le calcul suivant :

$damage\_temp1 <= used\_atk\_stat + atk\_stat - def\_stat$

On va ensuite appliquer à ce signal des multiplicateurs à l'aide des fonctions shift_right et shift_left. Les multiplicateurs sont ceux de la table de type vu precedemment qui se cumule si le pokemon qui subit à plusieurs types et un multiplicateur x1.5 si l'attaque est stabbé (cf Stat table).

6.2.6 Enable life

Ce module met à '1' un de ses 8 std_logic enable de sortie qui vont permettre de soustraire les degats calculés precedemment à la vie du pokemon subissant l'attaque. Le choix du enable à mettre à '1' se fait suivant les pokemons sur le terrain ainsi que suivant le joueur qui joue (On met à '1' le enable du pokemon du joueur qui ne joue pas, qui subit l'attaque). Ce module contient une FSM pour s'assurer que le signal enable ne reste à '1' que durant une periode d'horloge, lors de la phase de jeu "life_calcul" (cf FSM_menu).

6.2.7 Life register

Il y a un registre de vie par pokemon (8) et la vie de chaque pokemon est initialisée à 100 (unsigned). Si le signal enable est à '1' lors d'un front d'horloge, on compare la valeur des degats subit par le pokemon avec sa vie, si les degats sont inferieur à la vie alors on recalcule la vie en soustrayant les degats subis. Si le pokemon subit plus de degat qu'il n'a de vie alors on affecte 0 à sa vie et la valeur du std_logic de sortie dead passe à '1', indiquant que le pokemon est mort et qu'il ne pourra plus être utilisé lors du combat.

6.2.8 Dead tester

Ce module récupère les valeurs des variables dead de chaque registre et, à l'aide des variables poke1 et poke2, regarde si il y a un pokemon mort actuellement sur le terrain. Si il y en a un alors la sortie dead du module est mise à 1. Cette sortie dead est celle utilisée par la FSM.

6.2.9 Life selector

Ce module récupère les valeurs de vie de chaque registre et, à l'aide des variables poke1 et poke2, renvoie en sortie la vie des deux pokemons présent sur le terrain. La vie va être transmise au bloc d'affichage.

7 Bloc d'interface entre la logique de jeu et l'affichage

L'interface entre la logique et l'affichage se fait par un module asynchrone. Celui-ci prend en entrée la position du curseur, l'efficacité de l'attaque, la vie des pokémons, le pokémon choisi, l'attaque choisie et le joueur en cours. C'est dans ce module que sont gérées les adresses de départ des pokémons dans les ROM, ainsi que les adresses de départ des mots à afficher.