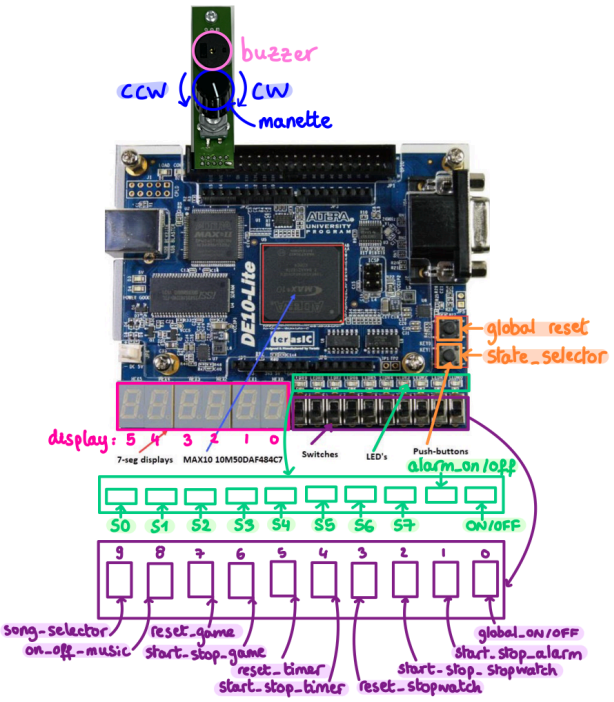
Systèmes logiques - rapport du projet

Le projet consiste à réaliser un réveil-matin digital ayant de multiples fonctionnalités complexes au moyen du logiciel **logisim-evolution**. Pour cela, une carte périphérique nous a été mise à disposition afin d'exécuter ces fonctions, notamment à travers un buzzer, une molette, des boutons-poussoirs, des DIP-Switches, 10 LED et 6 afficheurs à 7-segments.

Au moyen de fonctions logiques, le logiciel **logisim-evolution** nous a permis de créer une montre fonctionnelle et de la personnaliser avec l'ajout de fonctionnalités complexes et originales. Les fonctionnalités proposées sont les suivantes :

- l'heure (state 1, **S1** : normal watch mode), son affichage et son ajustement (state 0, S0 : time setting)
- un chronomètre (state 2, **S2** : stopwatch)
- un timer (state 3, **S3** : timer)
- l'alarme et sa sonnerie (state 4, **S4** : alarm)
- un jeu visuel (state 5, **S5** : Jumping Game)
- de la musique (state 6, **S6** : music)
- un jeu d'estimation de temps (state 7, **S7** : 10 Seconds Game)



Le Mode d'Emploi

Pour allumer ou éteindre le fonctionnement de la montre : utilisez le DIP-Switch 'Global_on_off'. La LED ON/OFF indique si la montre est allumée ou pas.

Pour reset tous les états : appuyer sur le bouton 'Global_reset'. Il est recommandé de reset la montre après le premier chargement de la carte périphérique.

Pour passer d'un état à un autre : Appuyer sur le bouton 'State_selector' pour choisir l'état désiré, la LED (S0 à S7) correspondante s'allume.

Pour l'horloge : Sélectionner le state S0 et utiliser la molette pour obtenir l'heure désiré (tourner dans le sens clockwise CW pour incrémenter, counter-clockwise CCW pour décrémenter). Sélectionner S1 puis switch 'Global_ON/OFF' pour démarrer l'horloge. L'heure s'affiche |heure|min|sec| (displays |5-4|3-2|1-0|).

Pour le chronomètre : Sélectionner S2 et utiliser 'start_stop_stopwatch' pour lancer ou arrêter le chronomètre. Utiliser 'reset_stopwatch' pour le réinitialiser. L'affichage est |min|sec|10e et 100e de sec| (displays |5-4|3-2|1-0|).

Pour le timer : Sélectionner S3, utiliser la molette pour set le timer et appuyer verticalement sur la molette pour le confirmer. Pour lancer ou arrêter le timer, switcher 'start_stop_timer'. Switcher 'reset-timer' pour réinitialiser le timer. Le timer s'affiche comme l'heure et décrémente à partir de la durée de timer choisie. Il sonne à 00h00min00s.

