Projet Blackjack
EN202

Filière Électronique

Semestre 7

Besson Thomas

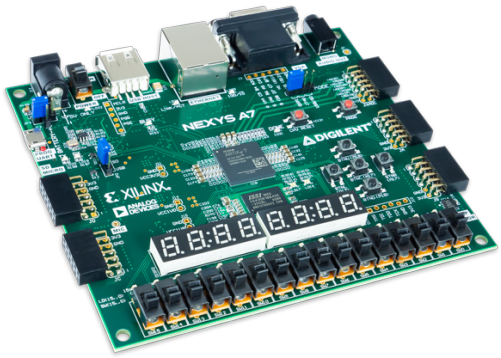
Encadrant : Y. Bornat

Table des matières

Codes complets du projet Blackjack

1 Introduction

Ce projet consiste à implémenter en **VHDL** le jeu du **Blackjack**. L'implémentation est réalisée sur une carte de prototypage **Digilent Nexys A7-100T**, cadencée par une horloge principale de **100 MHz**.



Le but du projet est de permettre à l'utilisateur de jouer une partie de Blackjack, incluant le tirage des cartes, le calcul des scores du joueur et du croupier, ainsi que l'affichage des informations de jeu sur des afficheurs 7 segments.

Ce projet permet de mettre en œuvre les principes de conception numérique vus au cours, notamment la description matérielle en VHDL et l'implémentation sur FPGA.

2 Règles du Blackjack

2.1 Objectif du jeu

Obtenir une main dont la valeur est la plus proche possible de 21 sans le dépasser, on parle de **bust** dans ce cas. Le joueur affronte uniquement le croupier, pas les autres joueurs **[1]**. Le jeu se joue avec 6 paquets de 52 cartes.

2.2 Valeur des cartes

Cartes	Valeur
2 à 10	Valeur de la carte
Valet, Dame, Roi	10
As	1 *ou* 11 (selon le cas le plus avantageux)

2.3 Déroulement d'une donne

- **Mise** : chaque joueur place sa mise.
- **Distribution** : deux cartes sont distribuées au joueur et au croupier.
- **Tour du joueur** : choix entre *tirer* ou *rester*.
- **Tour du croupier** : tire jusqu'à au moins 17.
- **Résolution** : comparaison des mains pour déterminer le résultat.

3 Cahier des charges

L'implémentation du Blackjack doit répondre aux exigences suivantes :

- Une partie de un joueur contre un croupier sans mise avec 6 paquets de 52 cartes (312 cartes) ;
- Interface par boutons ;
- Tirer une carte **aléatoire** lors d'un appui sur un bouton *HIT* par le joueur ;
- Permettre au joueur de valider sa main avec un appui sur un bouton *STAND* ;
- Calculer et afficher le **score du joueur** et du **croupier** après chaque tir tout en gérant les valeurs des cartes ;
- Respecter la règle du croupier : **tirer si score** < 17, arrêter sinon ;
- Afficher les informations sur **8 afficheurs 7 segments** ;
- Fournir un verdict **Victoire / Défaite / Égalité** ;
- Utiliser une LED RGB pour indiquer le résultat du jeu.

4 Architecture générale

4.1 Première approche

Cette section présente une vue d'ensemble du système avant d'entrer dans le détail de chacun des blocs matériels et de leur rôle dans le fonctionnement global du jeu.

