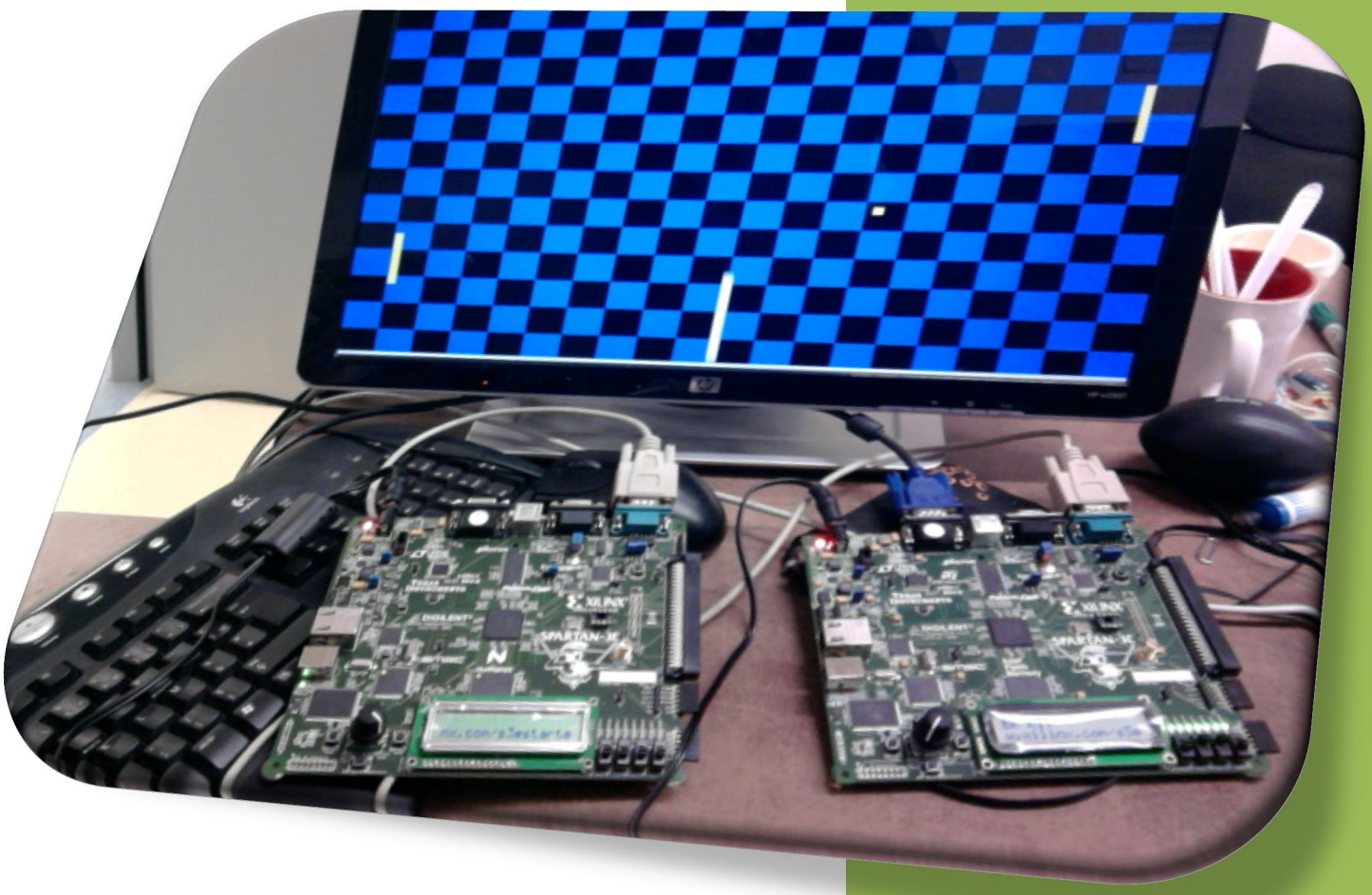
UPMC
SORBONNE UNIVERSITÉS

3E201

Console PMC-E201 sur FPGA



Enoncé Mini-Projet

Julien Denoulet

3E201 - Mini Projet – Console PMC-E201

Printemps 2013

INTRODUCTION

L'objectif de ces séances de TP est de réaliser une console de jeux sur FPGA. Pour cela, on partira d'un prototype partiellement fonctionnel de la console. Votre travail consistera à :

- Prendre en main cette version intermédiaire de la console ainsi que les logiciels.
- Compléter les modules manquants de l'architecture afin de réaliser le cahier des charges de la console
- Proposer (et implémenter) des améliorations pour la console.

Les architectures numériques à réaliser seront décrites en VHDL, simulées à l'aide du logiciel Modelsim, puis implémentées sur une carte Xilinx Spartan 3E à l'aide du logiciel ISE.

L'évaluation sera composée de deux notes :

- Note technique : basée sur l'avancement de votre projet.
- Note artistique : suite à une présentation orale à la fin du semestre.

PRE-ETUDE

Le cahier des charges de la console se trouve dans le document « Manuel Utilisateur + Carte FPGA »

Ce document contient :

- Une présentation générale de la console ***PMC-E201***
- La présentation et les règles des jeux implémentés dans la ***PMC-E201***
- L'interface utilisateur de la ***PMC-E201*** et son rôle pour chacun des jeux de la console.
- Une présentation de la carte FPGA ***Spartan 3E*** sur laquelle est implémentée la console.
- Une description générale ainsi qu'un synopsis de l'architecture interne du FPGA.

Travail à effectuer AVANT la 1ère séance de TP

- Lire attentivement le document « Manuel Utilisateur + Carte FPGA »
- Noter les questions que vous pourriez avoir suite à cette lecture. Vous pourrez les poser à l'intervenant de TP.
- Spécifier l'architecture et le code VHDL du diviseur d'horloge comme indiqué.
    - **Le code et l'architecture devront être validés par l'intervenant de TP pour que vous ayez accès au code VHDL de la console.**

PRISE EN MAIN

Dans cette partie, on va prendre en main les outils et la carte du projet. On va pour cela :

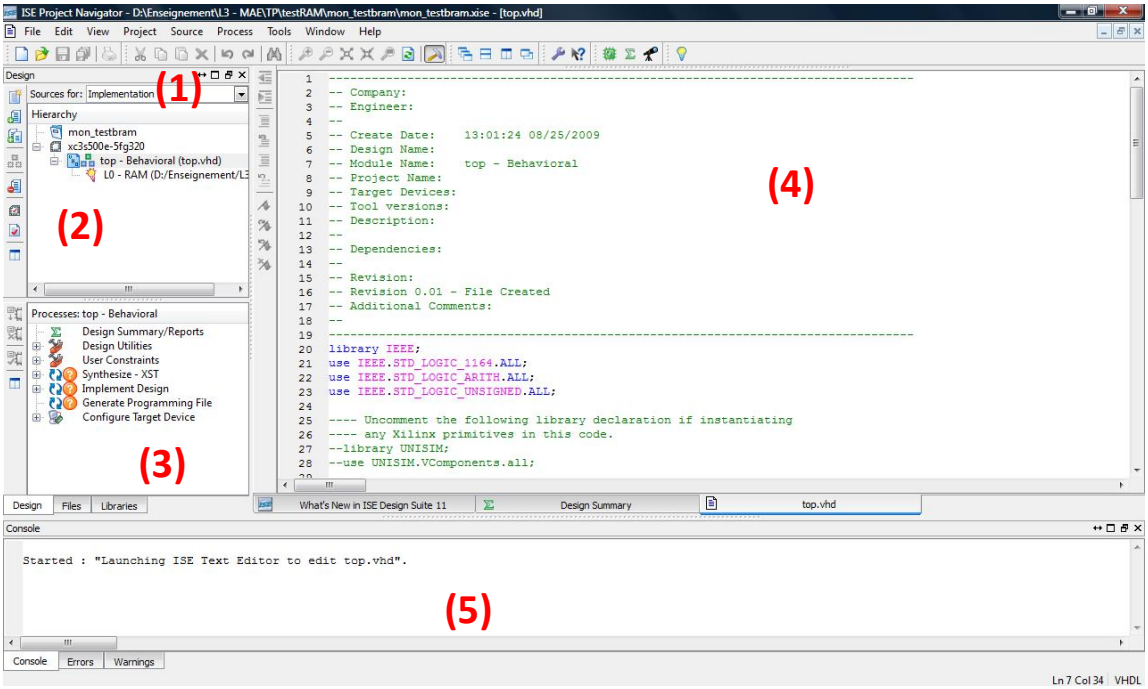
- Créer un projet dans Xilinx ISE et ajouter à ce projet les sources VHDL de la console ***PMC-E201***
- Coder un module simple de division d'horloge et l'ajouter au projet
- Simuler ce module de division d'horloge avec ***Modelsim***
- Implémenter la console sur la carte FPGA.

