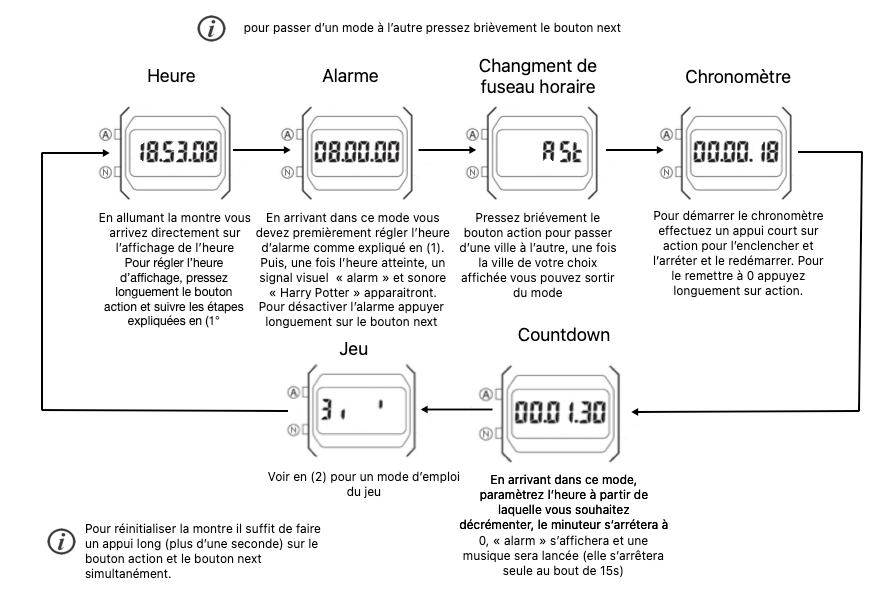
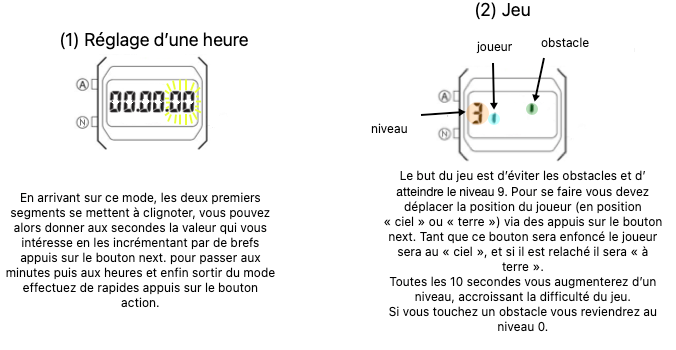
**RAPPORT MONTRE DE10-LITE**

# **Description Générale**

Dans le cadre de projet, nous avons dû réaliser une montre, celle-ci inclut les fonctionnalités suivantes : heure, alarme (avec génération de musique), changement de fuseau horaire, chronomètre, minuteur, et même un jeu. Pour ce-faire nous utilisons les deux boutons présents sur la carte ainsi que les 6 afficheurs 7-Segments et leurs points. Enfin, les LEDS indiquent dans quel mode l’utilisateur se trouve et pour la musique nous utilisons un buzzer. Nous considérons le bouton se situant le plus au nord de la carte comme le bouton « next » , et celui d’en-dessous comme le bouton « action ».

Nous distinguerons ici les appuis « longs » (plus d’une seconde) des appuis « courts » (moins d’une seconde), sur les boutons.

# Mode d’emploi



# 

# Solutions techniques apportées

# Définition des termes et nomenclature

Nous avons choisi une vitesse d’horloge de 200 KHz (voir partie musique).