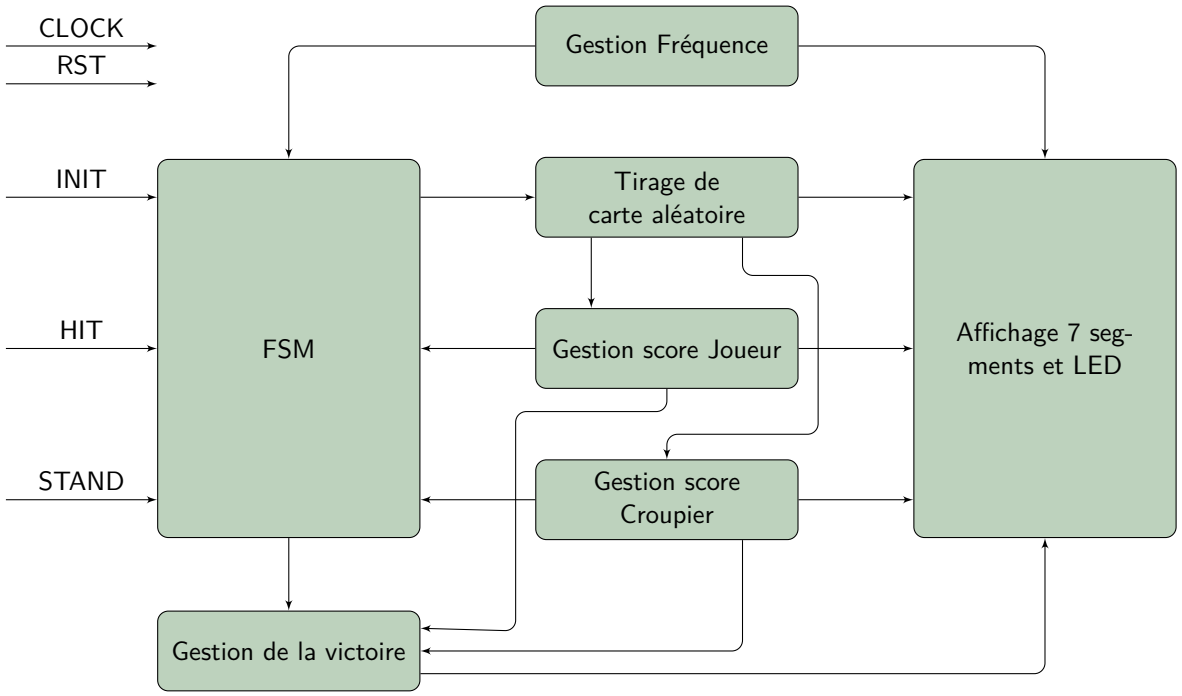


FIGURE 1 – Architecture du projet (CLOCK et RST sont distribués à l'ensemble des blocs séquentiels)

Ce premier modèle nous permet de fixer les blocs et la gestion de nos signaux tout au long du projet.

5 Machine d'état

La FSM (`fsm_blackjack`) est le cerveau du système. Elle définit l'enchaînement temporel des actions du jeu (tirage des cartes, attente joueur, tour du croupier et fin de partie), indépendamment des calculs de score et de l'affichage, qui sont délégués à des blocs spécialisés.

Elle coordonne la temporalité du jeu de manière synchrone. Elle a été conçue selon un modèle de Moore (les sorties dépendent uniquement de l'état courant).

Le diagramme suivant présente une version simplifiée de la machine à états, mettant en évidence les grandes phases du jeu et les conditions de transition associées.

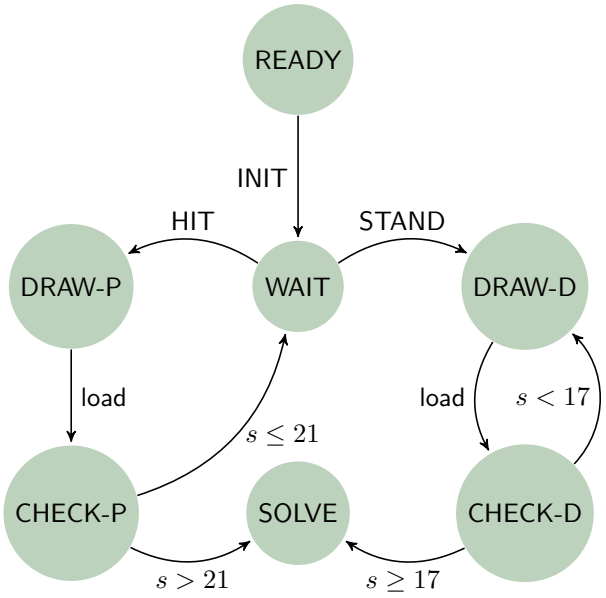


FIGURE 2 – Diagramme de transition d'une partie

Le diagramme présente une version simplifiée ; dans le code, la séquence croupier est détaillée en sous-états (DEALER_DRAW, DEALER_WAIT) pour intégrer la temporisation.

