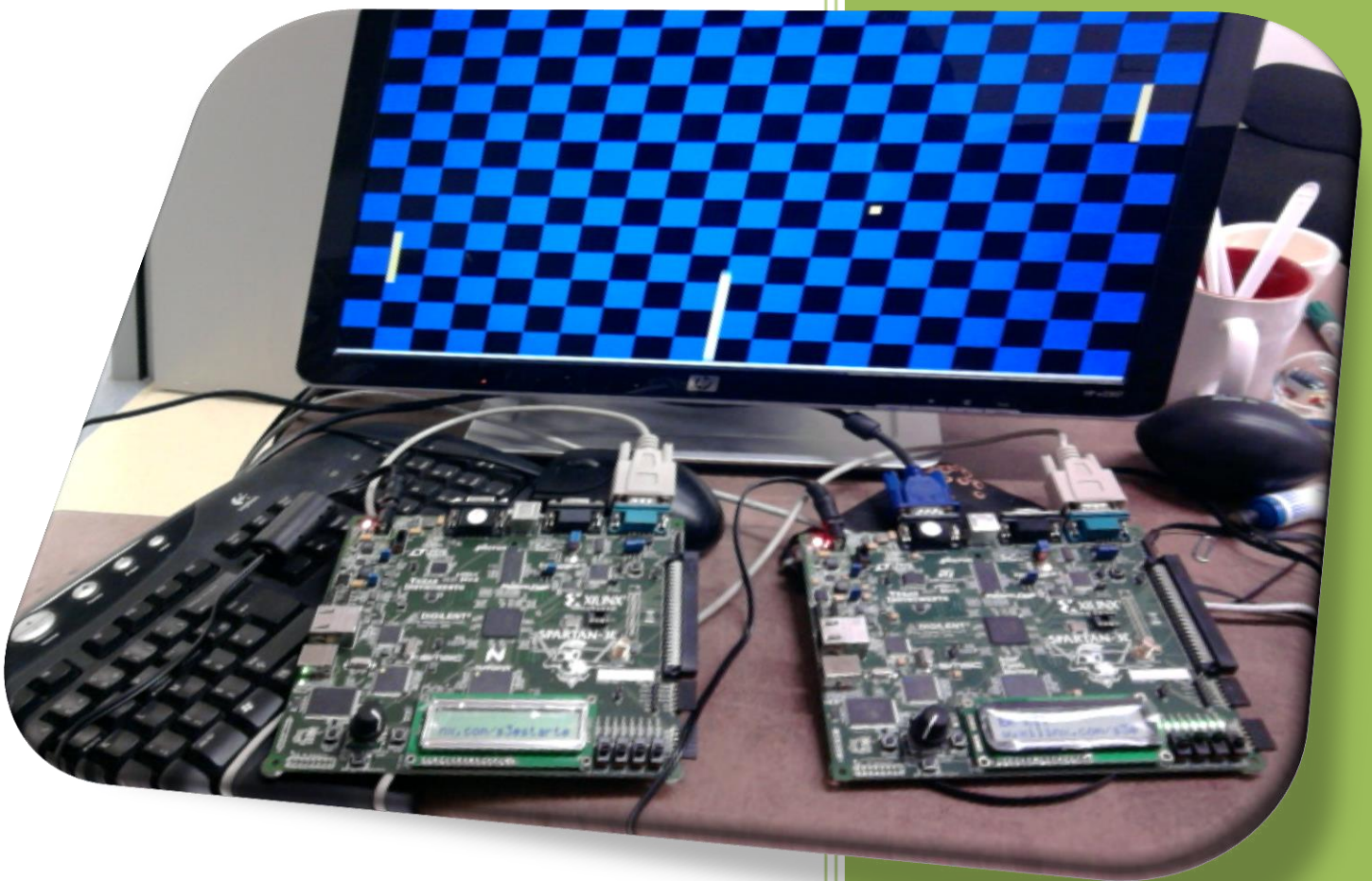


## Console SU-EE100 sur FPGA



## Enoncé Mini-Projet

# 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 Nexys 4 à l'aide du logiciel Vivado.



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-ÉTUDE

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 **SU-EE100**
- La présentation et les règles des jeux implémentés dans la **SU-EE100**
- L'interface utilisateur de la **SU-EE100** et son rôle pour chacun des jeux de la console.
- Une présentation de la carte FPGA **Nexys 4** sur laquelle est implémentée la console.
- Une description générale ainsi qu'un synopsis de l'architecture interne du FPGA.

## Travail à effectuer AVANT la 1<sup>ère</sup> séance de TP

- Lire attentivement le document « Manuel Utilisateur + Carte FPGA »
- Noter les questions que vous pourriez avoir suite à cette lecture.
  - o Vous pourrez les poser à l'intervenant de TP.
- Spécifier l'architecture et le code VHDL du diviseur d'horloge comme indiqué.
  - o 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 :

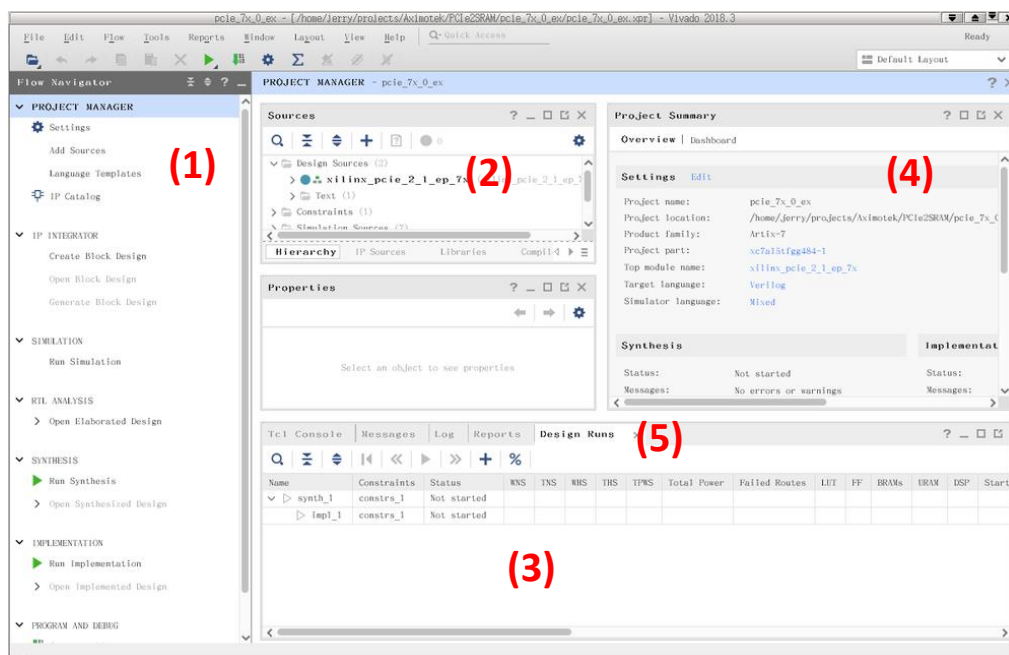
- Créer un projet dans Xilinx Vivado et ajouter à ce projet les sources VHDL de la console **SU-EE100**
- Coder un module simple de division d'horloge et l'ajouter au projet
- Simuler ce module de division d'horloge avec **Modelsim**
- Implémenter la console sur la carte FPGA.

