Description du Système Proposé en VHDL

Le système est un **compteur configuré** doté des fonctionnalités suivantes (conformément au cahier des charges) :

1. **Incrémentation contrôlée :**
   Le compteur s'incrémente à chaque front montant de l'horloge (`clock`) sauf si une autre commande (comme `hold` ou `load_count`) intervient.

2. **Maintien de la valeur (`hold`) :**
   Si le signal `hold` est activé (`hold = 1`), la valeur en sortie (`sortie`) reste figée jusqu'à ce que le signal soit désactivé (`hold = 0`).

3. **Chargement d'une valeur spécifique (`load_count`) :**
   Si `load_count = 1`, la valeur présente sur l'entrée `chargement` est transférée directement à la sortie.

4. **Limitation configurable :**
   - Deux limites (Limite1 et Limite2) sont définies.
   - Le signal `sel_limit` sélectionne laquelle des deux limites sera utilisée.
   - Si `sel_limit = 1`, la limite active est `Limite1`. Si `sel_limit = 0`, c'est `Limite2`.

5. **Dépassement (`overflow`) :**
   Lorsque le compteur atteint la limite active, il déclenche un signal `overflow` et retourne à 0.

Code VHDL du Système

Voici une description du compteur en VHDL :

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity compteur is
    Port (
        clock       : in  std_logic;      -- Signal d'horloge
```

```
        hold        : in  std_logic;      -- Maintien de la valeur
        load_count  : in  std_logic;      -- Chargement direct
        chargement  : in  std_logic_vector(7 downto 0); -- Valeur à charger
        load_limit  : in  std_logic;      -- Chargement des limites
        sel_limit   : in  std_logic;      -- Sélection de la limite active
        limite1     : in  std_logic_vector(7 downto 0); -- Limite 1
        limite2     : in  std_logic_vector(7 downto 0); -- Limite 2
        sortie      : out std_logic_vector(7 downto 0); -- Valeur en sortie
        overflow    : out std_logic       -- Signal de dépassement
    );
end compteur;

architecture Behavioral of compteur is
    signal compteur_val : std_logic_vector(7 downto 0) := "00000000"; -- Valeur interne du
compteur
    signal limite_active : std_logic_vector(7 downto 0); -- Limite actuellement utilisée
begin
    process(clock)
    begin
        if rising_edge(clock) then
            -- Gestion du signal hold
            if hold = '1' then
                compteur_val <= compteur_val;
            -- Gestion du chargement direct
            elsif load_count = '1' then
                compteur_val <= chargement;
            -- Gestion normale du comptage
            else
                if compteur_val = limite_active then
                    compteur_val <= "00000000";
                    overflow <= '1';
                else
                    compteur_val <= compteur_val + 1;
                    overflow <= '0';
                end if;
            end if;
        end if;
    end process;

    -- Chargement des limites
    process(load_limit, sel_limit, limite1, limite2)
    begin
        if load_limit = '1' then
            if sel_limit = '1' then
```

```
            limite_active <= limite1;
        else
            limite_active <= limite2;
        end if;
    end if;
end process;

-- Assignation des sorties
sortie <= compteur_val;

end Behavioral;
```

Testbench pour Valider le Comportement

Le testbench est conçu pour tester chaque fonctionnalité et vérifier qu'elle respecte le cahier des charges.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity tb_compteur is
end tb_compteur;

architecture Behavioral of tb_compteur is
    -- Composant à tester
    component compteur
        Port (
            clock       : in  std_logic;
            hold        : in  std_logic;
            load_count  : in  std_logic;
            chargement  : in  std_logic_vector(7 downto 0);
            load_limit  : in  std_logic;
            sel_limit   : in  std_logic;
            limite1     : in  std_logic_vector(7 downto 0);
            limite2     : in  std_logic_vector(7 downto 0);
            sortie      : out std_logic_vector(7 downto 0);
            overflow    : out std_logic
        );
    end component;
```

```
    -- Signaux internes
    signal clock       : std_logic := '0';
    signal hold        : std_logic := '0';
    signal load_count  : std_logic := '0';
    signal chargement  : std_logic_vector(7 downto 0) := "00000000";
    signal load_limit  : std_logic := '0';
    signal sel_limit   : std_logic := '0';
    signal limite1     : std_logic_vector(7 downto 0) := "00000101"; -- 5
    signal limite2     : std_logic_vector(7 downto 0) := "00001010"; -- 10
    signal sortie      : std_logic_vector(7 downto 0);
    signal overflow    : std_logic;

begin
    -- Instanciation du compteur
    uut: compteur
        Port map (
            clock       => clock,
            hold        => hold,
            load_count  => load_count,
            chargement  => chargement,
            load_limit  => load_limit,
            sel_limit   => sel_limit,
            limite1     => limite1,
            limite2     => limite2,
            sortie      => sortie,
            overflow    => overflow
        );

    -- Génération de l'horloge
    clock_process : process
    begin
        while true loop
            clock <= '0';
            wait for 10 ns;
            clock <= '1';
            wait for 10 ns;
        end loop;
    end process;

    -- Stimulus pour les tests
    stimulus_process : process
    begin
        -- Test de l'incrémentation normale
        wait for 50 ns;
```

```
        -- Test du maintien de la valeur
        hold <= '1';
        wait for 40 ns;
        hold <= '0';

        -- Test du chargement direct
        load_count <= '1';
        chargement <= "00000110"; -- Charger la valeur 6
        wait for 20 ns;
        load_count <= '0';

        -- Test des limites
        load_limit <= '1';
        sel_limit <= '1'; -- Activer limite1
        wait for 20 ns;
        load_limit <= '0';

        wait;
    end process;

end Behavioral;
```

Validation et Résultats Attendus

1. **Simulation dans un outil comme ModelSim ou Vivado.**
2. **Observation des signaux principaux :**
    - Vérifiez l'incrémentation, le maintien (`hold`), et le chargement direct (`load_count`).
    - Confirmez que la limite active fonctionne correctement.
    - Assurez-vous que le signal `overflow` est activé au dépassement.