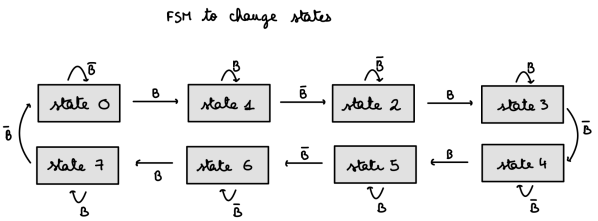
Pour l'alarme : Sélectionner S4 et utiliser la molette pour set l'alarme (CW pour incrémenter, CCW pour décrémenter). Basculer 'start_stop_alarm' pour allumer ou éteindre l'alarme (pour cela, il faut être dans S4). Même si l'on n'est pas dans le mode Alarm, l'alarme reste activée, la LED 'alarm_on/off' indiquant si l'alarme est activée ou pas. L'affichage est |heure|mins|sec|.

Pour jouer au jeu visuel : Sélectionner S5 et lancer ou arrêter le jeu en basculant 'start_stop_game'. C'est un jeu de sauts d'obstacles : grâce à la molette dans le sens CW ou CCW, le joueur manipule le mouvement vertical d'une boule (au display 5) pour éviter des obstacles. Ils apparaissent aléatoirement et se déplacent de droite à gauche en continu, chacun laissant une trace derrière lui avant le prochain objet. Si un objet atteint le joueur, c'est perdu (affichage GAME OVER)!. Basculer 'reset_game' pour réinitialiser le jeu.

Pour écouter de la musique : Sélectionner S6 et choisir une de deux chansons en switchant 'song_selector'. Pour arrêter la chanson, basculer 'on_off_music' (elle recommence une fois relancée). Pour chaque chanson, un affichage différent apparaît : des barres verticales s'allument en imitant des touches de piano.

Pour jouer au jeu d'estimation de temps : Sélectionner S7. Switcher 'start_stop_game' pour lancer le jeu, 'reset_game' pour le réinitialiser. Une fois le jeu lancé, le joueur ne voit aucun affichage et doit deviner à quel moment ce sont écoulés 10 secondes en switchant 'start_stop_game'. L'affichage révèle alors le temps écoulé, et SCORE si pile 10 secondes, LOSER sinon. Tant que le temps affiché est inférieur à 10 secondes, le joueur peut reswitch 'start_stop_game' tant de fois qu'il le souhaite pour résumer le comptage et tenter d'arriver pile à 10 secondes.

1) FSM pour passer d'un état à un autre (state selector FSM) : Pour changer de state avec le 'state_selector', nous avons voulu utiliser un toggle button plutôt qu'une Switch car cela nous paraissait plus facile et plus intuitif comme fonctionnement. Nous voulions qu'à chaque fois que l'on appuie sur le bouton, on change de state. Pour ce faire, la condition pour passer d'un state à un autre est la même que celle pour rester dans le state suivant (voir schéma). Avec cette logique, on évite de sauter des states car on demeure dans le même état même après avoir relâché le bouton (nous expliquons plus en détail le fonctionnement de ce toggle button plus loin).

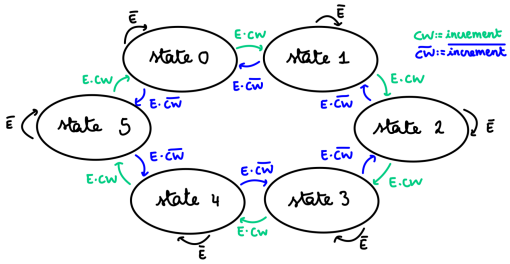


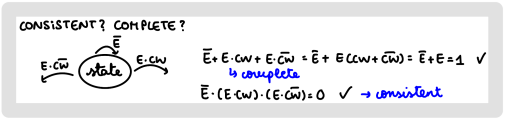
2) FSM des counters : nous avons réalisé des FSM spécifiques à différent comptage (de 0 à 5 pour les décimaux des secondes et minutes ; de 0 à 9 pour les 10ème et 100ème de seconde et les unités des secondes et minutes ; et de 0 à 23 pour les heures). Chaque chiffre de ces différents cycles de chiffres possède son propre code et représente un état dans la machine d'états finis. Les conditions incrément et increment-b sur les transitions permettent d'incrémenter et décrémenter le temps.

- counters 0-5, 0-9 : nous avons utilisé pour ces FSM le même nombre de bits (4 bits pour coder les états + 1 bit Enable + 1 bit "increment" pour incrémenter-1 ou décrémenter-0) afin de pouvoir utiliser le même décodeur pour 0-5 et 0-9
- counters 0-23 : pour cette FSM, il a fallu 5 bits pour coder les 24 états, 1 bit Enable et 1 bit "increment". Cette FSM est particulière car le cycle de chiffres est : 0 à 9 > 0 à 9 > 0 à 3 pour les unités des heures donc on a dû faire une FSM à 24 états on a fait 2 logiques de transition une pour l'incrémentation et une pour la décrémentation, on utilise un Demux et un Mux pour pouvoir alterner entre les 2 logiques de transition par rapport à la valeur booléenne du 'increment'.