Création d'un projet avec le logiciel ISE

- Ouvrir le logiciel Xilinx ISE. (***Menu Démarrer → Tous les Programmes → Xilinx ISE 12.2 → ISE Design Tools → Project Navigator***)
- Dans le menu ***File***, cliquer sur ***New Project.*** Entrer un nom et un emplacement sur le disque pour votre projet. Cliquer sur ***Next***
- Dans la fenêtre ***Device Properties***, renseigner les items suivants :
    - Product Category : ***All***
    - Family : ***Spartan 3E***
    - Device : ***XC3S500E***
    - Package : ***FG320***
    - Speed : ***-5***
    - Simulator : ***Modelsim SE VHDL***
    - Preferred Language : ***VHDL***

  Ils correspondent au circuit FPGA de la carte de TP. Cliquer sur ***Next*** puis ***Finish.***

A l'écran s'affiche alors un ensemble de fenêtres similaire à la figure ci-dessous.



(1) Fenêtre ***Sources for*** : permet de choisir le mode de fonctionnement du logiciel (simulation ou implémentation)
(2) Fenêtre ***Hiérarchie*** : Contient les différentes sources (VHDL, IP Cores, etc…) inclus dans le projet
(3) Fenêtre ***Processes*** : Permet de lancer les différentes étapes du flot de conception (simulation, synthèse, placement/routage, etc…)
(4) Fenêtre ***Sources*** : On peut y visualiser et modifier les diverses sources du projet ainsi que les informations concernant l'implémentation du FPGA.
(5) Fenêtre ***Console*** : Retourne les différents messages envoyés par ISE.

Configuration de Modelsim pour Xilinx ISE

- Aller dans le menu ***Edit → Preferences***
- Dans la partie gauche de la fenêtre, dérouler le menu ***ISE General*** et choisir ***Integrated Tools***.
- Dans la section ***Modeltech Simulator***, cliquer sur la case **. . .** et rentrer le chemin d'accès au simulateur Modelsim : ***c:\Modeltech_6.6f\win32\modelsim.exe***

Ajout des sources VHDL de la console PMC-E201

- Récupérer les sources VHDL de la console à l'endroit indiqué par l'intervenant de TP. Copier ces fichiers dans le répertoire du projet.

- Dans le menu ***Project***, cliquer sur ***Add Source***. Sélectionner tous les fichiers VHDL. Cliquer sur ***Ouvrir*** puis ***OK***

- Ces fichiers proposent une version incomplète de la console ***PMC-E201***, qu'il va falloir à présent compléter.
    - Cette version suit globalement le synopsis que vous avez analysé lors de la pré-étude.

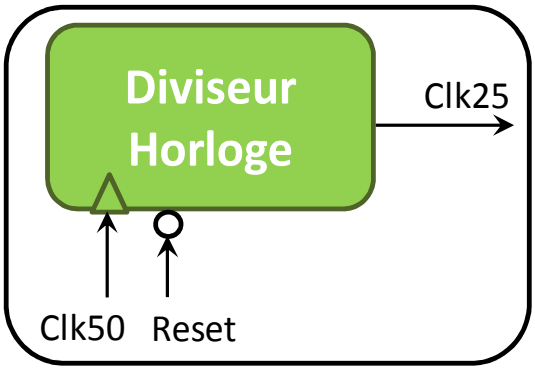
Ajout d'un module de division d'horloge

L'horloge de la carte ***Spartan 3E*** est cadencée à 50 MHz. En revanche, la fréquence de fonctionnement de la console ***PMC-E201*** doit être de 25 MHz. Il faut donc diviser par 2 l'horloge de la carte. Cette fonction n'est pour le moment pas présente dans le code VHDL de la console. Il faut donc l'ajouter à l'aide d'un fichier VHDL que vous allez décrire.

- Vérifier que la fenêtre ***Sources for*** est en mode ***Simulation***

- Dans le menu ***Project***, cliquer sur ***New Source***. Choisir le type ***VHDL Module***, donner le nom ***ClkDiv*** et vérifier que la case ***Add to Project*** est bien cochée. Cliquer sur ***Next***.
    - Le nom ClkDiv sera celui de l'entité. Il est important de bien respecter ce nom car il est utilisé par les autres modules VHDL du projet.

- Entrer les différentes entrées et sorties du module conformément à la figure ci-dessous. Encore une fois, il est indispensable de respecter les noms des ports d'entrée/sortie. Cliquer sur ***Next*** puis ***Finish***.



Entrées :

- ***Clk50*** : Horloge 50 MHz fournie par la carte
- ***Reset*** : Reset asynchrone, actif à l'état bas

Sortie :

- ***Clk25*** : Horloge de sortie à 25 MHz

- Dans la fenêtre ***Hiérarchie***, double cliquer sur ***ClkDiv.vhd*** pour ouvrir le fichier dans la fenêtre ***Sources***. Vous constaterez que l'entité du module VHDL a déjà été pré-remplie conformément aux indications que vous avez fournies précédemment.

- Pour effectuer la division d'horloge, il faut à chaque front montant de l'horloge ***Clk50*** inverser le niveau logique de la sortie ***Clk25***
  - Proposer un schéma bloc du diviseur d'horloge. **Le faire valider par l'intervenant de TP**
  - Décrire cette architecture en VHDL.
  - A tout moment, vous pouvez vérifier la conformité de votre code en :
    1) Sélectionnant le mode ***Implementation*** dans la fenêtre ***Sources for***
    2) Sélectionnant votre source VHDL dans la fenêtre ***Hiérachie.***
    3) Puis en cliquant dans la fenêtre ***Process*** sur l'option ***Check Syntax.***
  - Corriger les erreurs de code éventuelles

Testbench et simulation avec Modelsim

- Sélectionner le mode ***Simulation*** dans la fenêtre ***Sources for***

- Créer une nouvelle source pour le projet de type ***VHDL Testbench***. Nommer la source ***Simu*** et associer le testbench au module ***ClkDiv***

- Double cliquer sur l'objet ***Simu*** pour ouvrir le fichier dans la fenêtre ***Sources***

- Effacer tous les process de l'architecture et écrire le code suivant

```
Clk50<=not Clk50 after 10 ns;
reset<='1' after 15 ns;
```

- Dans la fenêtre ***Process***, dérouler le menu ***Modelsim Simulator*** et double cliquer sur ***Simulate Behavioral Model***. Le logiciel ***Modelsim*** s'ouvre alors.
  - Si vous voyez un message de warning s'afficher, cliquer sur ***NO***
  - Vous pouvez vérifier le bon fonctionnement de votre diviseur d'horloge.

Implémentation sur la carte FPGA

Nous allons procéder à l'implémentation de la console sur la carte ***Spartan 3E***.