## Création d'un projet avec le logiciel Vivado

- Ouvrir le logiciel Xilinx Vivado. (**Menu Démarrer → Tous les Programmes → Xilinx Design Tools → Vivado 2018.3 → Vivado 2018.3**).
- Cliquer sur **Create New Project** puis **Next**. Choisir un nom pour le projet et choisir comme répertoire **C:/Users/1234567** (si 1234567 est votre numéro d'étudiant). Puis cliquer deux fois sur **Next**.
- Dans la fenêtre suivante, sélectionner **VHDL** comme **Target Language** puis cliquer trois fois sur **Next**.
- Dans la fenêtre suivante pour choisir le FPGA, sélectionner les filtres suivants :
  - o Family : **Artix-7**
  - o Sub-Family : **Artix-7**
  - o Package : **CSG324**
  - o Speed : **-3**

Sélectionner le FPGA **xc7a100** puis cliquer sur **Next** puis **Finish**.

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



- (1) **Flow Navigator** : Permet de démarrer les différentes phases du flot de conception, de la création des sources VHDL à la simulation, l'implémentation et jusqu'à la programmation du FPGA.
- (2) **Sources** : Liste les différentes sources (VHDL, Testbenches, Fichier de contraintes, etc...) du projet ainsi que leur hiérarchie.
- (3) **Console/Messages** : Indique les messages générés par Vivado. Parmi les onglets disponibles (5), **Reports** permet d'accéder aux rapports de synthèse et d'implémentation.
- (4) **Editeur** : Permet d'éditer les sources du projet ou de consulter les rapports d'implémentation.

## Quelques mots sur les outils de simulation...

- Vivado dispose d'un simulateur VHDL, plus complet que celui d'EDA Playground, notamment pour ce qui est de la simulation des machines à états.
- Le fonctionnement du simulateur sera décrit dans les sections suivantes.

## Ajout des sources VHDL de la console SU-EE100

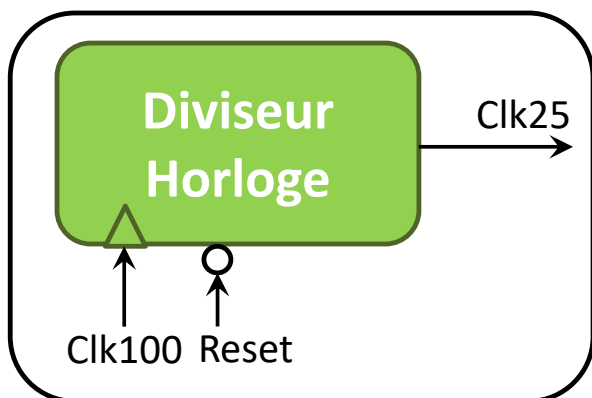
- En allant sur le site Moodle de l'UE, dans la section TP, récupérer l'archive **Sources\_Debut\_TP.zip** qui contient les codes VHDL de la console. Extraire ces fichiers de l'archive dans un répertoire temporaire
  - o Les sources proposent une version incomplète de la console **SU-EE100**, qu'il va falloir à présent compléter.
  - o Cette version suit globalement le synopsis que vous avez analysé lors de la pré-étude.
- Dans le **Flow Navigator/Project Manager**, cliquer sur **Add Sources**. Choisir **Add or Create Design Sources**, puis cliquer sur **Next**. Cliquer sur **Add Files**, aller à l'endroit où vous avez sauvegardé les sources de la console, puis sélectionner tous les fichiers du répertoire **VHDL**. Vérifier que la case **Copy Sources into Project** est bien cochée. Cliquer sur **OK** puis sur **Finish**.
  - o NB : Les sources VHDL ont été copiées dans le répertoire du projet Vivado.
- Dans la fenêtre **Sources**, les fichiers VHDL apparaissent dans le répertoire **Design Sources** et sont organisés de façon hiérarchique.
  - o En double cliquant sur le module top, vous verrez la hiérarchie complète s'afficher.
  - o Les noms indiqués correspondent aux modules VHDL instanciés dans l'architecture de **top.vhd**
  - o Parmi les modules, vous verrez à gauche du bloc **Clk25MHz** un point d'interrogation.
    - Cela indique que le fichier VHDL correspondant à ce module est manquant
    - Il s'agit du diviseur d'horloge que vous deviez réaliser avant la première séance et que nous allons à présent ajouter.



## Ajout d'un module de division d'horloge

L'horloge de la carte **Nexys 4** est cadencée à 100 MHz. En revanche, la console **SU-EE100** a besoin de deux horloges, cadencées pour l'une à 25 MHz et pour l'autre à 25 Hz. L'horloge 25 MHz n'est pour le moment pas présente dans le code VHDL de la console. Il faut donc la générer à l'aide d'un module VHDL de division d'horloge que vous allez d'abord décrire et simuler dans Modelsim, puis ajouter au projet dans Vivado.

- Ouvrir Modelsim et ouvrir le fichier VHDL où vous avez décrit le diviseur d'horloge
  - o Pour rappel, l'entité doit **OBLIGATOIREMENT** s'appeler **ClkDiv**, et **les ports d'entrée/sortie devront être les suivants (bien respecter les noms également)**



### Entrées :

- **Clk100** : Horloge 100 MHz fournie par la carte
- **Reset** : Reset asynchrone (Actif à l'état bas)

### Sortie :

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

- Compiler le code et corriger les éventuelles erreurs.
- Utiliser le testbench **tb\_clkdiv.vhd** disponible parmi les sources de la console, dans le répertoire **TESTBENCH**.
  - o Lancer la simulation. Vérifier le bon fonctionnement du module.
- Retourner dans Vivado et ajouter votre diviseur d'horloge à votre projet.
  - o Le point d'interrogation à côté du module **Clk25MHz** aura normalement disparu.

## Implémentation sur la carte FPGA

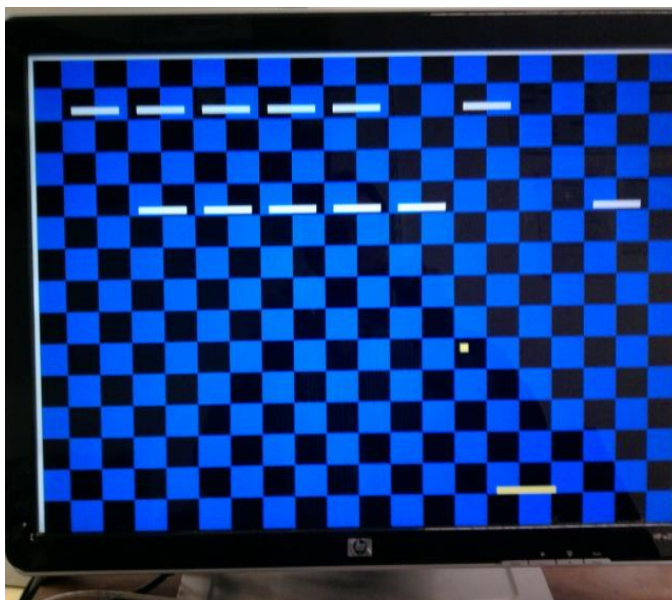
Nous allons procéder à l'implémentation de la console sur la carte **Nexys 4**.