Nous avons pu étendre l'usage de ces fonctions au service des autres (alarme, stopwatch, timer). Ayant réalisé les counters à la main, ils ont été des outils bien plus flexibles et manipulables pour nous. Par exemple, pour le stopwatch, on a pu recycler les counters 0-5 et 0-9 et on a utilisé' increment' à logique 1 constante. Pour le timer, c'est le même fonctionnement que l'heure mais avec une condition d'arrêt et de sonnerie.

La FSM pour le counter 0-5 et vérification de si elle est consistante et complète :





3) FSM musique : pour chaque chanson, une note possède un code unique et représente un état de la FSM ; on passe d'une note à une autre sans condition. Ceci crée une séquence de notes et forme la chanson. Chaque note ayant un code spécifique, nous lui avons attribué une fréquence particulière afin qu'elle émette le son correspondant. Pour ceci, nous avons fait des calculs par rapport à la fréquence principale de la machine (cf schéma à droite).

exemple avec la note C (octave 5) :

fréquence de C : $f_c = 523.251$ Hz

Si l'on choisit une fréquence principale de $f_{main} = 64$ kHz, nous voulons diviser cette valeur pour obtenir f_c.

64000 tics par seconde

$\frac{f_{main}}{x} = f_c \quad \Rightarrow \quad x = \frac{f_{main}}{f_c}$

un high tick correspond à la partie 'haute' du tic ici, c'est-à-dire la moitié d'1 tic 'entier' (qui se répète).

$\Rightarrow$ high tick de C : $x' = \frac{x}{2} = \frac{f_{main}}{2 \cdot f_c}$

4) Calcul des fréquences des clocks :

Main Physical CLK = 64 kHz

- For the Stopwatch, we need it at 1 Tick/0,01 Second
  to 100 Ticks per Second

$\frac{64 \cdot 10^3}{n} = 100$

$\Rightarrow n = 640$

$T = \frac{1}{f}$; For a 100 Hz Clock:

$T_{100Hz} = \frac{1}{100Hz} = 0,01$ Seconds.

- Duration of High Tick = Duration of Low Tick = $\frac{T_{100Hz}}{2} = 5$ ms
- $T_{64kHz} = \frac{1}{64 \cdot 10^3} = 15,625\,\mu s$

No of Ticks = $\frac{\text{Duration of High/Low}}{T_{64kHz}} = \frac{5\,ms}{15,625\,\mu s} = 320$ ticks

Summary:

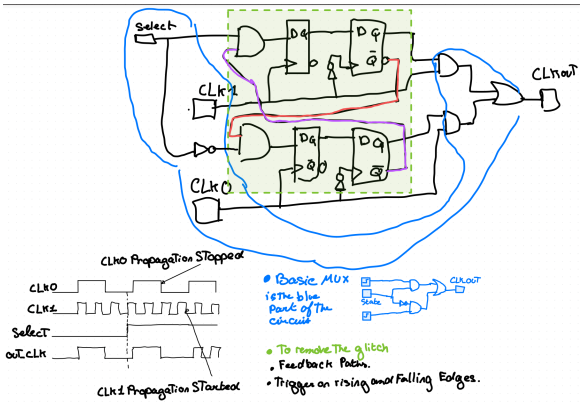
- High Tick Duration (For the Stop Gated)
  - Time: 5 ms
  - Ticks: 320 Tick of the 64 kHz CLK
- Low Tick duration
  - Time: 5 ms
  - Ticks: 320 Tick of the 64 kHz CLK
- High-Tick Seconds 32000

5) Gated Clocks pour le buzzer :

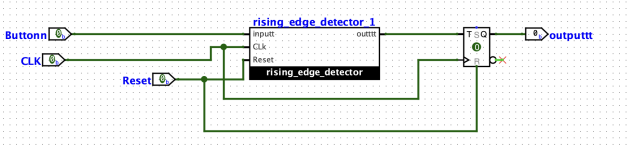
Pour toutes les fonctions du buzzer (musique, alarme de minuterie ou alarme), l'utilisation d'une Gated Clock ne pose aucun problème, car les éventuelles irrégularités (Glitches) ne sont pas perceptibles à l'oreille.