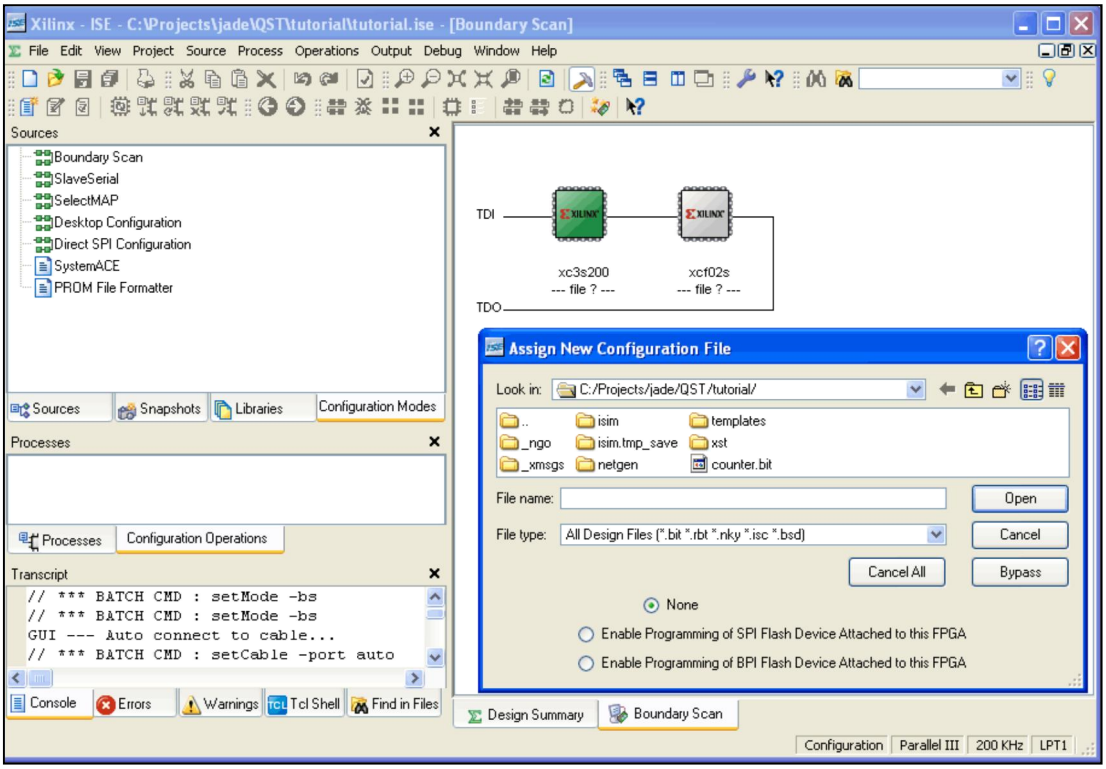
- Vérifier que la fenêtre ***Sources for*** est en mode ***Implementation***, puis dans le menu ***Project***, sélectionner ***Add Source*** puis sélectionner le fichier ***pin.ucf***.

- Le fichier ***UCF*** a pour objet de connecter les entrées/sorties du code VHDL aux entrées/sorties de la carte.

- Ouvrir le fichier.ucf en allant dans la fenêtre ***Processes***, et en cliquant sur ***Edit Constraints File***. Regarder le fichier et vérifier que les noms des broches (NET) correspondent aux noms des ports d'entrée/sortie du fichier ***top.vhd*** de la console.

- Dans la fenêtre ***Hierarchy***, cliquer sur le fichier ***top*** puis dans le menu ***Processes***, exécuter ***Synthesize***.
    - Cette procédure analyse les sources VHDL et les transforme en cellules logiques de base.
    - Une fois la synthèse terminée, vous pouvez cliquer sur l'onglet ***Design Summary*** et analyser le travail de l'outil. On peut y voir notamment le nombre de buffers d'entrées/sorties (IOB) ou le nombre de bascules. Cela permet de vérifier si la synthèse est bien en accord avec votre description VHDL.
- Dans la fenêtre ***Processes***, exécuter la fonction ***Implement Design***, qui déroule successivement les process ***Translate***, ***Map*** et ***Place & Route***.
    - Ces différents process vont placer les cellules logiques identifiées lors de la synthèse dans le FPGA, conformément aux contraintes données par le fichier ***UCF***. Les cellules placées sont ensuite routées les unes avec les autres.
    - Après cette étape, consulter le ***Design Summary*** pour connaître le taux d'occupation du FPGA.

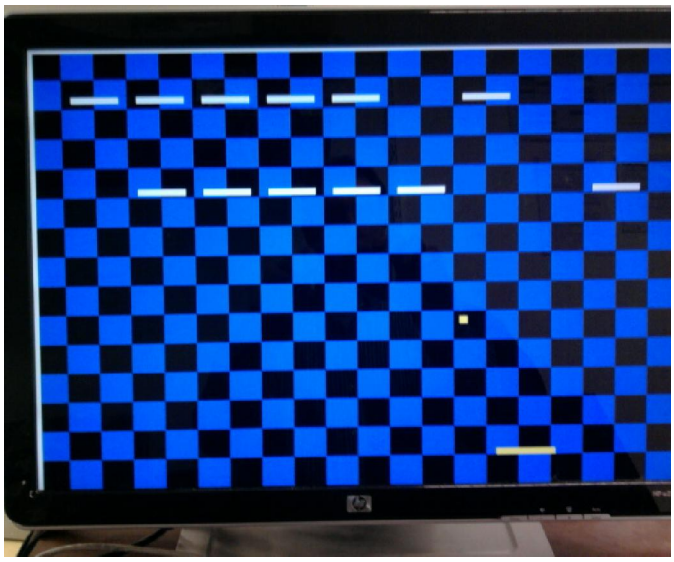
- Exécuter à présent ***Generate Programming File*** puis ***Configure Target Device***. Ce dernier process lance le logiciel ***Impact***.



- Dans ***Impact***, effectuer les commandes suivantes :
    - Cliquer sur ***File*** → ***New Project***.
    - Cocher l'option ***Configure devices using Boundary-Scan*** et vérifier que l'option ***Automatically connect to a cable…*** est sélectionnée. Cliquer sur ***Finish***, sélectionner le fichier ***TEST.bit***, puis cliquer successivement sur ***No, Bypass***, ***Bypass*** et ***OK***. Cliquer ensuite avec le bouton droit sur le composant ***xc3s500e***, choisir ***Program*** puis ***OK***.

- Le FPGA est à présent programmé.
    - Connecter la sortie VGA de la carte ***Spartan 3E*** à l'écran de votre PC.
    - Positionner l'interrupteur ***S0*** vers le haut
    - Vous devez normalement voir le jeu Casse-Briques s'afficher

- En appuyant sur le bouton ***Ouest***, vous pouvez faire un Reset de la console.



- Par rapport au cahier des charges, on peut constater qu'il y a des fonctions non réalisées
    - L'encodeur rotatif n'a pas d'effet sur le déplacement de la raquette
    - Si on appuie sur le bouton ***Nord*** ou ***Est***, il n'y a pas de changement de jeu (on reste toujours sur Casse-Briques)
    - Il ne se passe rien quand la balle sort en bas de l'écran.
        - Normalement, l'écran devrait devenir rouge et une nouvelle partie devrait débuter.
    - Le switch ***S1*** permet de mettre le jeu en pause.
        - Normalement c'est le rôle du bouton de l'encodeur rotatif.

Dans les étapes suivantes, voua allez ajouter ces fonctionnalités manquantes pour réaliser complètement le cahier des charges de la ***PMC-E201***.

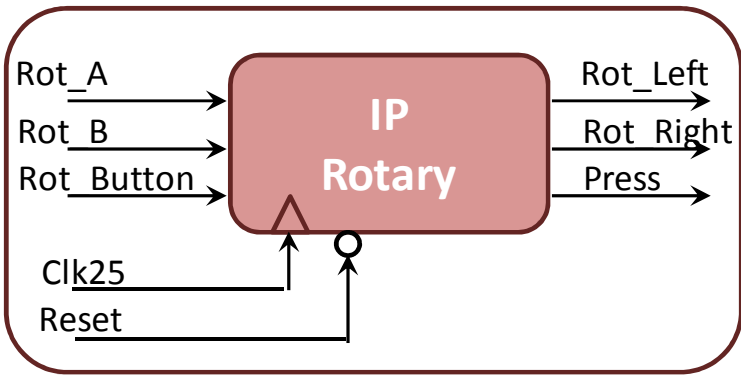
DEPLACEMENT DE LA RAQUETTE

Dans cette partie, on va réaliser une IP (Intellectual Property) permettant de gérer l'encodeur rotatif de la carte ***Spartan 3E*** et ainsi effectuer les déplacements de la raquette du jeu Casse-Briques. Il s'agira notamment de développer la machine à états de cette IP.