- Cliquer sur **Add Sources** puis choisir **Add or Create Constraints**. Cliquer sur **Add File** puis aller à l'endroit où vous avez sauvegardé les sources de la console. Dans le répertoire **XDC**, choisir le fichier correspondant à votre version de la carte FPGA (**Nexys4** ou **Nexys4 DDR**). Cliquer sur **OK** puis sur **Finish**.
  - o Le fichier **XDC** a pour but de connecter les entrées/sorties du code VHDL à celles de la carte FPGA.
  - o Pour visualiser le fichier **XDC**, aller dans la fenêtre **Sources**, au répertoire **Constraints**
- Dans le **Flow Navigator**, cliquer sur **Run Synthesis** puis dans la fenêtre qui s'ouvre cliquer sur **OK**.
  - o Cette procédure analyse les sources VHDL et les transforme en cellules logiques de base.
  - o NB : L'onglet **Project Summary** dans la fenêtre **Editeur** vous indiquera les caractéristiques de l'architecture implémentée (% d'utilisation des ressources du FPGA, performances temporelles, estimation de la consommation...).
- Lorsque la synthèse est terminée, cliquer sur **Run Implementation**
  - o Au cours de cette phase, Vivado va placer puis router dans le FPGA les cellules logiques identifiées lors de la synthèse, conformément aux contraintes données par le fichier **XDC**.





- A la fin de l'implémentation, cliquer sur **Generate Bitsream** puis sur **OK** pour générer le fichier de configuration du FPGA.
- Cliquer ensuite sur **Open Hardware Manager** pour programmer le FPGA.
  - o Connecter la carte au port USB du PC et allumer la carte avec l'interrupteur ON/OFF
  - o Dans le bandeau vert en haut de l'écran, cliquer sur **Open New Target**.
  - o Cliquer autant de fois que nécessaire sur **Next** puis **Finish**.
  - o A nouveau dans le bandeau vert, cliquer sur **Program Device**. Vérifier que le chemin du bitstream (Test.bit) est bien spécifié avant de lancer la programmation du FPGA..
    - Si ce n'est pas le cas, le bitstream se trouve à l'endroit suivant :  
C:\Users\1234567\My\_Project\MyProject.runs\impl\_1\top.bit
- Le FPGA est à présent programmé.
  - o Connecter la sortie VGA de la carte **Nexys 4** à l'écran de votre PC.
  - o Connecter l'encodeur rotatif au connecteur **PMOD B** de la carte, en vous assurant que la pastille de couleur du connecteur est bien tournée de votre côté.
  - o Positionner l'interrupteur **S0** vers le haut
  - o Vous devez normalement voir le jeu Casse-Briques s'afficher
- Si vous positionnez l'interrupteur **S15** vers le haut et que vous désactivez la pause (switch **S1**), vous devez pouvoir faire bouger la raquette en inclinant la carte d'un côté ou de l'autre.
- 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
  - Si vous positionnez l'interrupteur **S15** vers le bas et que vous désactivez la pause (switch **S1**), 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 de l'écran.
    - o Normalement, l'écran devrait devenir rouge et une nouvelle partie devrait débiter.
  - Le switch **S1** permet de mettre le jeu en pause.
    - o Cela devrait être le rôle du bouton central.
- Dans les étapes suivantes, vous allez ajouter ces fonctionnalités manquantes pour réaliser complètement le cahier des charges de la **SU-EE100**.
- Avant cela, fermer le **Hardware Manager**, en cliquant sur la croix à droite du bandeau bleu.

# TACHE 1 – CONTROLEUR VGA

## AVEC COULEURS SUR 4 BITS

L’affichage des jeux de la console **SU-EE100** est réalisé à l’aide d’un **contrôleur VGA** qui code les couleurs sur 3 bits (1 bit pour le rouge, 1 bit pour le vert et 1 bit pour le bleu. Mais la sortie VGA de la carte **Nexys** permet de coder chaque couleur primaire sur 4 bits, permettant ainsi de générer 4096 couleurs différentes, au lieu des 8 couleurs possibles avec la version actuelle de la console.

Il devient ainsi possible d’améliorer les possibilités graphiques des jeux, et également de pouvoir mettre en place une variation au fil du temps des teintes des objets du jeu. On souhaite donc ajouter à la console un **contrôleur VGA** avec sortie de chaque couleur sur 4 bits.

Les codes VHDL et XDC pour cette tâche se trouvent dans le répertoire **TACHE\_1** de l’archive **Sources\_Debut\_TP.zip** que vous avez téléchargée précédemment.

### Création d’un Projet de Travail

Créer un nouveau projet **Vivado** avec les mêmes paramètres que le projet de la Console.

Dans les sources de la **Console SU-EE100**, faire une copie du **Diviseur d’Horloge** et du **Contrôleur VGA (VGA.vhd)** sur le bureau. Renommer la copie du fichier **VGA.vhd** en **VGA\_4 bits.vhd**

Ajouter ces deux fichiers à votre projet. Ajouter également au projet les sources **Top.vhd** ainsi que le fichier de contraintes **.xdc** du répertoire **TACHE\_1**.

- Le fichier **Top.vhd** instancie le diviseur d’horloge et le **Contrôleur VGA 4 bits**.
- Pour construire la version 4 bits de ce module, nous allons travailler à partir de la version 1 bit et faire les modifications nécessaires.
- Le fichier **.xdc** connecte les 12 bits correspondant aux valeurs des 3 couleurs R,G,B (rouge, vert, bleu) sur les 12 interrupteurs de droite de la carte **Nexys**. Ouvrir le fichier **.xdc** pour voir l’association entre couleurs et interrupteurs.

### Mise à jour du Contrôleur VGA – Implémentation

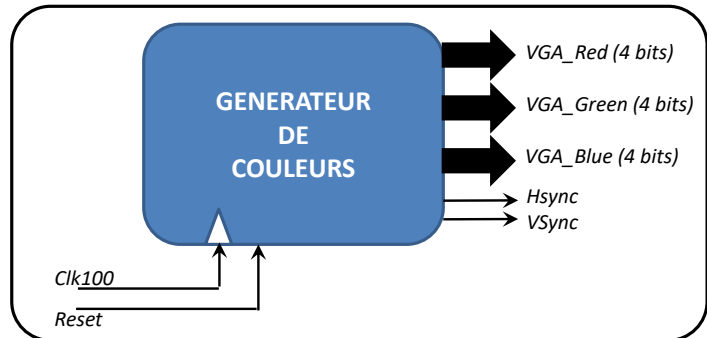
- Modifier le code de **VGA.vhd** en passant toutes les couleurs sur 4 bits.
- On n’oubliera pas également de changer le nom de l’entité, conformément au nom indiqué dans **Top.vhd**.
  - Vous pouvez vérifier que le nom de l’entité est conforme en regardant dans la fenêtre **Sources** que le contrôleur VGA est rattaché à Top dans la hiérarchie du projet.
- Il faut à présent implémenter sur la carte cette architecture.
  - Dans le **Flow Navigator**, cliquer directement sur **Generate Bitsream**
    - Cela lancera également les étapes **Run Synthesis** et **Run Implementation**
- Ouvrir à nouveau le **Hardware Manager** et reprogrammer la carte FPGA comme précédemment.