5.1 Initialisation et Attente (`READY`)

C'est l'état par défaut après un Reset.

- **Action :** Le signal `reset_jeu` est maintenu à '1', forçant les scores à zéro.
- **Transition :** Sur appui de `INIT`, on passe à `WAIT`.
- **Rôle :** Permet d'afficher un message d'accueil.

5.2 La boucle du Joueur (`WAIT` à `SOLVE`)

- **`WAIT` :** La FSM scrute les boutons. C'est un point d'arrêt stable.
- **`DRAW-P` :** Si "HIT" est pressé, cela déclenche le tirage de la carte et l'addition dans le Score Manager.
- **`CHECK-P` :** Juste après le tirage, la FSM compare `score_player` à 21.
  Logique de décision : Si le score est supérieur à 21, transition immédiate vers `SOLVE`. Le tour du croupier est annulé. Sinon, retour à `WAIT`.

5.3 La boucle du Croupier (WAIT à SOLVE)

Si le joueur appuie sur "STAND", la FSM devient autonome.

- `DRAW-D` **:** Point d'entrée de l'algorithme.
- **Algorithme :** On vérifie `score_dealer`.
  - Si le score $<$ 17 : Transition vers `DRAW-D` (Tirer une carte).
  - Si le score $\geq$ 17 : Transition vers `SOLVE` (Fin de partie).

5.4 Le résultat (SOLVE)

État final stable.

- **Action :** Le signal `is_final` passe à '1'. C'est le signal de validation pour le module suivant
- **Sortie :** Un appui sur `INIT` renvoie à `READY`.

6 Tirage pseudo-aléatoire : `random_hit`

La génération d'un tirage aléatoire constitue une difficulté conceptuelle sur FPGA, celui-ci étant par nature un système déterministe. Dans le cadre du jeu de Blackjack, l'utilisation de plusieurs paquets de cartes rend acceptable la possibilité de tirer plusieurs fois une même carte au cours d'une partie. Cette hypothèse permet de simplifier la modélisation du tirage tout en restant cohérente avec le cahier des charges.

Le tirage « aléatoire » est obtenu via le **joueur** : un compteur tourne à la fréquence de l'horloge en continu, et sa valeur est capturée au moment exact où l'utilisateur appuie sur *HIT*.

```
if rising_edge(clk) then
    if enable_in = '1' then
        if cnt_rank = 12 then
            cnt_rank <= (others => '0');
        else
            cnt_rank <= cnt_rank + 1;
        end if;
    end if;
end if;
```

Listing 1 – Tirage pseudo-aléatoire d'une carte

Le compteur est modulo 13 puis la valeur est mappée vers 2..11 (As inclus). On capture lors d'un hit la valeur du compteur et on l'adapte pour permettre de l'envoyer à la FSM.