Présentation de l'IP Rotary

Ce module permet de gérer l'encodeur rotatif de la carte ***Spartan 3E***. Il génère les commandes de déplacement ***Rot_Left*** et ***Rot_Right*** (rotation vers la gauche ou vers la droite) ainsi qu'un signal ***Press*** indiquant l'état du bouton poussoir de l'encodeur. Ces signaux sont utilisés par la suite pour piloter la raquette du jeu Casse-Briques, ainsi que la raquette gauche du jeu Pong.

NB : Sauf indication contraire, tous les signaux sont actifs à l'état haut

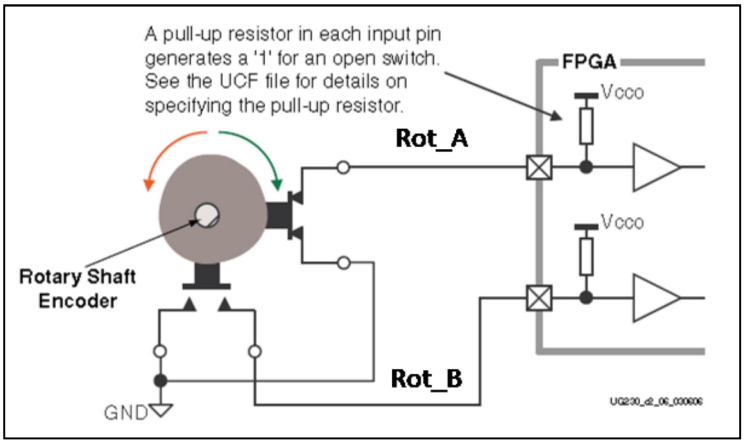


Entrées :

- ***Clk25*** : Horloge 25 MHz
- ***Reset*** : Reset asynchrone, actif à l'état bas
- ***Rot_A***: 1er Interrupteur de l'encodeur
- ***Rot_B***: 2ème Interrupteur de l'encodeur
- ***Rot_Button***: Bouton de l'encodeur

Sorties :

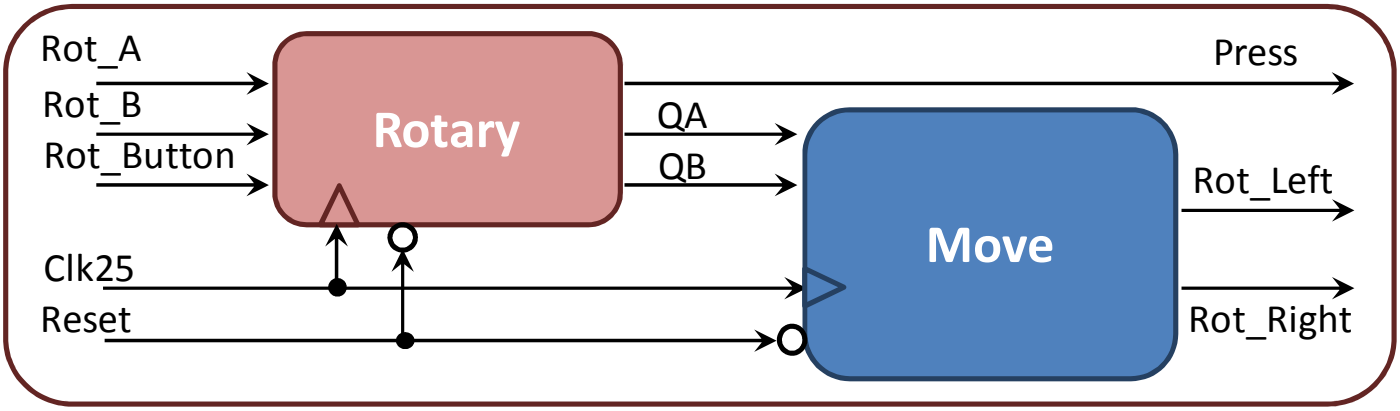
- ***Rot_Left***: Commande de rotation à gauche
- ***Rot_Right***: Commande de rotation à droite
- ***Press***: Commande d'appui sur le bouton



Les signaux ***Rot_A*** et ***Rot_B*** sont générés par des interrupteurs situés sur l'encodeur. Leur état dépend des rotations de l'encodeur (voir figure ci-contre). Selon l'évolution des niveaux logiques de ***Rot_A*** et ***Rot_B*** (Par exemple, est-ce que ***Rot_A*** passe au niveau haut avant ou après ***Rot_B*** ?), il est possible de déterminer le sens de rotation de l'encodeur.

Fonctionnement de IP Rotary

- Le module **IP Rotary** est composé de deux sous-blocs
  - **Rotary** gère les signaux issus de l'encodeur rotatif
  - **Move** est une Machine à Etats qui génère les commandes de déplacement ***Rot_Left*** et ***Rot_Right***



- Dans l'état actuel de la console ***PMC-E201***, le module **Rotary** est déjà implémenté (fichier ***rotary.vhd***).

- Ouvrir le fichier ***ip_rotary.vhd***. On peut y trouver l'instanciation du module **Rotary**. On y trouve également deux instructions forçant les sorties ***Rot_Left*** et ***Rot_Right*** à '0' (cela explique pourquoi les mouvements de l'encodeur ne sont pas répercutés à l'écran sur les déplacements de la raquette).

- Il vous faut donc supprimer ces deux instructions et les remplacer par un module **Move** qui va implémenter la MAE générant les bons signaux ***Rot_Left*** et ***Rot_Right***.