- Vérifier à présent en jouant avec les interrupteurs que vous pouvez changer le niveau d'intensité de chaque couleur rouge, verte ou bleue.

## Ajout d'un Générateur de Couleurs

On va maintenant ajouter un module permettant de générer une série de couleurs sur 12 bits. Ce module sera ajouté ensuite à la console. La vue externe du système est indiquée ci-contre :



## Architecture du Système

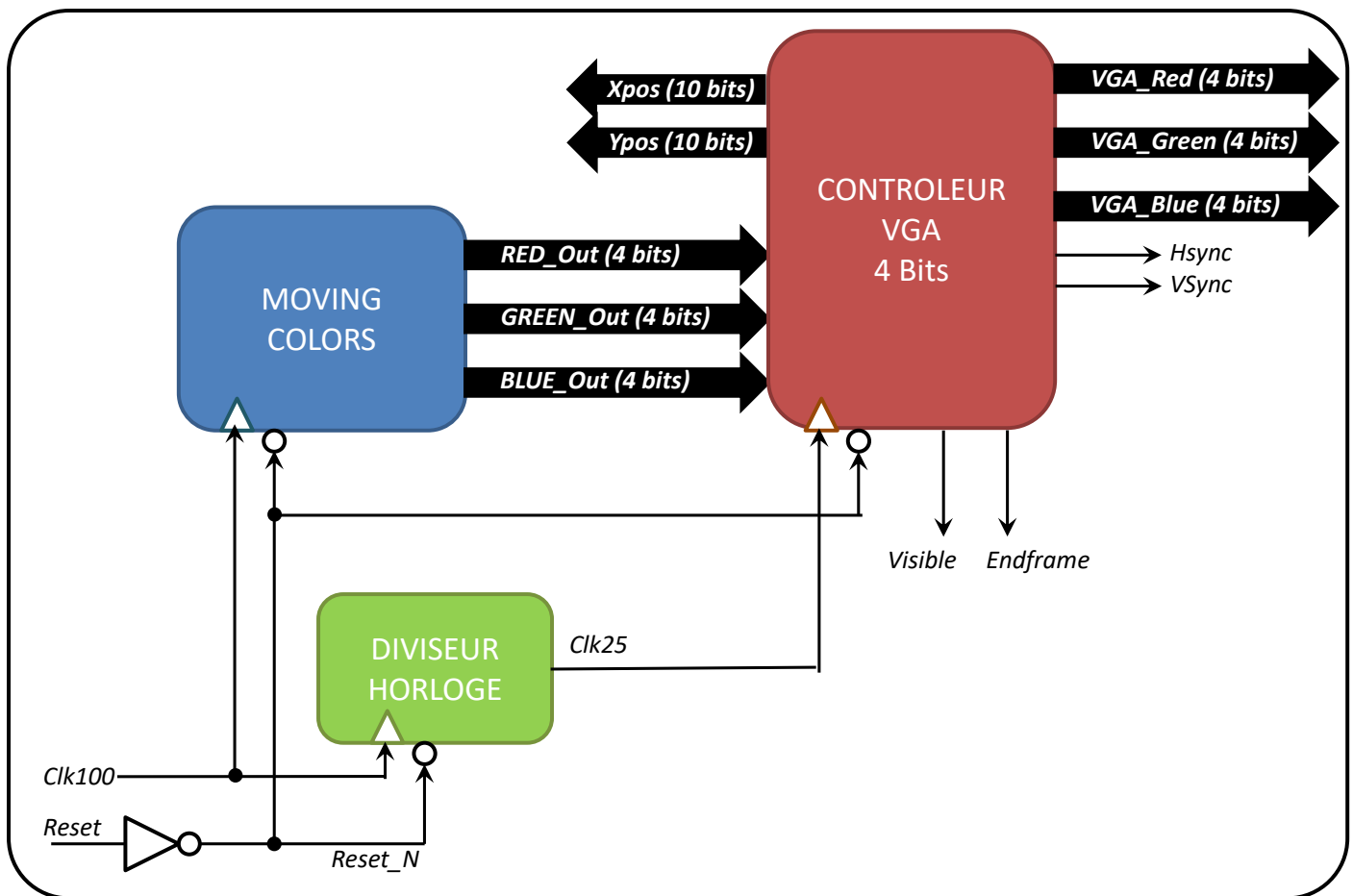
La vue interne du système est donnée dans le schéma ci-dessous. Le générateur de couleurs est composé de trois modules, qui doivent être instanciés dans le module **Top.vhd**.

- 1) **Diviseur d »horloge**
- 2) **Contrôleur VGA 4 Bits**
- 3) **Moving Colors**

- Ce module va générer une teinte de couleur sur 12 bits qui va évoluer périodiquement.

Il faut donc décrire le module **Moving Colors**, l'ajouter au système et tester l'ensemble du système.





## Module Moving Colors

Le module **Moving Colors** comprend trois parties

- 1) Un **diviseur d'horloge** qui va générer une horloge qui va cadencer le changement de teinte en sortie du module
  - Pour obtenir un résultat visible, l'horloge générée doit avoir une **fréquence de 20 Hz**.
  - Cependant, pour faciliter les simulations de l'architecture, nous considérerons que le diviseur générera une horloge de **fréquence 10 MHz** à partir de l'horloge 100 MHz.
  - Il faudra repasser avec une horloge de 20 Hz en sortie pour l'implémentation.
- 2) Trois **compteurs 5 bits**
  - Chaque **compteur** va représenter la teinte de rouge, de vert ou de bleu à envoyer en sortie
  - Les **compteurs** sont cadencés par l'horloge issue du **diviseur d'horloge** (20 Hz ou 10 MHz en simulation)
  - Les **compteurs** peuvent être réinitialisés de manière asynchrone.
    - Le **compteur Rouge** sera initialisé à "11111".
    - Les **deux autres compteurs** seront initialisés à "00000".
  - Chaque **compteur** peut être individuellement commandé de manière synchrone pour :
    - **S'incrémenter**
    - **Se décrémenter**
    - **Rester à sa valeur actuelle.**
  - Les 4 MSB de chaque **compteur** seront transmis en sortie sur **RED\_Out**, **GREEN\_Out**, et **BLUE\_Out**.