Nous avons défini une nomenclature pour les noms des modules et les noms des entrées sorties très strictes :

nom module : nomDeLaFonction\_nomDuModule, nom d’une entrée : nomDeLentréeOuSorti

# FSM Générale

Notre FSM est codée sur 4 bits, chaque état étant consistant et complet, le reste d’états disponibles ont été considérés comme des don’t care, une montre n’étant pas un système vital.

|  |  |
| --- | --- |
| State | Code |
| hour | 0000 |
| setHour | 0001 |
| alarm | 0010 |
| setAlarm | 0011 |
| setTimeZone | 0100 |
| chronometer | 0101 |
| countdown | 0110 |
| setCountDown | 0111 |
| game | 1000 |

Notre FSM suit le diagramme d’état suivant :

Figure 1 Code Table

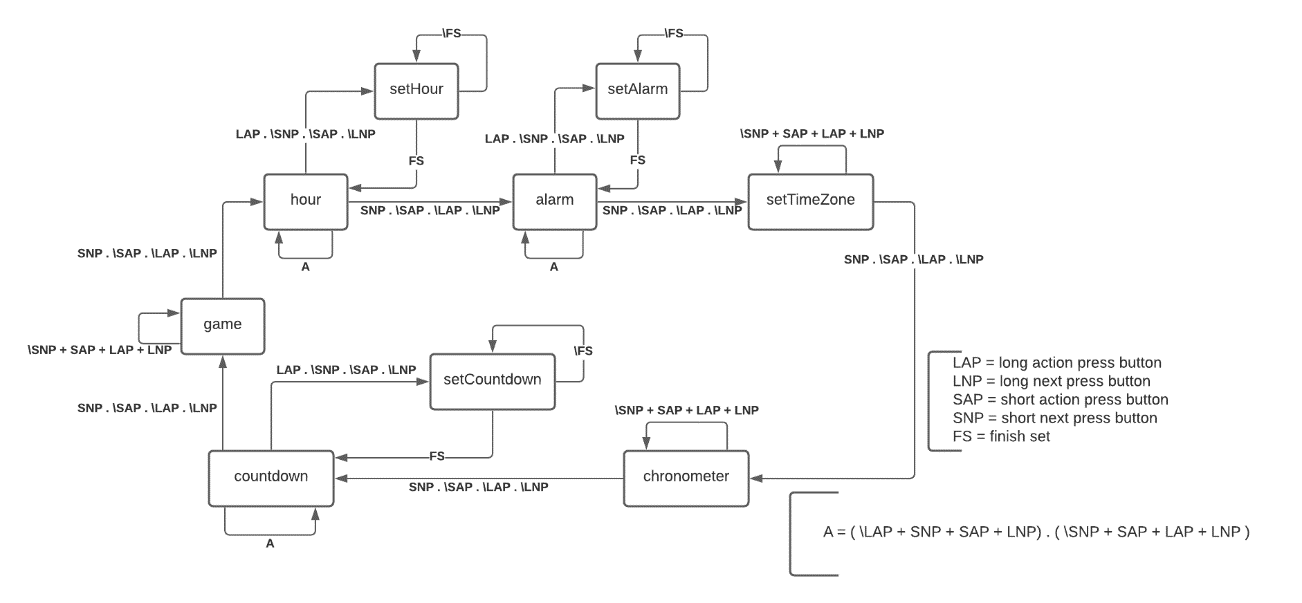


Figure 2 Diagramme d'état de la FSM générale

# L’interface avec l’utilisateur

Dans la donnée, il nous est demandé de minimiser le nombre d’éléments physique qui peuvent interagir avec l’utilisateur. Nous avons donc choisi d’utiliser seulement les deux boutons de la carte DE10-lite.

Pour pouvoir interagir correctement avec la montre, chaque bouton possède deux états : un appuie court et un appuie long. Nous avons ainsi 5 combinaisons possibles (bouton action court, bouton next court, bouton action long, bouton next long, bouton action long et next long).

Pour le reset de la montre, nous avons choisi d’utiliser la combinaison la moins évidente pour éviter les resets accidentels, nous avons donc assigné buton action long et next long à cet effet.