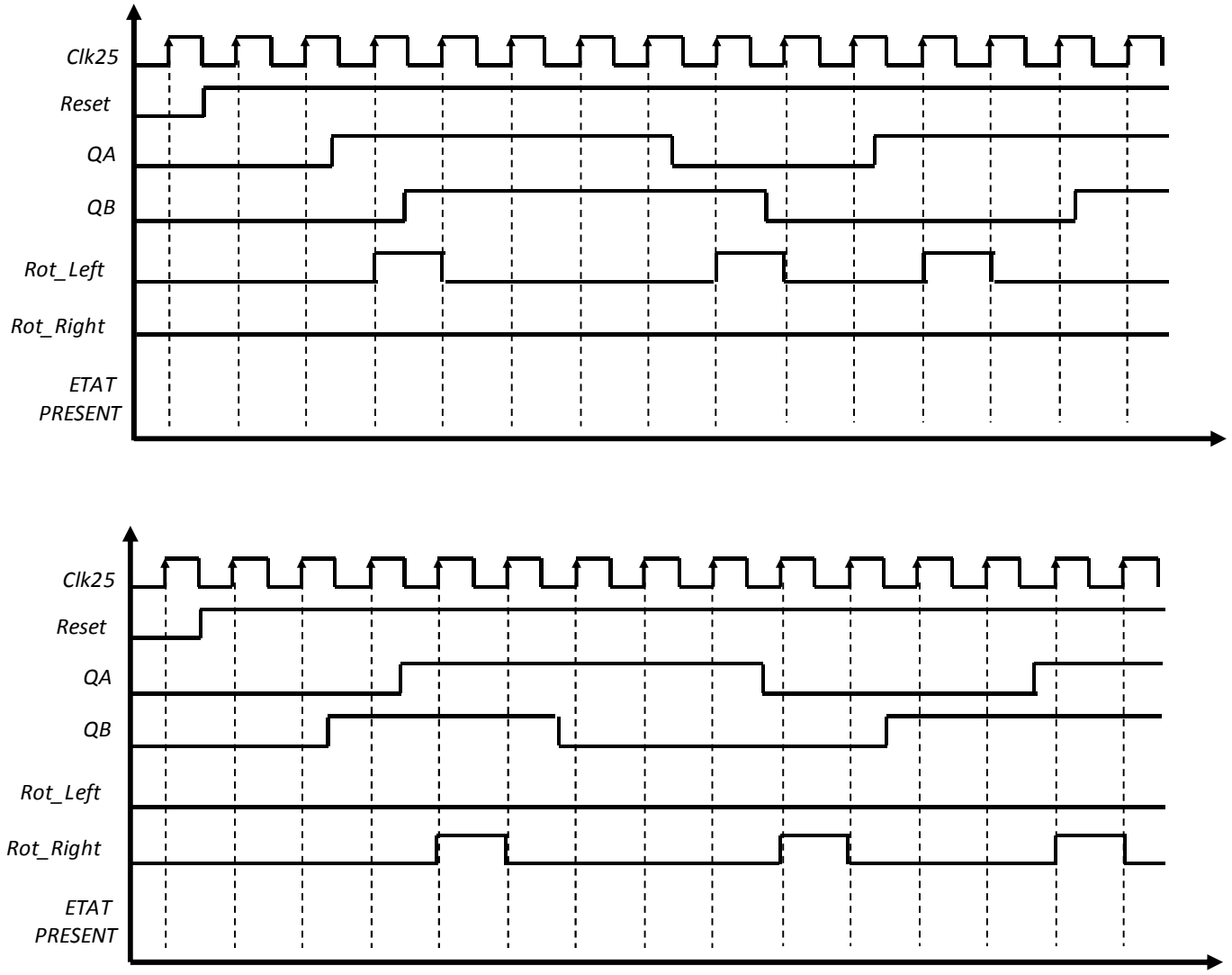
Fonctionnement de Move

- Ce sous-module est une machine à états.
  - Elle prend en entrée les signaux ***QA*** et ***QB*** et génère les sorties ***Rot_Left*** et ***Rot_Right***.

- ***QA*** et ***QB*** sont des signaux qui évoluent de façon similaire, mais avec un déphasage. Ce déphasage dépend du sens de déplacement de l'encodeur.
  - En déterminant si le signal ***QA*** est en avance ou en retard sur le signal ***QB***, on va générer une commande de déplacement vers la gauche ou la droite (voir les chronogrammes ci-dessous).

- Concrètement, le graphe d'états est établi de la façon suivante.
  - On prend ***QA*** comme signal de référence. Une commande ***Rot_Left*** ou ***Rot_Right*** est générée uniquement si on a détecté un changement d'état sur ***QA.***
  - Le sens de rotation va alors dépendre de ***QB***
    - Si ***QA*** change d'état AVANT ***QB***, cela correspond à une rotation à gauche du codeur
      - On met ***Rot_Left*** à 1 pendant un cycle d'horloge
    - Si ***QA*** change d'état APRES ***QB***, cela correspond à une rotation à droite du codeur
      - On met ***Rot_Right*** à 1 pendant un cycle d'horloge

Chronogrammes à compléter pour trouver le graphe d'état



Description VHDL de la MAE – Simulation - Implémentation

- Vérifier que la fenêtre ***Sources for*** est en mode ***Simulation***

- Dans le menu ***Project***, cliquer sur ***New Source***. Choisir le type ***VHDL Module***, donner le nom ***move*** et vérifier que la case ***Add to Project*** est bien cochée. Cliquer sur ***Next***.

- Entrer les différentes entrées/sorties de **Move** comme sur la figure ci-dessus. Cliquer sur ***Next*** puis ***Finish***.

- Compléter les chronogrammes ci-dessus pour déterminer les états successifs du système.

- En déduire le graphe d'états de la MAE
    - **Faire valider ce graphe d'états par l'intervenant de TP.**

- Décrire ce graphe d'états en VHDL dans le fichier ***move.vhd***. Vérifier s'il n'y a pas d'erreur à l'aide de la fonction ***Check Syntax*** de Xilinx ISE.

- Lorsque votre MAE est décrite, il faut la simuler. Créer pour cela une nouvelle source de type ***VHDL Testbench***, associée au module ***move***

- Ecrire un testbench permettant de valider le fonctionnement de votre MAE.

- Lancer ***Modelsim*** et faire la simulation.

- Lorsque tout fonctionne bien, il faut instancier le module **Move** dans le fixhier ***ip_rotary.vhd***.
    - Vous pouvez vous inspirer pour cela de la façon dont a été instanciée dans ce même fichier le module **Rotary**.

- Il faut à présent implémenter sur la carte la nouvelle version de la console.

- Vérifier que la fenêtre ***Sources for*** est en mode ***Implementation***

- Lancer les étapes successives d'implémentation, puis programmer la carte sous ***Impact***, comme vous l'avez fait précédemment.

- Vérifier à présent que vous pouvez bien déplacer la raquette du jeu Casse-Briques à l'aide de l'encodeur rotatif.

SELECTION DU JEU

Nous allons à présent implémenter la fonction permettant de choisir le jeu de la console ***PMC-E201*** (Pong ou Casse-Briques)

Cette fonction va être réalisée à l'aide d'une MAE qui sera instanciée dans le module **Game**.

Présentation du module Game

Ce bloc contrôle le fonctionnement de la console ce qui consiste principalement à :

- Sélectionner le mode Console ou Manette pour la ***PMC-E201***
- Sélectionner le jeu actif (Pong ou Casse Briques)
- Activer ou désactiver le mode Pause
- Indiquer si la partie est gagnée ou perdue
- Transmettre à l'***IP VGA*** la couleur des objets que l'on souhaite afficher

- La gestion de toutes ces tâches est réalisée grâce à 4 sous-modules, qui sont instanciés dans le fichier ***game.vhd***

1) Display:
- Sélection de la couleur à afficher
- ✓ Ce bloc est déjà implémenté

2) Master Slave Manager
- Gestion du mode de la ***PMC-E201*** (Console / Manette)
- ✓ Ce bloc est déjà implémenté

3) Game Manager
- Sélection du jeu actif (Casse Briques / Pong)
- × Module restant à implémenter

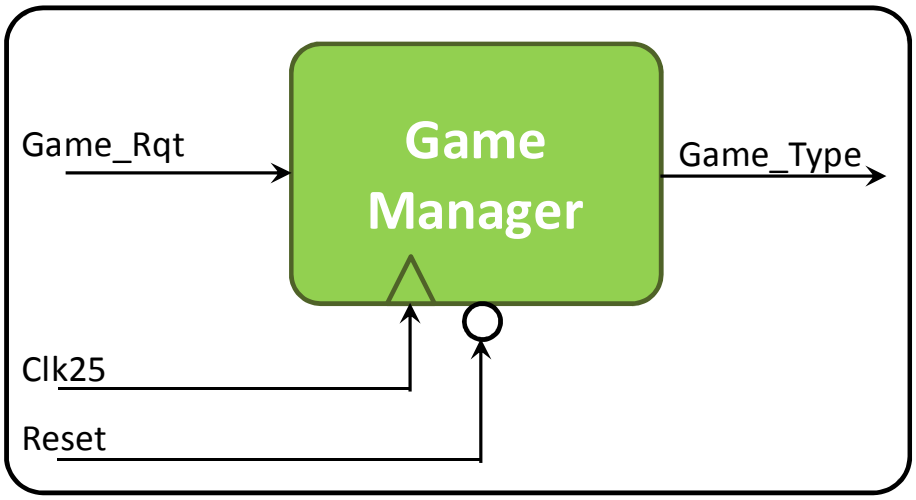
4) Mode
- Gestion de l'état de la partie (En cours, gagnée, perdue…)
- Gestion du mode Pause
- × Module restant à implémenter

- Pour le moment, la fonction correspondant au module **Game Manager** est réalisée par l'instruction ***Game_Type <= '0'.*** Cela explique pourquoi la console est bloquée sur le jeu Casse-Briques et qu'il n'est pas possible de passer au jeu Pong.
- Il faut donc remplacer cette instruction par une machine à états qui va générer le bon comportement du signal ***Game_Type***.