### 3) Une machine à états

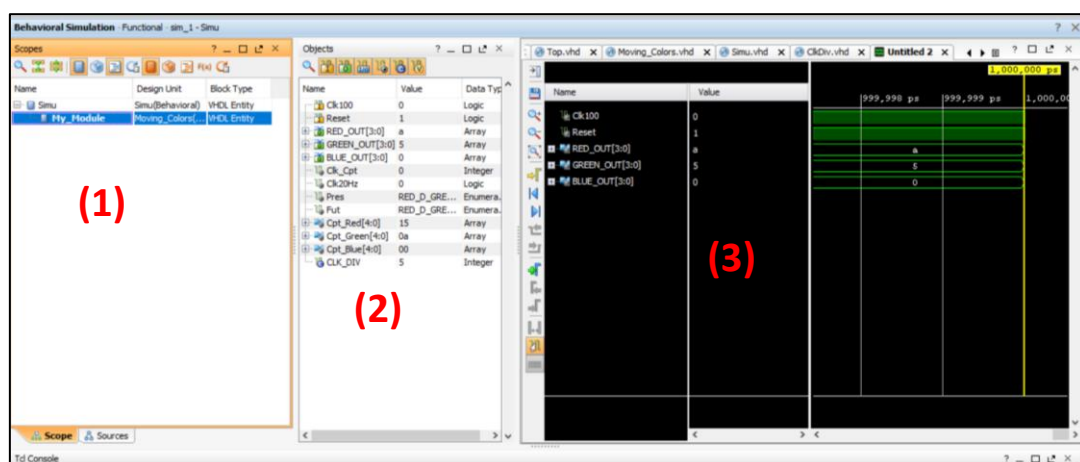
- La **machine à états**, cadencée par l'horloge à 100 MHz, a pour but de commander les **compteurs** de teinte pour générer l'évolution suivante :

Phase	Teinte Rouge	Teinte Verte	Teinte Bleue
1	Diminution	Augmentation	Constante
2	Constante	Diminution	Augmentation
3	Augmentation	Constante	Diminution




- Le changement de phase s'opère lorsque le compteur de la teinte que l'on a augmentée a atteint sa valeur maximale.
- Quand on a terminé la phase 3, on revient en phase 1 pour rejouer la séquence de teintes.

## Description VHDL – Simulation - Implémentation

- Dans le **Flow Navigator**, partie **Project Manager**, cliquer sur **Add Sources** pour créer une nouvelle **Design Source** appelée **Moving\_Colors.vhd** et y décrire l'architecture présentée ci-dessus.
  - Vous pouvez faire valider votre graphe d'états par l'encadrant.
- Ajouter une nouvelle source, cette fois de type **Simulation Source** appelée **Simu.vhd**. Ne pas mettre d'entrées/sorties dans ce module.
  - Instancier dans ce **testbench** le module **Moving\_Colors**
  - Décrire le comportement de l'horloge (d'une fréquence de 100 MHz) et du Reset asynchrone.
- Dans la fenêtre **Sources** de **Vivado**, dérouler la partie **Simulation Sources** et cliquer avec le bouton droit sur votre fichier **Simu.vhd** et cliquer sur **Set as Top**.
  - Cela va permettre au simulateur de se lancer en exécutant ce **testbench**.
- Dans le **Flow Navigator**, partie **Simulation**, cliquer sur **Run Simulation** puis **Behavioral Simulation** pour lancer le simulateur.
  - S'il n'y a pas d'erreur, le simulateur s'ouvre sur cette fenêtre. Il a également démarré la simulation du **testbench**, en faisant avancer le temps de simulation de 1  $\mu$ s.
  - Le simulateur est composé de 3 fenêtres :



- (1) Donne la hiérarchie du système simulé.

- (2) Donne la liste des ports et signaux d'un module. En cliquant sur un module dans (1), la liste des signaux de (2) se met à jour. Cela permet de visualiser le comportement de signaux internes. Pour visualiser un signal dans le chronogramme, faites-le glisser dans (3)
- (3) Chronogramme de simulation. Les icônes à gauche permettent de régler le niveau de zoom et de configurer des curseurs. Il est possible de changer la base de représentation d'un signal en cliquant sur son nom avec le bouton droit et en sélectionnant **Radix**.
  - Les icônes au-dessus de ces fenêtres assurent la gestion de la simulation. En particulier :
    -  permet de réinitialiser la simulation (sans recompiler les fichiers sources VHDL)
    -  fait avancer la simulation d'un temps spécifié dans le cadre (donc ici 10 µs)
    -  relance la simulation (en recompilant les sources suite par exemple à une modification)
- Simuler le système complet en vérifiant la bonne succession des états de la **MAE** et le bon comportement des sorties RGB. Quitter le simulateur lorsque tout est validé.
  - Repasser la fréquence d'horloge du diviseur à 20 Hz en prévision de l'implémentation.
- Modifier **Top.vhd** pour y instancier le module **Moving\_Colors**, conformément au schéma global de l'architecture ci-dessus.
- Mettre à jour le fichier **.xdc** en mettant en commentaires les broches des interrupteurs, qui ne sont plus utilisées à présent.
- Lancer l'implémentation et la génération du bitstream. Programmer la carte FPGA et vérifier sur l'écran VGA que désormais, vous pouvez voir un dégradé de couleurs qui change en permanence.

## Intégration sur la Console SU-EE100

Retourner dans le projet de la Console. Ajouter au projet les sources **Moving\_Colors.vhd** et **VGA\_4bits.vhd**

Dans le fichier **Top.vhd**, apporter les modifications suivantes :

- Mettre en commentaires l'instanciation du **Contrôleur VGA** (ligne 351 et suivantes), ainsi que les 3 instructions fixant la valeur des 3 LSB de **VGA\_Red**, **VGA\_Green** et **VGA\_Blue**.
- Décommenter l'instanciation du **Contrôleur VGA 4 bits** (ligne 376 et suivantes), celle du module **Moving\_Colors** (ligne 397 et suivantes), ainsi que les 3 instructions fixant la valeur de **VGA\_Red\_i**, **VGA\_Green\_i** et **VGA\_Blue\_i**.
- Relancer l'implémentation de la console. Après avoir programmé la carte, que constatez-vous ?



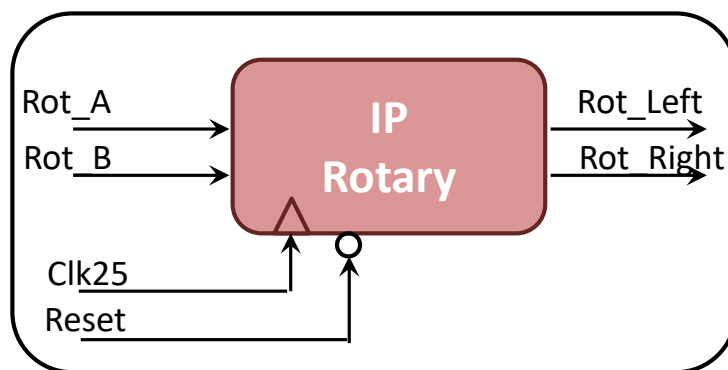
# TACHE 2 – GESTION DE LA RAQUETTE AVEC L'ENCODEUR ROTATIF

Dans cette partie, nous allons réaliser une IP (Intellectual Property) permettant de gérer l'encodeur rotatif connecté à la carte **Nexys 4** 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 connecté à la carte **Nexys 4**. Il génère les commandes de déplacement **Rot\_Left** et **Rot\_Right** (rotation vers la gauche ou vers la droite). 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*