—----------

D'accord, voici comment procéder pour utiliser **Xilinx Vivado** pour simuler votre design VHDL et vérifier que le comportement du compteur respecte le cahier des charges.

Étapes pour Simuler le Compteur dans Vivado

1. **Créer un Projet Vivado :**
    - Ouvrez Vivado.
    - Cliquez sur **Create New Project**.
    - Donnez un nom à votre projet (par exemple `CompteurProject`).
    - Sélectionnez le répertoire pour stocker le projet et cliquez sur **Next**.
2. **Sélectionner la Carte/Partie (Part) :**
    - Choisissez votre FPGA ou carte cible. Si vous ne travaillez pas avec un FPGA spécifique pour cette simulation, sélectionnez une partie générique (par exemple, `xc7z010clg400-1`).
    - Cliquez sur **Next** et terminez la création du projet.
3. **Ajouter des Fichiers Sources :**
    - Cliquez sur **Add Sources**.
    - Ajoutez votre fichier **compteur.vhdl** (le code du compteur) en tant que **HDL Source**.
    - Ajoutez également votre **testbench.vhdl** en tant que **Simulation Source**.
    - Cliquez sur **Finish**.
4. **Ajouter des Fichiers de Contraintes (si nécessaire) :**
    - Si vous n’avez pas de contraintes (comme pour un FPGA), vous pouvez ignorer cette étape.
5. **Créer le Diagramme de Simulation :**
    - Allez dans **Flow Navigator** à gauche et cliquez sur **Run Simulation** → **Run Behavioral Simulation**.
    - Vivado générera un diagramme de simulation avec les signaux que vous avez définis dans votre testbench.
6. **Configurer la Simulation :**
    - Dans la fenêtre de simulation qui apparaît, vous verrez une ligne de temps avec les signaux du design (comme `clock`, `hold`, `load_count`, etc.).
    - Si les signaux ne sont pas visibles, vous pouvez les ajouter dans la vue de simulation en cliquant avec le bouton droit sur la ligne de simulation et en choisissant **Add to Wave**.

7. **Exécuter la Simulation :**

    - Cliquez sur **Run** (ou **Run - All**) pour exécuter la simulation.
    - Vous pourrez voir l'évolution des signaux dans le temps.
    - **Vérifiez les résultats des tests** : Vous devez voir si le signal `overflow` est correctement activé quand la limite est atteinte, si les valeurs sont maintenues quand `hold = 1`, et si le compteur se charge correctement avec la valeur de `chargement`.

Explication des Résultats Attendus :

1. **Test d'incrémentation :**

    - À chaque front montant de l'horloge, le compteur doit s'incrémenter de 1, sauf si `hold = 1` ou `load_count = 1`.
    - Exemple : Si le compteur commence à `00000000`, il doit évoluer en `00000001`, `00000010`, etc.
2. **Test du signal `hold` :**

    - Si `hold = 1`, le compteur doit garder la même valeur, même pendant plusieurs cycles d'horloge.
    - Exemple : Si `hold = 1` à `00000011`, le compteur doit rester à `00000011` pendant 3 cycles d'horloge.
3. **Test du signal `load_count` :**

    - Lorsque `load_count = 1`, la valeur présente sur `chargement` doit être chargée directement dans le compteur.
    - Exemple : Si `load_count = 1` et `chargement = "00000110"`, la sortie du compteur doit immédiatement passer à `00000110`.
4. **Test des limites (`limite1`, `limite2` et `overflow`) :**

    - Le signal `overflow` doit s'activer lorsque le compteur atteint la limite active (`limite1` ou `limite2`).
    - Exemple : Si `limite1 = "00000101"` (5 en décimal) et le compteur atteint cette valeur, `overflow` doit passer à `1`.
5. **Test de la sélection de la limite (`sel_limit`) :**

    - Si `sel_limit = 1`, la limite active doit être `limite1`. Si `sel_limit = 0`, la limite active doit être `limite2`.

Conseils Pratiques :

- Si vous ne voyez pas les signaux ou si les résultats sont incorrects, vérifiez les affectations des signaux dans le testbench et assurez-vous que l'horloge est générée correctement.
- Vous pouvez ajouter des **assertions** ou des **affichages de messages** dans le testbench pour mieux suivre l'exécution de la simulation.

Exemple d'Observations dans Vivado :

Lors de l'exécution de la simulation, vous devriez observer les suivantes :

1. **Le signal `overflow` devrait passer de `0` à `1` lorsque le compteur atteint la limite sélectionnée**.
2. **Le signal `sortie` devrait changer selon les actions (compte, maintien, chargement).**
3. **Les signaux `hold` et `load_count` doivent produire les comportements spécifiés**.

Analyse des Résultats :

Lorsque la simulation est terminée, si tout est correct, vous pourrez observer :

- Que le compteur a bien atteint `limite1` ou `limite2` et que `overflow` a été activé.
- Que le comportement `hold` fonctionne en maintenant la sortie à sa valeur actuelle lorsque `hold = 1`.
- Que le chargement fonctionne lorsque `load_count = 1`.

Si vous avez des questions supplémentaires pendant la simulation ou si vous rencontrez des erreurs spécifiques, n'hésitez pas à demander. Bonne simulation dans Vivado ! 😊