Présentation du Module Game Manager

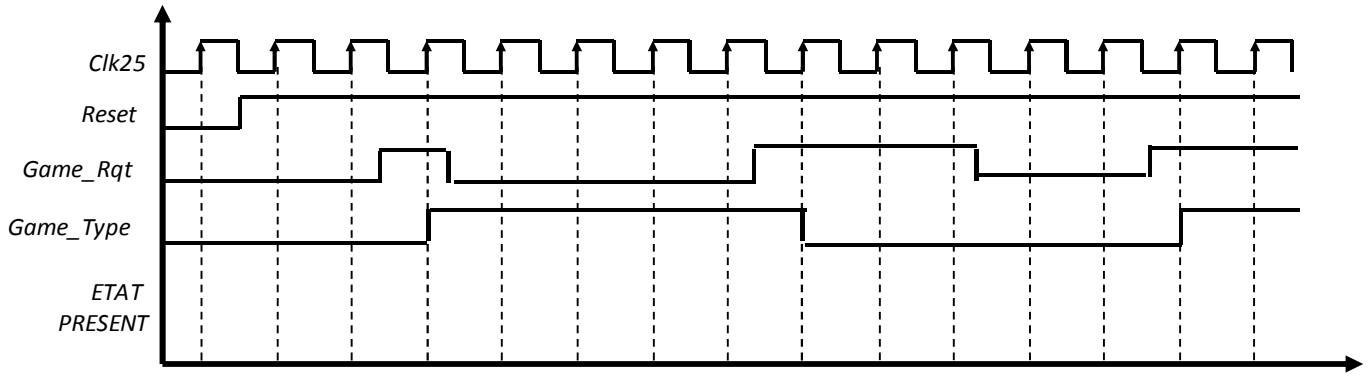
- Ce sous-module (structuré en Machine à Etats) fixe le jeu actif de la ***PMC-E201*** à l'aide du signal ***Game_Type***.
  - Casse Briques → ***Game_Type*** = 0
  - Pong → ***Game_Type*** = 1

- L'évolution du signal ***Game_Type*** dépend du comportement de l'entrée ***Game_Rqt***.
  - ***Game_Rqt*** est directement connectée aux boutons ***Nord*** et ***Est*** de la carte ***Spartan 3E***.



- Le comportement de ***Game_Type*** est décrit à l'aide d'une machine à états dont le comportement est le suivant (voir également le chronogramme ci-dessous) :
  - A l'état initial, le jeu actif est le Casse Briques
    - SI on détecte une demande de changement de jeu (***Game_Rqt***), la ***PMC-E201*** change de jeu
    - On va passer ainsi du Casse Briques à Pong, ou inversement.
    - Il faut veiller à ce qu'UN ET UN SEUL changement de mode soit effectué POUR CHAQUE REQUETE.
      - Une requête est terminée lorsque ***Game_Rqt*** est désactivée.

Chronogramme à compléter pour trouver le graphe d'état



Description VHDL de la MAE – Simulation - Implémentation

- Vérifier que la fenêtre ***Sources for*** est en mode ***Simulation***

- Dans le menu ***Project***, cliquer sur ***New Source***. Choisir le type ***VHDL Module***, donner le nom ***game_mgr*** et vérifier que la case ***Add to Project*** est bien cochée. Cliquer sur ***Next***.

- Entrer les différentes entrées/sorties du module **Game Manager** comme sur la figure ci-dessus. Cliquer sur ***Next*** puis ***Finish***.

- Compléter les chronogrammes ci-dessus pour déterminer les états successifs du système.

- En déduire le graphe d'états de la MAE
    - **Faire valider ce graphe d'états par l'intervenant de TP.**

- Décrire ce graphe d'états en VHDL dans le fichier ***game.vhd***. Vérifier s'il n'y a pas d'erreur à l'aide de la fonction ***Check Syntax*** de Xilinx ISE.

- Lorsque votre MAE est décrite, il faut la simuler. Créer pour cela une nouvelle source de type ***VHDL Testbench***, associée au module ***game_mgr***

- Ecrire un testbench permettant de valider le fonctionnement de votre MAE.

- Lancer ***Modelsim*** et faire la simulation.

- Lorsque tout fonctionne bien, il faut instancier le module **Game Manager** dans le fixhier ***game.vhd***.
    - Vous pouvez vous inspirer pour cela de la façon dont a été instancié dans ce même fichier le module **Master Slave Manager**.

- Il faut à présent implémenter sur la carte la nouvelle version de la console.

- Vérifier que la fenêtre ***Sources for*** est en mode ***Implementation***

- Lancer les étapes successives d'implémentation, puis programmer la carte sous ***Impact***, comme vous l'avez fait précédemment.

- Vérifier à présent que vous pouvez changer le jeu de la console en appuyant sur les bouton ***Nord*** ou ***Est***.

- Pour jouer à Pong :
    - A l'aide d'un câble RS232, connecter deux cartes ***Spartan 3E*** sur lesquelles ont été implémentées la console ***PMC-E201***. (brancher le câble sur le port série bleu de la carte).
    - Connecter une carte à un écran VGA et vérifier que les LEDs sont allumées. Cette carte sera la console principale
    - Sur l'autre carte, appuyer sur le bouton Sud et vérifier que les LEDs sont éteintes. Cette carte sera la manette pour le 2ème joueur du jeu Pong.
    - Enjoy…

GESTION DES JEUX

Nous allons à présent générer des signaux indiquant quand les parties de Pong ou Casse-Briques sont gagnées ou perdues. Ces signaux seront utilisés par le module d'affichage pour que l'écran devienne rouge ou vert selon les cas.

Nous allons également faire en sorte que le bouton de l'encodeur rotatif puisse mettre le jeu en pause.

Pour le moment, ces fonctions sont réalisées dans le module **Game**, à l'aide des instructions suivantes

```
pause <= not pause_rqt;
lost_game <= '0';
brick_win <= '0';
```

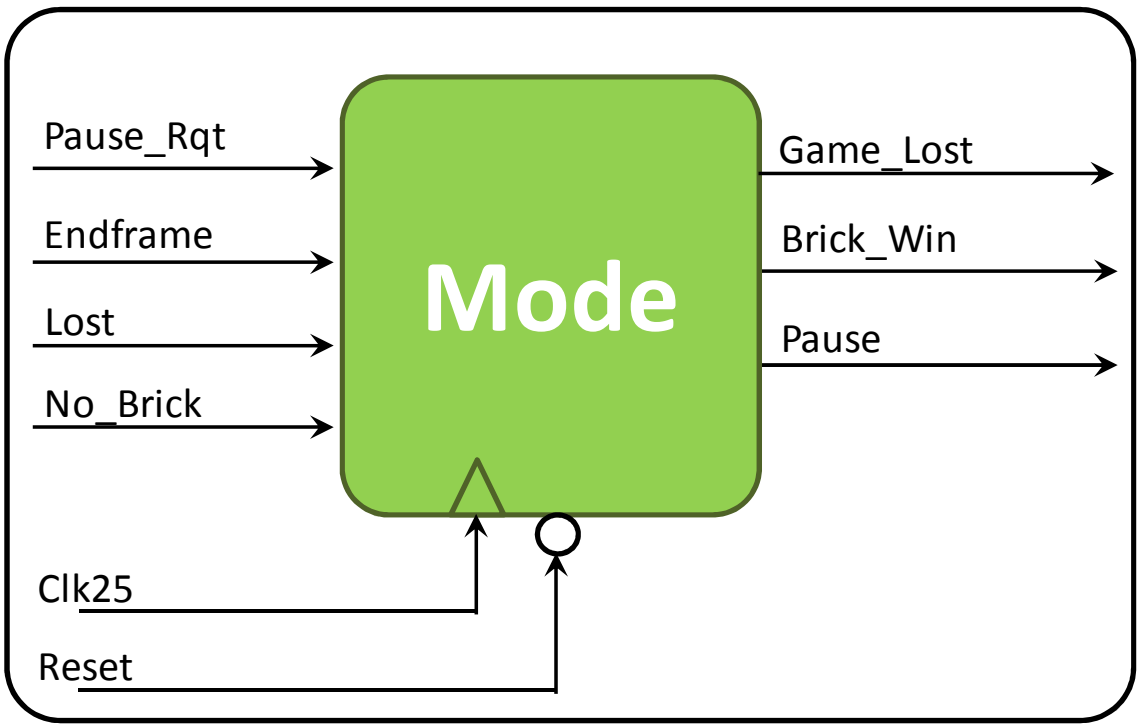
Vous devez à présent remplacer ces instructions par un module appelé **Mode** qui sera instancié dans le bloc **Game**.