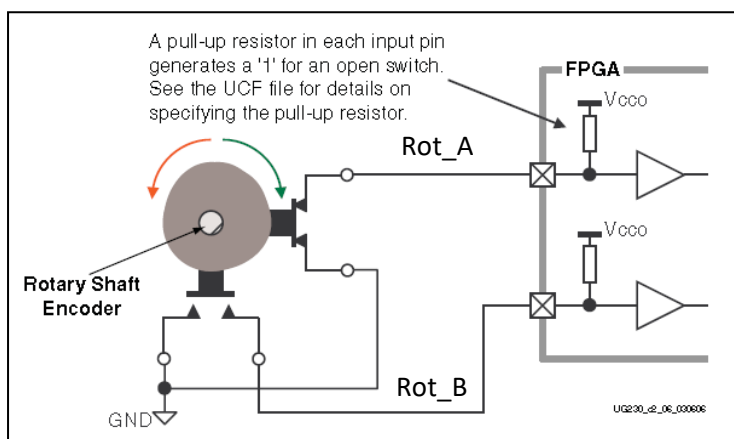


### Entrées :

- **Clk25** : Horloge 25 MHz
- **Reset** : Reset asynchrone, actif à l'état bas
- **Rot\_A** : 1<sup>er</sup> Interrupteur de l'encodeur
- **Rot\_B** : 2<sup>ème</sup> Interrupteur de l'encodeur
- **Rot\_Button** : Bouton de l'encodeur

### Sorties :

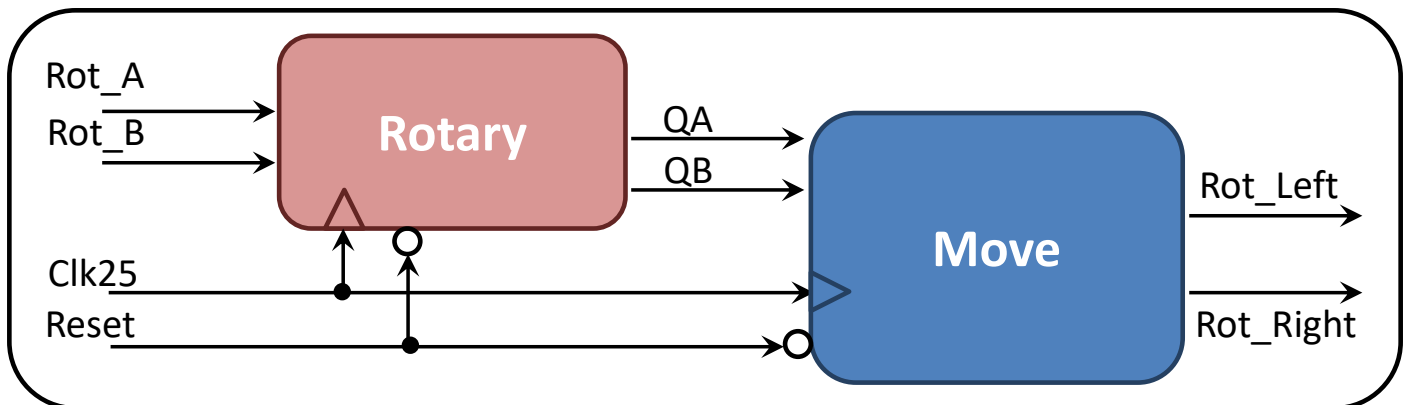
- **Rot\_Left** : Commande de rotation à gauche
- **Rot\_Right** : Commande de rotation à droite
- **Press** : Commande d'appui sur le bouton



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**



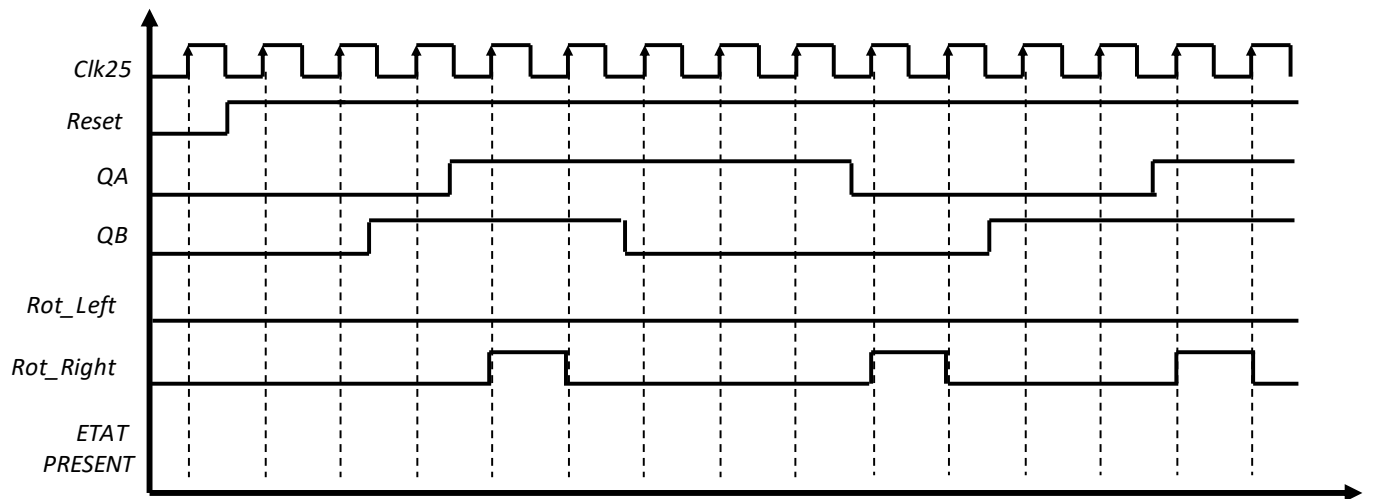
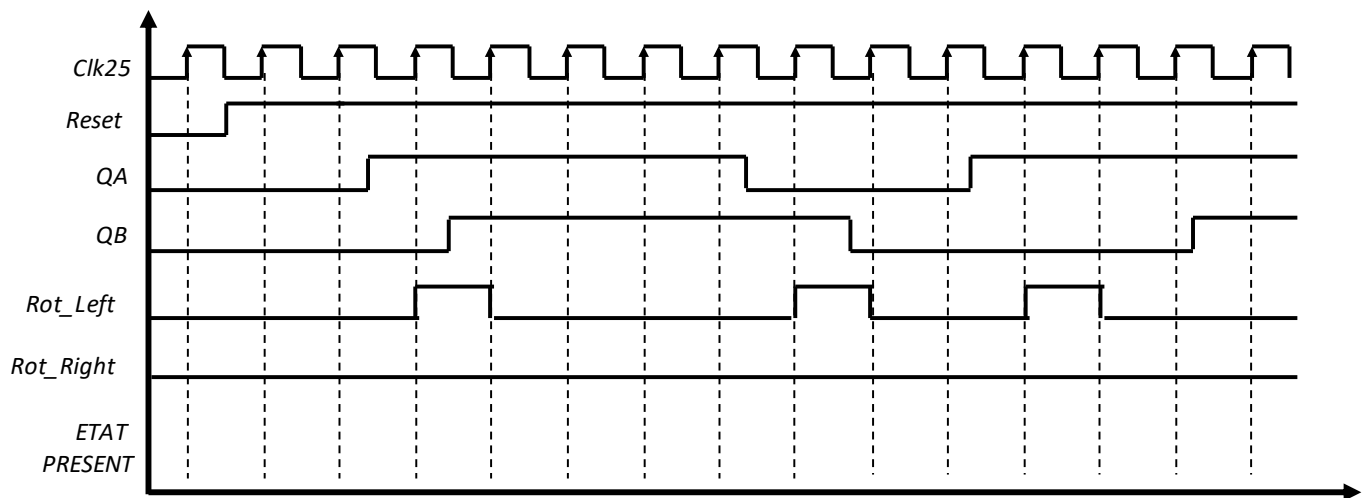
- Dans l'état actuel de la console **SU-EE100**, le module **Rotary** est déjà implémenté (fichier **rotary.vhd**).
- Ouvrir le fichier **ip\_rotary.vhd**. On peut y trouver l'instanciation du module **Rotary**. On y trouve également deux instructions forçant les sorties **Rot\_Left** et **Rot\_Right** 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