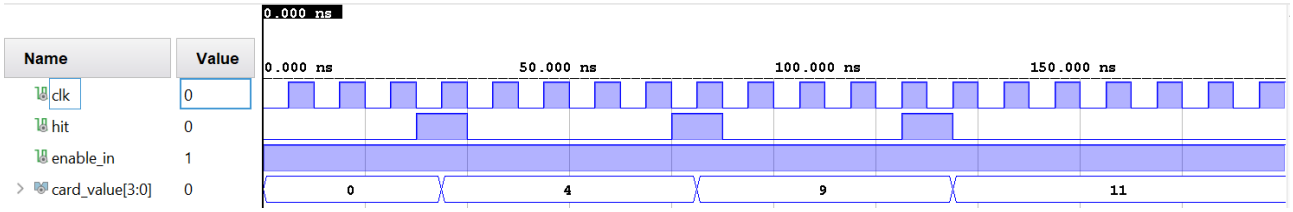


FIGURE 3 – Chronogramme de `random_hit`

Le chronogramme de la figure ci-dessus valide le fonctionnement du bloc `random_hit` : à chaque impulsion sur le signal *HIT*, une nouvelle valeur de carte est échantillonnée et reste ensuite stable jusqu'au tirage suivant. Le chronogramme montre trois tirages distincts (4, 9 et 11 correspondant à un As), confirmant le caractère pseudo-aléatoire du mécanisme basé sur l'action de l'utilisateur.

Cette implémentation ne simule pas un paquet réel avec retrait des cartes déjà tirées, mais fournit un comportement statistiquement acceptable dans le cadre du projet.

7 Gestion du score : score_manager

Le module score_manager assure l'ensemble de la logique de calcul des scores, indépendamment de la FSM, ce qui permet de simplifier le contrôle du jeu et de rendre le système plus modulaire.
Le score est un accumulateur (sur 8 bits en sortie) piloté par un signal load. La gestion de l'As est intégrée : l'As vaut 11 au tirage, puis peut être ramené à 1 si le score dépasse 21. Pour cela un compteur incrémente le nombre d'As tirés pour permettre de retirer 10 si un As a été tiré et que le score dépasse 21.

```
if card_value = "1011" then
  temp_score := temp_score + 11;
  temp_aces := temp_aces + 1;
else
  temp_score := temp_score + to_integer(unsigned(card_value));
end if;

for i in 0 to 3 loop
  if temp_score > 21and temp_aces > 0then
    temp_score := temp_score - 10;
    temp_aces := temp_aces - 1;
  end if;
end loop;
```

Listing 2 – Accumulation et gestion d'As

Une boucle bornée (4 itérations) suffit car au plus 4 As peuvent impacter simultanément la correction dans notre représentation.

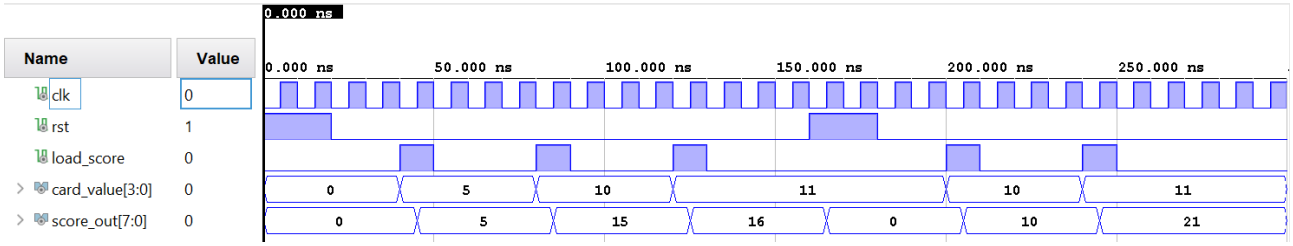


FIGURE 4 – Chronogramme de score_manager

Le chronogramme de la figure ci-dessus valide le fonctionnement du module score_manager. On observe dans un premier temps l'accumulation correcte des cartes, suivie de l'ajout d'un As qui est automatiquement ajusté afin d'éviter un dépassement de 21, conduisant à un score de 16. Après une remise à zéro du score, un second scénario est testé, montrant un score de 10 puis 21 suite au tirage d'un As, ce qui confirme la bonne gestion dynamique de la valeur des As.

8 Résultat : win_manager

Le module win_manager est un bloc purement combinatoire chargé de déterminer l'issue de la partie. Tant que le signal is_final = '0', toutes les sorties restent à l'état nul, indiquant qu'aucun résultat définitif n'a encore été établi.
Lorsque is_final = '1', le module applique un ordre de priorité strict afin de déterminer le vainqueur :

- Si le joueur dépasse 21 (*bust joueur*), la victoire est attribuée au croupier.
- Si le croupier dépasse 21 (*bust croupier*), la victoire est attribuée au joueur.
- Sinon, les scores du joueur et du croupier sont comparés.
- En cas d'égalité, le signal draw est activé.