Nous n’avons pas choisi d’intégrer de debouncer car il y a déjà un dans la carte (voir en annexe).

# Musique

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Note | Fréquence | Période (ms) | Nb tick | Nb tick haut |
| A | 550 | 1.82 | 364 | 182 |
| AS | 582 | 1.72 | 344 | 172 |
| B | 617 | 1.62 | 324 | 162 |
| C | 654 | 1.53 | 306 | 153 |
| CS | 692 | 1.45 | 290 | 145 |
| D | 734 | 1.36 | 272 | 136 |
| DS | 777 | 1.29 | 258 | 129 |
| E | 824 | 1.21 | 242 | 121 |
| F | 873 | 1.15 | 230 | 115 |
| FS | 925 | 1.08 | 216 | 108 |
| G | 980 | 1.02 | 204 | 102 |
| GS | 1003 | 1.00 | 200 | 100 |
| A2 | 1100 | 0.91 | 182 | 91 |
| A2S | 1165 | 0.86 | 172 | 86 |
| B2 | 1234 | 0.81 | 162 | 81 |

Nous avons choisi de générer les fréquences avec une précision de 1e-5 sec sur leur période.

On veut pouvoir incrémenter par 1e-5 sec mais comme on doit, dans Logisim, définir les temps haut et bas des horloges, il faut doubler la fréquence d’horloge calculée.

Nous avons choisi un tempo de 200ms afin de n’obtenir que des nombres entiers pour la durée de chaque note.

Nous avons aussi créé une deuxième musique pour le jeu qui est le son du levelup du jeu mario bros.

Figure 3 Nb tick pour générer une frequence

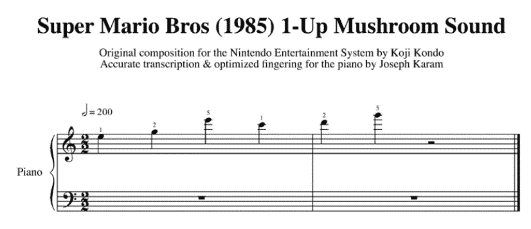
La fréquence des noires est de 400 bpm soit 6.6hz (6.6 noire par seconde) donc la durée d’une noire est de 30 000 ticks selon notre vitesse d’horloge de 200 KHz.

Figure 4 Partition du son « level up »

# SuperAdder et SuperSubtracter

Pour notre projet, nous avons besoin d’un compteur. Nous avons choisi de réaliser un compteur très général pouvant s’adapter à de nombreux cas rencontrés dans la conception du projet.

L’entrée « lastNumber » est ce jusqu’à combien le compteur va compter.

« SetBits » et « setSignal » permet de mettre le compteur à une valeur désirée.

L’entrée « stopEnd » permet si elle est au logique haut, d’arrêter le compteur de compter dès qu’il est à la valeur « lastNumber ».

Figure 5 Vue externe du compteur superAdder

Le prix à payer pour ce compteur polyvalent est que pour compter de 0 à 15 (ou moins), on utilisera tout de même 5 DFF alors que la solution optimale en utiliserai 4 (ou moins).

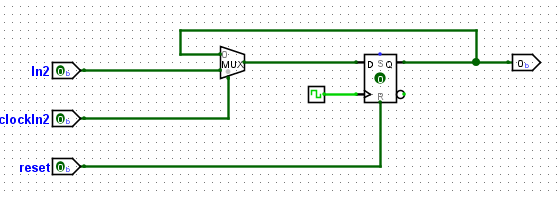
L’entrée clockIn a besoin d’une clock en entrée avec un temps haut d’exactement 2 ticks. Nous avons décider de toujours mettre la clock la plus rapide (1 tick haut 1 tick bas) sur l’entrée clock de toute nos DFF du projet afin d’éviter des bugs. Nous devons donc utiliser un MUX contrôlé en partie par l’entrée clockIn. C’est pourquoi il faut un temps haut de 2 ticks pour garantir que le signal de clockIn soit compté une fois par notre clock sur la DFF comme le montre le schéma simplifié ci-contre.