- 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.
- Dans **Vivado**, ouvrir le projet de la **Console SU-EE100**, créer un fichier **move.vhd** et y décrire ce graphe d'états en VHDL.
- Lorsque votre MAE est décrite, il faut la simuler. Créer pour cela un testbench VHDL en l'ajoutant au projet dans les **Simulation Sources** (ne pas oublier de le mettre en top-level de la simulation en cliquant avec le bouton droit sur le fichier, puis sur **Set as Top**).
  - Simuler et vérifier le bon fonctionnement de votre MAE.
- Il faut à présent insérer le module dans la hiérarchie du projet de la Console.
  - En effet, vous pourrez constater dans la fenêtre **Sources** qu'à ce stade, **move.vhd** n'est pas rattaché à la hiérarchie du reste du système.
  - Pour cela, il faut instancier la MAE de **move.vhd** dans le fichier **ip\_rotary.vhd**, à l'endroit indiqué dans le code.
    - Vous pouvez vous inspirer pour cela de la façon dont a été instanciée dans ce même fichier le module **Rotary**.
    - Après modification d'**ip\_rotary.vhd**, le fichier **move.vhd** prendra sa place dans la hiérarchie de la console.
- Il faut à présent implémenter sur la carte la nouvelle version de la console.
  - Dans le **Flow Navigator**, cliquer directement sur **Generate Bitsream**
    - Cela relancera également les étapes **Run Synthesis** et **Run Implementation**
- Ouvrir à nouveau le **Hardware Manager** et reprogrammer la carte FPGA comme précédemment.
  - En connectant l'encodeur rotatif au connecteur Pmod B de la carte **Nexys**, vérifiez à présent que vous pouvez bien déplacer la raquette du jeu Casse-Briques à l'aide de l'encodeur rotatif.



# TACHE 3 – 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**.

## 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 **SU-EE100**
- 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

### 3) Game Manager

- Sélection du jeu actif (Casse Briques / Pong)
- ✓ Ce bloc est déjà implémenté.

### 2) Master Slave Manager

- Gestion du mode de la **SU-EE100** (Console / Manette)
- ✓ Ce bloc est déjà implémenté

### 4) Mode

- Gestion de l'état de la partie (En cours, gagnée, perdue...)
- Gestion du mode Pause
- ✗ Module restant à implémenter

En l'état, les fonctionnalités du module **Mode** sont remplacées par les trois 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**.

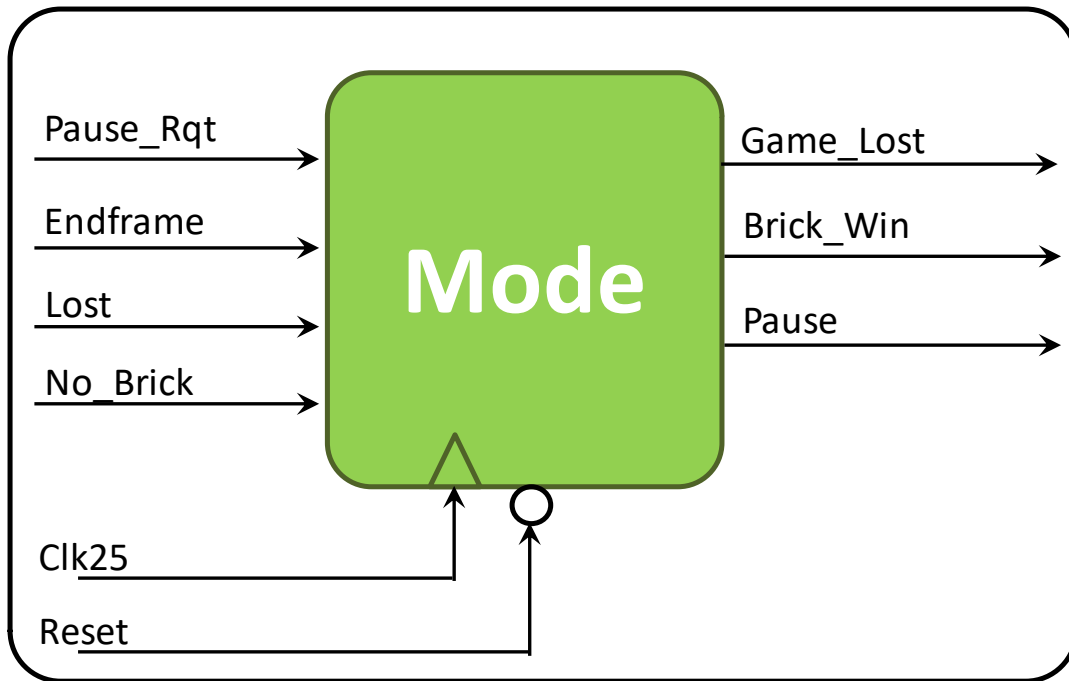
## Affectation du bouton central pour la gestion de la pause