Affectation du bouton de l'encodeur rotatif

Dans le fichier ***top.vhd***, dans l'instanciation du module **Game**, remplacer l'instruction connectant le port ***Pause_Rqt*** par celle mise en commentaire juste en dessous.

Présentation du Module Mode

- Ce sous-module détermine l'état de la partie en cours et gère le mode Pause de la console.



Entrées :

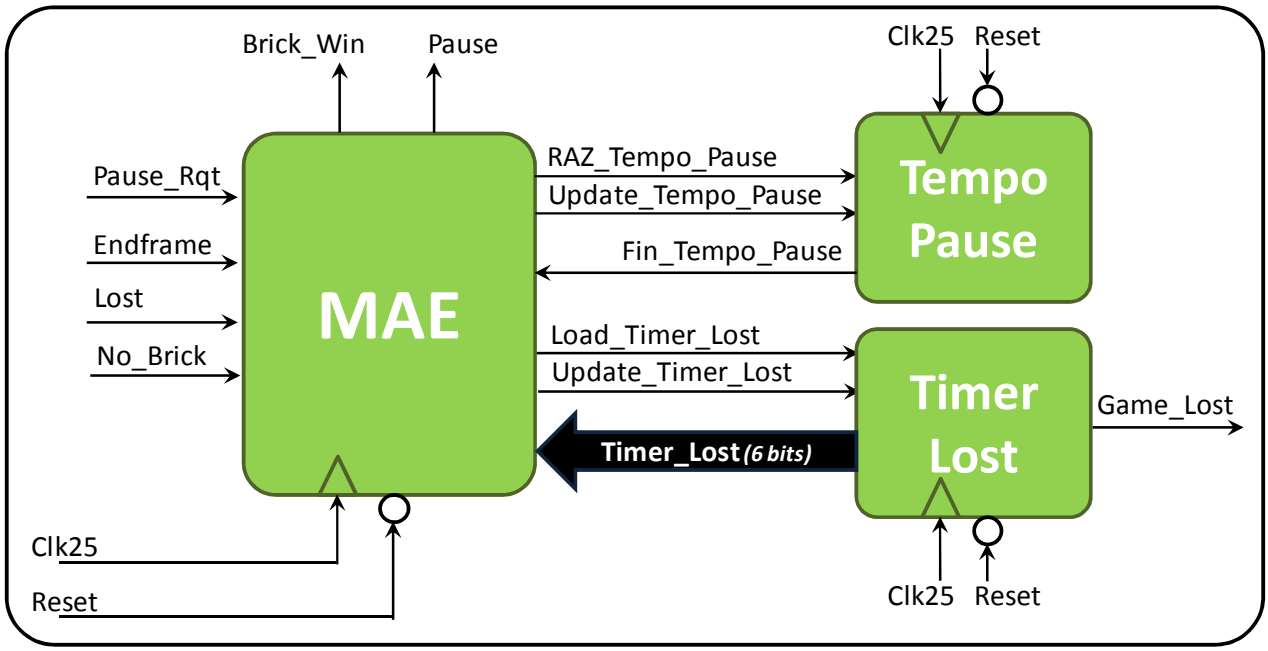
- ***Clk25*** : Horloge 25 MHz
- ***Reset*** : Reset asynchrone, actif à l'état bas
- ***Endframe***: Signal de fin de trame de l'image
- ***Pause_Rqt***: Demande de mise en pause du jeu ou de sortie de pause.
- ***Lost***: Indique que le pixel se trouve sur un des bords de l'écran
- ***No_Brick***: Indique que toutes les briques ont été détruites (jeu Casse Briques)

Sorties :

- ***Pause***: Indique si le jeu est en mode pause
- ***Game_Lost***: Signal indiquant que la partie est perdue (quand la balle est sortie de l'écran).
- ***Brick_Win*** : indique qu'une partie de Casse Briques est gagnée.

NB : Sauf indication contraire, tous les signaux sont actifs à l'état haut

Ce sous-module est lui-même composé de 3 blocs



1) **Tempo Pause:**
   - Compteur permettant de gérer l'anti-rebond du bouton poussoir de l'encodeur rotatif.
   - Utilisé pour la gestion du mode Pause de la ***PMC-E201***

2) **Timer Lost**
   - Compteur permettant de générer la sortie ***Game_Lost***

3) **MAE**
   - Machine à états pour gérer
     - Les sorties ***Brick_Win*** et ***Pause***
     - Les deux compteurs **Tempo Pause** et **Timer Lost**

- L'entrée ***No_Brick*** de la MAE est la sortie d'une porte ET prenant en entrée tous les bits de ***Brick_Bounce***.
  - ***No_Brick*** est activé si la balle a rebondi contre toutes les briques (donc si toutes les briques ont été détruites)

Fonctionnement des sous-modules Timer_Lost et Tempo_Pause

- **Tempo Pause** est un compteur 10 bits possédant les fonctionnalités suivantes
  - RAZ synchrone (si ***RAZ_Tempo_Pause*** est activée)
  - Incrémentation (si ***Update_Tempo_Pause*** est activée)
  - Maintien de la valeur précédente (si aucune commande n'est activée)

  - La sortie du compteur est comparée de façon combinatoire à la valeur maximale (tous les bits à 1). Si tel est le cas, la sortie ***Fin_Tempo_Pause*** est activée.

- **Timer Lost** est un compteur 6 bits possédant les fonctionnalités suivantes
  - Chargement parallèle à la valeur 63 (tous les bits à 1) (si ***Load_Timer_Lost*** est activée)
  - Décrémentation (si ***Update_Timer_Lost*** est activée)
  - Maintien de la valeur précédente (si aucune commande n'est activée)

  - La sortie du compteur est comparée de façon combinatoire à 0. Si la sortie est supérieure à 0, la sortie ***Game_Lost*** est activée.

Cahier des charges de la MAE

- La **MAE** a pour but de gérer :
  - Les sorties ***Brick_Win*** et ***Pause***
  - Les deux compteurs **Tempo Pause** et **Timer Lost**

- A l''état initial, le jeu est en Pause.
  - Le compteur de temporisation **Tempo Pause** est initialisé à 0.

- On reste en pause tant que l'on a pas de requête de changement du mode Pause
  - Si tel est le cas, on sort du mode Pause et on démarre la temporisation
  - ➢ Le jeu est alors en mode actif.

- On reste dans le mode actif tant que l'on a pas de requête de changement du mode Pause
  - Si tel est le cas, on rentre en mode Pause et on démarre la temporisation
  - ➢ Le jeu est de nouveau stoppé.

- Il faut veiller à ce qu'UN ET UN SEUL changement de mode soit effectué POUR CHAQUE APPUI sur le bouton.
  - ➢ Si l'utilisateur garde le bouton poussoir appuyé, il ne faut pas changer plusieurs fois le mode Pause.

- Quand le jeu est actif
  - Si on détecte que toutes les briques ont été détruites, on génère le signal ***Brick_Win*** indéfiniment.
  - SI on détecte que la partie est perdue,
    - On fait un chargement de **Timer Lost** puis on passe en mode Pause

- Quand **Timer Lost** a été chargé,
  - A chaque fin d'image (***Endframe***) et tant que la valeur du compteur **Timer Lost** est supérieure à 0,
    - On décrémente la valeur de **Timer_Lost**
    - ➢ Cela permettra au signal ***Game_Lost*** (voir descriptif de **Timer Lost**) de rester activé pour une durée de 64 images.