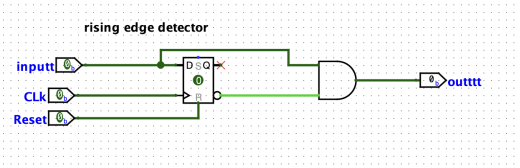
6) Le multiplexeur d'horloge sans glitch :

Le multiplexeur d'horloge sans glitch fonctionne en imposant une séquence de contrôle rigoureuse avant de transmettre la nouvelle horloge. Concrètement, il utilise des bascules (flip-flops) et des boucles de rétroaction pour surveiller l'état de l'horloge actuelle avant d'autoriser le passage de la suivante. D'abord, le circuit s'assure que le signal en cours est complètement bloqué, garantissant ainsi qu'aucune partie de celui-ci ne se superpose à l'horloge entrante. Cette étape, synchronisée grâce à des bascules déclenchées sur des fronts différents, permet de capturer et de stabiliser le signal de sélection. Une fois l'horloge précédente arrêtée et la nouvelle clairement validée, le multiplexeur ouvre la voie à cette dernière. Ainsi, la transition entre deux fréquences se réalise proprement, sans impulsions parasites, assurant la continuité et la fiabilité du système. Ce principe justifie pleinement l'utilisation de gated clocks sans provoquer de glitch. Ce circuit nous a permis de réduire le nombre total de circuits. Au lieu d'avoir un module dédié pour chaque fréquence, on peut désormais réutiliser la même architecture grâce à ce « switch » d'horloge.



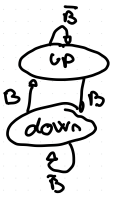
7) Toggle button pour la state selector FSM :





Afin de réaliser le fonctionnement d'un toggle button, nous avons créé un circuit simple mais utile. En connectant un T-flip-flop à un détecteur de fronts montants (schéma de droite), on obtient un montage qui se comporte comme un bouton à bascule : une impulsion place la sortie à l'état logique-1, une autre la ramène à l'état logique-0 (schéma à gauche).

8) Jumping Game :



Cette FSM contrôle le mouvement vertical du joueur, de la balle (voir schéma à droite). Un générateur pseudo-aléatoire est connecté à quatre D-FF en cascade (dans le fichier 'Game'), reliés aux sorties d'affichage, créant l'effet d'une « fusée » qui approche, à la manière d'un canon dans Super Mario Bros. Lorsque le pulse atteint le dernier D-FF, la position de la « fusée » est comparée à celle du joueur. Si le joueur n'esquive pas, il perd. Une autre FSM, sans condition, affiche alors « Game Over ».

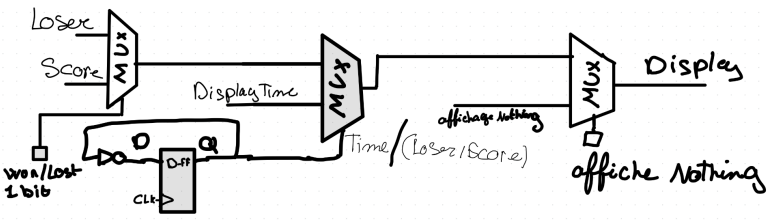
9) 10 Seconds Game :

Ce jeu s'appuie sur la logique d'un chronomètre, complétée par une logique supplémentaire. Si le temps est à 00:00:00, on n'affiche rien. Si le temps n'est pas à 00:00:00 et que start_stop = 0, deux cas se présentent :

- Si time ≠ 00:10:00, le joueur a perdu
- Si time = 00:10:00, le joueur a gagné

Durant le décompte, rien ne doit être affiché afin de ne pas faciliter le jeu. En cas de défaite ou de victoire, il faut alterner entre l'affichage du temps et du résultat, grâce à une FSM.

Logique d'affichage du 10 Seconds Game :



10) Information supplémentaire :

- Nous n'avons pas filtré les boutons 'global_reset' et 'state_selector' car ces derniers sont pré-filtrés. Le reste des boutons et switchs ont été filtrés.
- Le click vertical sur la molette est à logique-1 lorsque l'on ne clique pas dessus et logique-0 lorsque l'on clique. Donc, nous avons mis un inverseur dans le circuit logique pour résoudre ceci.
- Pour la sonnerie de l'alarme et du Timer, nous avons introduit un décalage d'un Tick. Cela nous permet de distinguer clairement la sonnerie, même si une autre alarme ou de la musique retentit à la même fréquence.