Ce module centralise ainsi toute la logique de décision de fin de partie. Il permet de séparer clairement le calcul du résultat du jeu de sa représentation visuelle, facilitant par la suite l'implémentation de l'affichage de fin de partie.

9 Architecture globale : `top_blackjack`

L'entité `top_blackjack` est la clé de voûte du système. Elle ne contient aucune logique complexe, mais assure le routage des signaux entre les sous-modules.

Les flux de données principaux sont :

- **Flux de Contrôle :** La FSM reçoit les entrées utilisateurs et envoie des commandes.
- **Flux de Données :** Le générateur aléatoire produit `card_val`, et est connecté en parallèle aux deux `score_manager`.
- **Flux de Visualisation :** Les scores sortent des `score_manager` pour aller vers deux destinations simultanées : le `win_manager` (pour le verdict) et le multiplexeur d'affichage.

10 Gestion de l'affichage

L'affichage constitue une contrainte matérielle forte du projet, en raison du nombre limité de sorties disponibles et de la nécessité d'un rafraîchissement rapide pour garantir une lecture confortable.

10.1 Gestion des Horloges et Synchronisation

La carte Nexys A7 fournit une horloge de 100 MHz. Cela signifie que des signaux d'activation sur les fronts montants de l'horloge (Clock Enable) sont nécessaires pour afficher de l'information à des fréquences exploitables par l'œil humain ou pour temporiser les signaux. Dans ce projet, deux signaux d'activation sont nécessaires :

- **CE_perception (2 kHz) :** Utilisé par le module d'affichage. Il est impératif pour tromper l'œil humain (persistance rétinienne) lors du multiplexage des 8 digits.
- **CE_jeu (1 Hz) :** Utilisé pour la logique de jeu. Il permet d'introduire des délais artificiels (notamment l'état `WAIT_1S`) pour que le joueur ait le temps de voir la carte être tirée.

Répartition réelle des informations (dans `top_blackjack`) Le tableau ci-dessous décrit exactement ce qui est envoyé dans `digits(0..7)` puis sélectionné par `mux8` **[2]**.

AN	commande	digits(i)	Contenu affiché
AN0	000	digits(0)	Carte **joueur** (dizaine)
AN1	001	digits(1)	Carte **joueur** (unité)
AN2	010	digits(2)	Score **joueur** (dizaine)
AN3	011	digits(3)	Score **joueur** (unité)
AN6	110	digits(6)	Score **croupier** (dizaine)
AN7	111	digits(7)	Score **croupier** (unité)

10.2 Affichage : logique de `top_blackjack`

Message d'accueil - le process d'affichage force :

$$digits = \{\text{“B”}, \text{“L”}, \text{“A”}, \text{“C”}, \text{vide}, \text{“J”}, \text{vide}, \text{vide}\}$$

L'affichage de lettres est permis par `transcodeur` (codes spéciaux x"C", x"D", x"E").

Affichage en partie (règles réelles) Toujours dans `top_blackjack`, le process :

- affiche les scores en dizaines/unités ;
- affiche le score du croupier ;
- fait clignoter le score du perdant en fin de partie.

Clignotement du perdant Le clignotement est un masquage sélectif du multiplexage. On utilise un bit d'un compteur comme un interrupteur. Si cet interrupteur est à '0' et que le joueur ou le croupier a perdu la partie, on envoie le code d'extinction (x"F") au décodeur au lieu du score. C'est une solution qui évite de créer un module supplémentaire.