Figure 6 schéma simplifié d'une DFF dans son ensemble

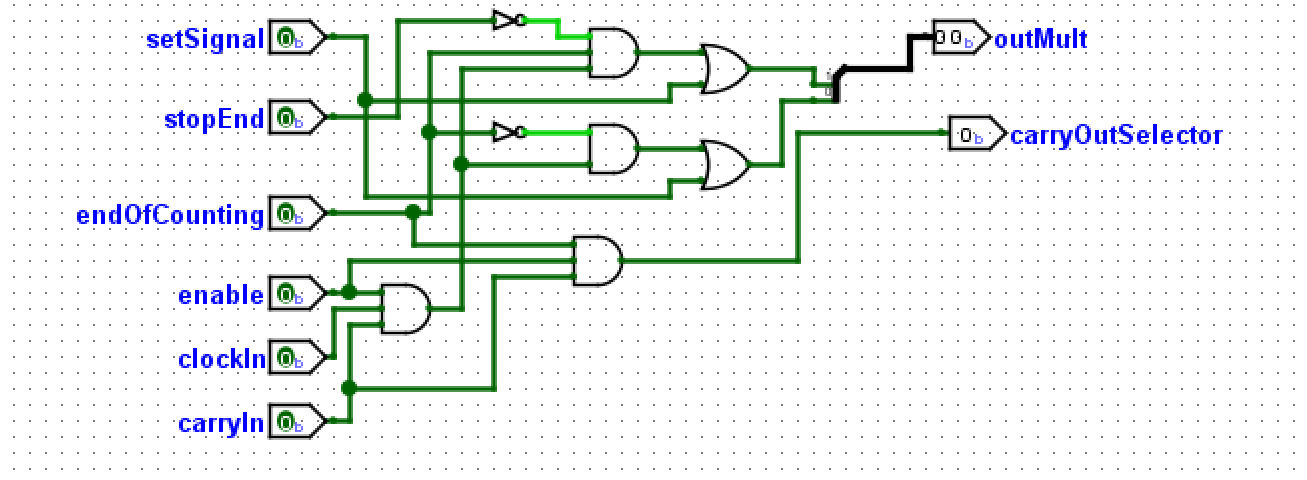
Fonctionnement du superAdder de transition : pour émettre un « carryOutSelector » si la valeur additionnée est supérieure à celle de « last number » alors nous émettons un « carryOut » à 1, permettant ainsi d’incrémenter les superAdders suivants.

Figure 7 superAdderTransition

Notre superAdder est contrôlé par un multiplexer, lui-même commandé par la sortie outMult du superAdderTransition, Le multiplexer nous permet de, soit de retransmettre le même nombre, soit le même nombre incrémenté de 1, soit 0, soit une variable initiale (par exemple lorsque nous paramétrons l’heure dans Time Setting et que nous la transmettons à l’heure actuelle). Tous ces états sont encodés par le circuit de la Figure 7.

Le module SuperSubtractor est très similaire au SuperAdder, nous n’avons pas besoin de le détailler.

# LogicEdge

Cet élément nous permet de détecter un front montant ou descendent d’un signal. Ce dernier émet par la suite un signal d’exactement 2 ticks de durée. Comme pour nos FSM nous avons une clock de période 2 ticks (1 tick haut, 1 tick bas), si nous mettons la sortie de notre logicEdge à l’entrée de notre FSM et comme la durée de notre signal est de 2 ticks, cela nous garantit exactement 1 front montant de la clock durant le niveau haut du signal. Le signal sera donc pris en compte une fois par la FSM comme vous pouvez le voir sur la figure 7.