Description VHDL de la MAE – Simulation - Implémentation

- Dans le menu ***Project***, cliquer sur ***New Source***. Choisir le type ***VHDL Module***, donner le nom ***mode*** et vérifier que la case ***Add to Project*** est bien cochée. Cliquer sur ***Next***.

- Entrer les différentes entrées/sorties du module **Mode** comme sur la figure ci-dessus. Cliquer sur ***Next*** puis ***Finish***.

- Faire des schémas blocs des deux compteurs ainsi que le graphe d'états de la MAE, conformément au descriptifs donnés ci-dessus.
    - **Faire valider le graphe et les schémas blocs par l'intervenant de TP.**

- Décrire en VHDL la MAE et les deux compteurs dans le fichier ***mode.vhd***. Vérifier s'il n'y a pas d'erreur à l'aide de la fonction ***Check Syntax*** de Xilinx ISE.

- Lorsque votre module est décrit, il faut la simuler. Créer pour cela une nouvelle source de type ***VHDL Testbench***, associée au module ***mode***

- Ecrire un testbench permettant de valider le fonctionnement de votre module VHDL.

- Lancer ***Modelsim*** et faire la simulation.

- Lorsque tout fonctionne bien, il faut instancier le module **Mode** dans le fixhier ***game.vhd***.
    - Vous pouvez vous inspirer pour cela de la façon dont a été instancié dans ce même fichier le module **Master Slave Manager**.

- Il faut à présent implémenter sur la carte la nouvelle version de la console.

- Vérifier que la fenêtre ***Sources for*** est en mode ***Implementation***

- Lancer les étapes successives d'implémentation, puis programmer la carte sous ***Impact***, comme vous l'avez fait précédemment.

- Vérifier à présent que :
    - Le jeu se met en pause lorsque vous appuyez sur le bouton de l'encodeur rotatif.
    - Lorsque la balle sort de l'écran, celui-ci evien trouge pendant un bref instant.
    - Lorsque dans le jeu Casse-Briques, toutes les briques ont été détruites, l'écran devient vert et reste ainsi indéfiniment.

AJOUT D'UN OBSTACLE MOBILE

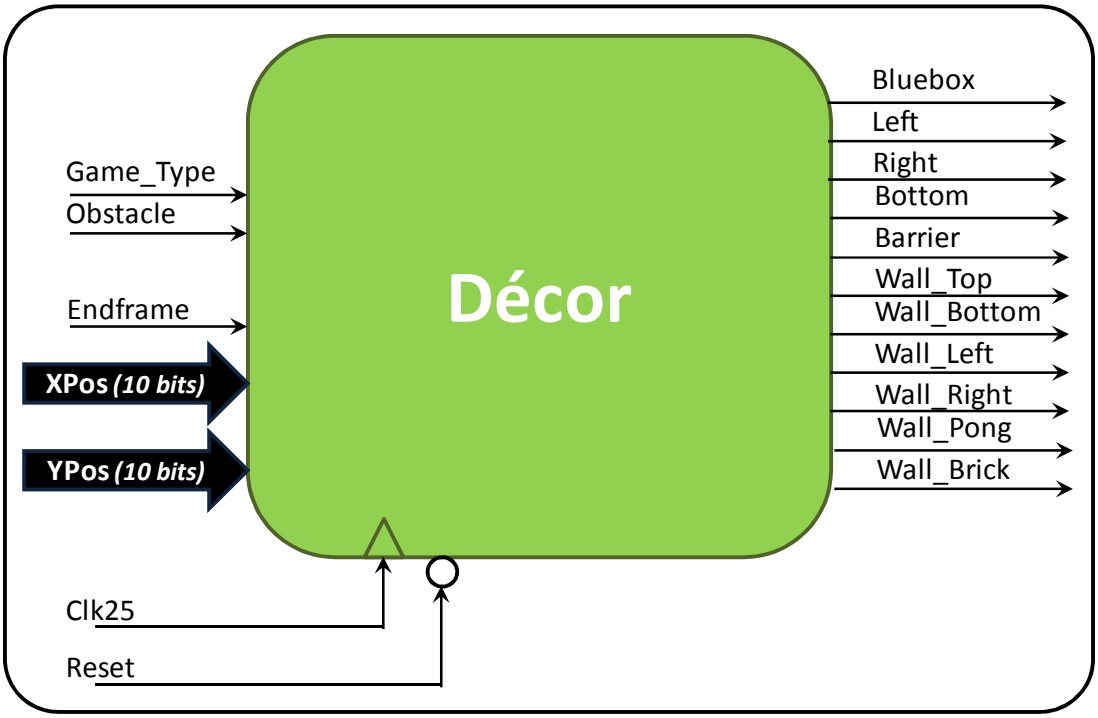
Nous allons à présent ajouter pour le jeu Pong un obstacle mobile au milieu de l'écran. En cas de rebond avec la balle, cet obstacle renverra la balle dans la direction opposée.

Le joueur pourra décider de la présence ou non de l'obstacle à l'aide de l'interrupteur S1.

L'obstacle sera implémenté dans le sous-module Décor (fichier ***decor.vhd***), qui est instancié dans le module Objects.

Présentation du Module Décor

Le sous-module **Décor** prend en entrée les coordonnées du pixel courant (celui qui est en cours d'affichage à l'écran) et détermine si celui-ci correspond à un des objets du décor (mur, case bleue, etc…) des jeux.



Entrées :

- ***Clk25*** : Horloge 25 MHz
- ***Reset*** : Reset asynchrone, actif à l'état bas
- ***Endframe***: Signal de fin de trame de l'image
- ***Game_Type***: Jeu actif (Casse Briques = '0' / Pong = '1')
- ***Obstacle***: Commande d'affichage de l'obstacle ('0' = pas d'obstacle / '1' = obstacle présent)
- ***XPos***: Abscisse du pixel courant (0 = gauche de l'écran / 639 = droite de l'écran)
- ***YPos***: Ordonnée du pixel courant (0 = haut de l'écran / 479 = bas de l'écran)

Sorties :

- ***Barrier***: Indique si le pixel courant appartient à l'obstacle
- Les autres signaux de sortie ne sont pas importants pour implémenter l'obstacle

Cahier des charges du signal *Barrier*

- L'obstacle a une dimension de 8 pixels de large et 100 pixels de haut
- Il se déplace verticalement de deux pixels à chaque image.
- Lorsqu'il atteint l'un des murs du haut ou du bas, son sens de déplacement est inversé.

- La commande ***Barrier*** est activée si :
    - Le jeu actif est Pong
    - La commande ***Obstacle*** est activée
    - Le pixel courant appartient à l'obstacle mobile.

- *NB : les rebonds contre l'obstacle sont déjà implémentés dans un autre module. Vous n'aurez donc pas à vous en occuper.*

- Pour vous aider à implémenter l'obstacle, 3 signaux internes sont déjà déclarés dans le module Décor.
    - ***XBarrier*** : signal actif si l'abscisse du pixel courant appartient à l'obstacle
        - NB : l'obstacle se trouve au milieu de l'écran et celui-ci fait 640 pixels de largeur
    - ***YBarrier*** : signal sur 9 bits. Indique l'ordonnée de la 1ère ligne de l'obstacle.
    - ***Direction*** : indique le sens de déplacement de l'obstacle.

Description VHDL de l'obstacle - Implémentation

- Ouvrir le fichier ***decor.vhd***. A l'endroit indiqué en commentaires, décrire en VHDL l'architecture permettant d'afficher l'obstacle à l'écran, conformément au cahier des charges ci-dessus.

- Il est conseillé, dans un premier temps, d'implémenter un obstacle fixe, puis dans un deuxième temps de le rendre mobile.

AMELIORATIONS

Vous pouvez à présent proposer et implémenter des améliorations pour la console ***PMC-E201***, que ce soit en terme d'affichage (modification de couleurs), de gestion des jeux ou de l'interface utilisateur.

Vous pourrez pour cela vous appuyer sur la documentation technique de la console ***PMC-E201***, disponible sur le disque Commun des machines.