LED RGB (LD16) avec PWM [2] Un compteur `pwm_cnt` (8 bits) crée un rapport cyclique faible (`pwm_on` actif quand `pwm_cnt < 25`). En fin de partie :

- Victoire joueur : LED verte modulée
- Victoire croupier : LED rouge modulée
- Égalité : blanc (R+G+B) modulé

11 Validation du cahier des charges

11.1 Tests réalisés

- **Reset** : retour en accueil, affichage « BLAC J », scores à 0.
- **INIT** : transition `READY` $\rightarrow$ `WAIT`.
- **HIT** : le tirage d'une carte fait une impulsion `load_player` (1 cycle), et le score du joueur est mis à jour après un délai de 1 seconde.
- **Gestion d'As** : un As provoquant un score > 21 entraine une correction (soustraction de 10).
- **Bust joueur** : si le score du joueur > 21, passage au `SOLVE`.
- **STAND** : tant que le score croupier < 17, tirage automatique et temporisation (`DEALER_DRAW` puis `DEALER_WAIT`).
- **Règle croupier** : arrêt dès que score_croupier ≥ 17 puis `SOLVE`.
- **Verdict** : activation de `win_p/win_d/draw` uniquement quand `is_final='1'`.
- **Affichage** : clignotement du perdant visible et LED.

11.2 Synthèse de conformité

Exigence cahier des charges	Statut
FSM du jeu (joueur et croupier)	Validé
Hit pseudo-aléatoire	Validé (`random_hit`)
Scores indépendants	Validé (2x `score_manager`)
Règle croupier (tirer à 16, stop à 17)	Validé (`fsm_blackjack`)
Affichage 8 digits multiplexés	Validé (`mod8+mux8+trans`)
Verdict Win/Lose/Draw	Validé (`win_manager`)
Signalisation LED RGB	Validé

12 Conclusion

Ce projet a abouti à une implémentation fonctionnelle du Blackjack sur FPGA en VHDL, permettant à l'utilisateur de jouer contre un croupier selon les règles définies dans le cahier des charges. La partie est entièrement pilotée par une machine à états, tandis que le calcul des scores (incluant la gestion dynamique des As), le tirage pseudo-aléatoire et l'interface d'affichage sont délégués à des blocs dédiés.

En perspective, plusieurs améliorations sont envisageables : enrichir les règles (double, split, assurance), ajouter la gestion de mises, introduire un véritable paquet de cartes avec retrait des cartes déjà tirées, ou encore améliorer l'ergonomie de l'affichage (messages dédiés, animation de fin de partie, affichage en VGA).

13 Annexe

Liste des fichiers VHDL et rôle de chaque bloc

Fichier	Rôle dans l'architecture
`top_blackjack.vhd`	Top-level : interconnexion de tous les blocs, gestion de l'affichage (construction des 8 digits), gestion LED RGB, clignotements en fonction du résultat.
`fsm_blackjack.vhd`	Unité de contrôle (FSM) : séquence de jeu (READY, actions joueur, tour croupier, final), génération des signaux de commande.
`random_hit.vhd`	Générateur pseudo-aléatoire : compteur cyclique (0..12) et capture sur `hit`.
`score_manager.vhd`	Gestion de score (joueur et croupier) : accumulateur interne, avec une sortie codée sur 8 bits, mis à jour sur `load_score` et gestion des As.
`win_manager.vhd`	Comparateur combinatoire du verdict : lorsque `is_final='1'`.
`gestion_freq.vhd`	Génération des CE : `CE_perception` (balayage affichage), `CE_jeu` (temporisation 1 s).
`mod8.vhd`	Compteur de multiplexage : sur `CE_perception` incrémente un index 0..7 (`commande`) et pilote l'anode active bas `AN[7:0]`.
`mux8.vhd`	Multiplexeur 8 : sélectionne la sortie `sept_seg` selon `commande`.
`trans.vhd`	Transcodeur 4 bits → 7 segments : nombres 2..11 et codes spéciaux pour lettres.

Références

[1] *Le blackjack, règles et histoire*, https://fr.wikipedia.org/wiki/Blackjack_(jeu)

[2] Digilent Inc., *Nexys A7 Reference Manual*, https://digilent.com/reference/programmable-logic/nexys-a7/reference-manual

Lien Github du projet

Codes du projet Blackjack