Figure 8 Chronogramme de l’utilisation du module LogicEdge

# Display

Le module Display nous sert à convertir l’information que l’on veut afficher sur les 6 afficheurs 7-segments. Ainsi nous avons choisi d’utiliser 5 bits d’encodage nous permettant de représenter 32 chiffres et lettres. En plus de ces 5 bits chaque par 7-segment, nous fournissons aussi une valeur enable, nous permettant de choisir si l’on veut ou non un affichage, et une variable no-convert (pour l’affichage du jeu) gérée à l’aide d’un multiplexer. L’encodage des chiffres va de 00000 correspondant à 0 jusqu’à 01001 valant 9, pour les lettres nous avons 01010 valant « a » jusqu’à 11111 valant « y ». Il est à noter que j et x ne sont pas représentés (en effet se sont les deux lettres les moins utilisées dans l’alphabet anglais) et s et z sont représentés comme des 2 et 5.

# Time Setting

Cette fonction de réglage d’une heure donnée nous sert dans plusieurs des fonctionnalités de la montre. Pour ce faire nous utilisons nos SuperAdders pour incrémenter l’heure que l’on veut fixer. une fois le paramétrage de l’heure souhaitée terminée, nous mettons notre variable « finish » à 1 durant 2 ticks, cette variable nous servira par la suite comme feu vert pour mettre à jour l’heure choisi en entrée des aux autres modules.

# Module supplémentaire

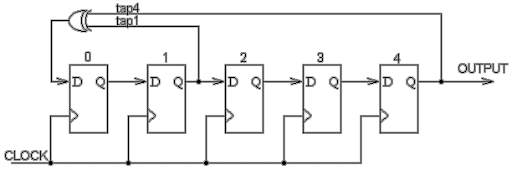
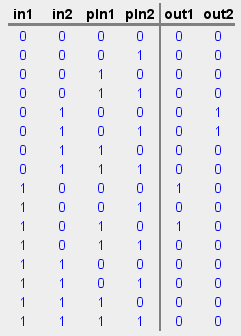
En fonctionnalité bonus, nous avons décider d’ajouter un jeu à notre montre. Le jeu consiste en un joueur qui doit sauter pour éviter des obstacles arrivant de la droite l’écran. Le but du jeu est d’arriver et de rester au niveau 9 le plus longtemps possible.

Figure Logic gate random number generator

Pour cela nous avons dû créer un générateur d’aléatoire pour choisir la position de l’obstacle. Le générateur est basé sur le schéma logique ci-contre qui permet de générer une suite de nombre pseudo-aléatoire. Lors du reset ou du Game over, on reset les DFF de manière identique (seulement une DFF est mise à 1 et le reste à 0) afin d’obtenir la même suite de nombre pseudo aléatoire et donc la même difficulté dans le jeu.

Nous utilisons deux générateurs (un pour le haut l’autre pour le bas) et avons dû mettre en place un filtre pour que toutes les combinaisons générées soit rendues jouable (pas de mur complet ou le joueur serait obligé de perdre). Le filtre suit la table de vérité ci-contre. int1 , int2 représentent les sorties des deux générateurs (haut et bas) et pin1 et pin2 représentent l’état précèdent validé par le filtre afin d’éviter de mauvaises combinaisons (voir figure). Si le joueur se trouve à l’endroit où se situe un obstacle, le jeu redémarre à 0.

Figure 10 Table de vérité du filtre

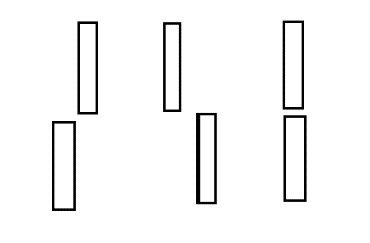
La difficulté du jeu augmente progressivement, le nombre d’obstacle généré par seconde et la vitesse de d’affilement augmente en même temps.

Figure 11 Combinaison que le filtre doit écarter