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 **SU-EE100**

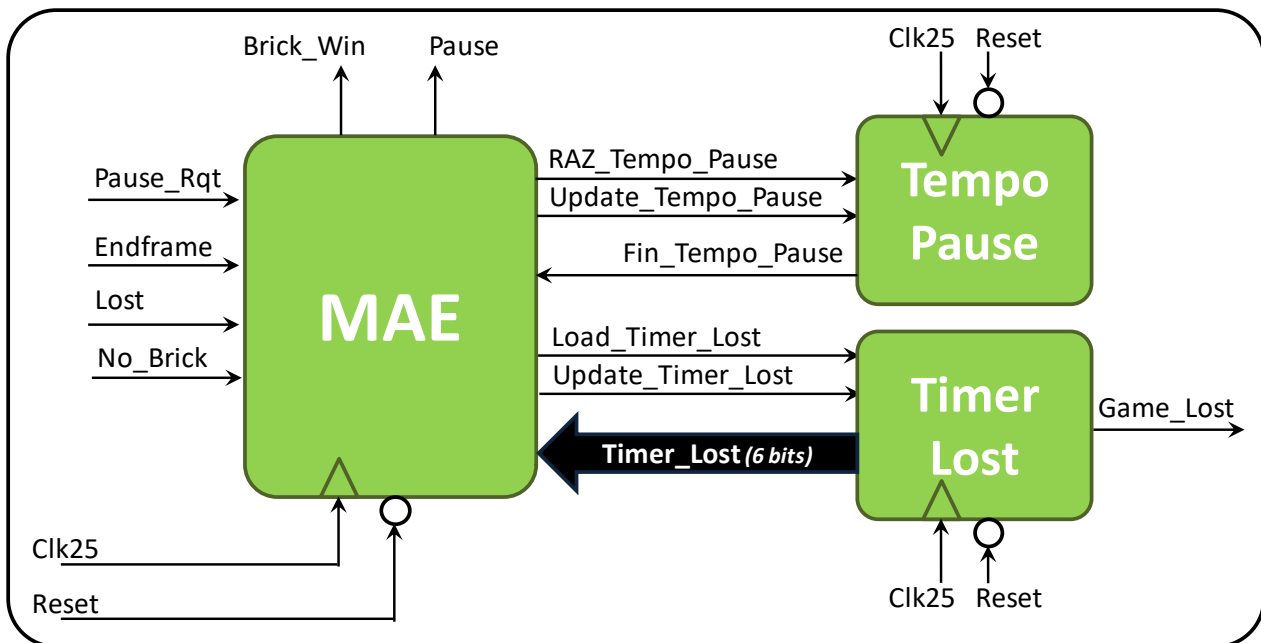
### 2) Timer Lost

- 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

- Faire des schémas blocs des deux compteurs ainsi que le graphe d'états de la MAE, conformément aux descriptifs donnés ci-dessus.
  - Faire valider le graphe et les schémas blocs par l'intervenant de TP.
- Créer un fichier **mode.vhd** que vous allez ajouter aux sources du projet et y décrire le graphe d'états et les deux compteurs.
  - Simuler le fichier à l'aide d'un testbench en suivant les mêmes modalités que pour les tâches précédentes.
- Lorsque tout fonctionne bien, instanciez l'architecture décrite dans **mode.vhd** dans le fichier **game.vhd**, à l'endroit indiqué dans le code.
- Il faut à présent implémenter sur la carte la nouvelle version de la console.
  - Dans le **Flow Navigator**, cliquer directement sur **Generate Bitsream**
  - Ouvrir à nouveau le **Hardware Manager** et reprogrammer la carte FPGA comme 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 devient rouge 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

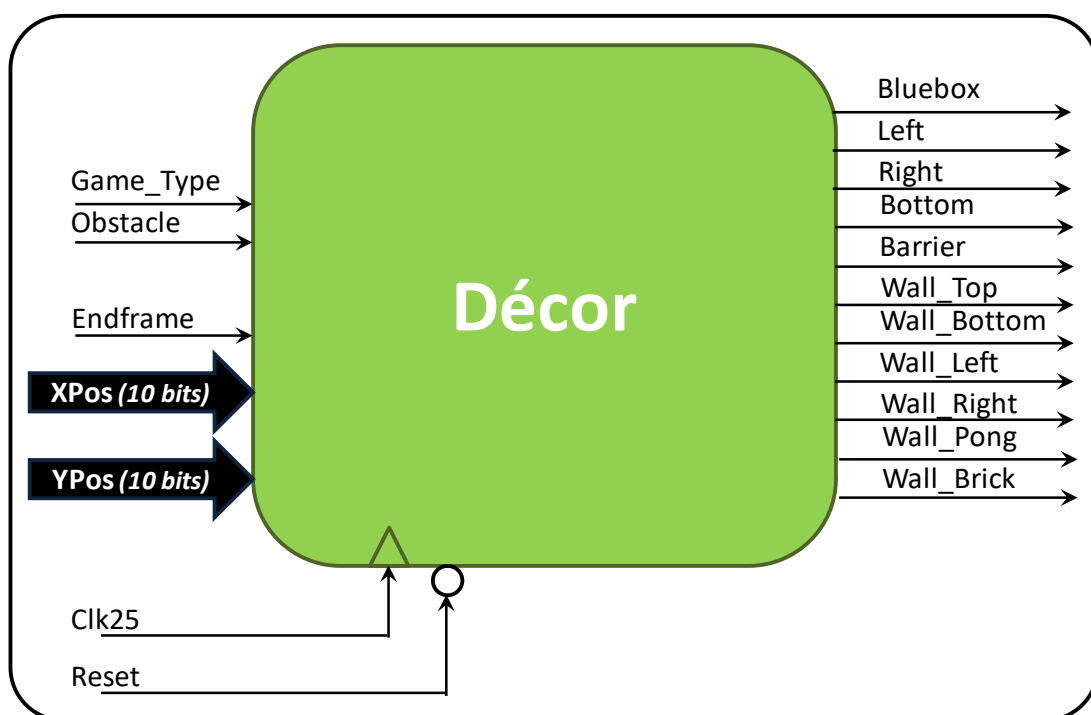
Nous allons à présent ajouter pour le jeu Casse Briques un obstacle mobile horizontal au milieu de l'écran. En cas de rebond avec la balle, cet obstacle renverra la balle dans la direction opposée.

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 hauteur et de 100 pixels de largeur.
- Il se déplace horizontalement de deux pixels à chaque image.
- Lorsqu'il atteint l'un des murs de gauche ou de droite, son sens de déplacement est inversé.
- La commande **Barrier** est activée si :
  - o Le jeu actif est Casse Briques
  - o La commande **Obstacle** est activée
  - o 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.
  - o **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
  - o **YBarrier** : signal sur 9 bits. Indique l'ordonnée de la 1<sup>ère</sup> ligne de l'obstacle.
  - o **Direction** : indique le sens de déplacement de l'obstacle.

## Description VHDL de l'obstacle - Implémentation

- Dans **Vivado**, ouvrir le fichier **decor.vhd**. A l'endroit indiqué en commentaires, et 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

Choisissez maintenant une des améliorations suivantes que vous allez implémenter sur la console.

- **Réglage des seuils de l'accéléromètre pour améliorer la jouabilité.**
  - On pourra notamment accélérer le déplacement de la raquette si on augmente l'inclinaison de la carte.
- **Ajout d'un compteur de points pour le jeu Casse-briques.**
  - Ajouter un compteur de points qui s'incrémente à chaque nouvelle brique cassée.
  - Ce score sera affiché sur les afficheurs 7 Segments.
- **Ajout d'un timer pour le jeu Casse-briques.**
  - Ajouter un compteur de temps (par exemple 2 minutes) qui va se décrémenter si l'on joue à Casse-briques et que la pause est désactivée.
  - Lorsque le compteur arrive à échéance, si le joueur n'a pas encore cassé toutes les briques, alors la partie est perdue.
  - On pourra éventuellement afficher sur les afficheurs le temps restant pour finir la partie.
- **Couleur d'affichage des raquettes**
  - La sortie VGA de la carte **Nexys 4** comporte en fait 4 bits par couleur (rouge, vert, bleu). Il est donc possible de modifier la teinte des objets du jeu, comme par exemple la raquette.
  - A l'aide d'un compteur qui s'incrémente toutes les N images, modifier la teinte de la raquette pour qu'elle passe du jaune au blanc puis au vert (par exemple).

Vous pourrez pour cela vous appuyer sur la documentation technique de la console **SU-EE100**, ainsi que de la carte **Nexys 4**, disponibles sur le site Moodle de l'UE, section TP

