Voici la démarche complète pour concevoir et tester un **additionneur 16 bits** basé sur les concepts des additionneurs 1 bit et 4 bits.

3.1 Organisation

L'additionneur 16 bits sera composé de **4 additionneurs 4 bits**, connectés en série.
Chaque additionneur 4 bits utilise la retenue de sortie (**COUT_4**) du précédent comme entrée de retenue (**CIN_4**) pour le suivant.

3.2 Description en VHDL

Entité de l'additionneur 16 bits

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

entity ADD16B is
    generic (
        TP: time := 10 ns -- Temps de propagation par défaut
    );
    Port (
        A16     : in  STD_LOGIC_VECTOR(15 downto 0);
        B16     : in  STD_LOGIC_VECTOR(15 downto 0);
        CIN_16  : in  STD_LOGIC;
        S16     : out STD_LOGIC_VECTOR(15 downto 0);
        COUT_16 : out STD_LOGIC
    );
end ADD16B;
```

Architecture de l'additionneur 16 bits

```
architecture Behavioral of ADD16B is
    signal COUT: STD_LOGIC_VECTOR(3 downto 0); -- Retenues internes entre les blocs 4 bits
begin
    -- Instanciation des 4 additionneurs 4 bits
    U0: entity work.ADD4B
        generic map (TP => TP)
        port map (A4 => A16(3 downto 0), B4 => B16(3 downto 0), CIN_4 => CIN_16, S4 =>
S16(3 downto 0), COUT_4 => COUT(0));

    U1: entity work.ADD4B
```

```
    generic map (TP => TP)
    port map (A4 => A16(7 downto 4), B4 => B16(7 downto 4), CIN_4 => COUT(0), S4 =>
S16(7 downto 4), COUT_4 => COUT(1));

  U2: entity work.ADD4B
    generic map (TP => TP)
    port map (A4 => A16(11 downto 8), B4 => B16(11 downto 8), CIN_4 => COUT(1), S4 =>
S16(11 downto 8), COUT_4 => COUT(2));

  U3: entity work.ADD4B
    generic map (TP => TP)
    port map (A4 => A16(15 downto 12), B4 => B16(15 downto 12), CIN_4 => COUT(2), S4 =>
S16(15 downto 12), COUT_4 => COUT_16);
end Behavioral;
```

3.3 Test (Testbench)

Le banc de test suit une logique similaire à celle utilisée pour l'additionneur 4 bits.

Testbench VHDL

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

entity TB_ADD16B is
end TB_ADD16B;

architecture Test of TB_ADD16B is
  signal A, B, S: STD_LOGIC_VECTOR(15 downto 0);
  signal CIN, COUT: STD_LOGIC;
begin
  -- Instanciation de l'additionneur 16 bits
  UUT: entity work.ADD16B
    generic map (TP => 10 ns)
    port map (
      A16 => A,
      B16 => B,
      CIN_16 => CIN,
      S16 => S,
      COUT_16 => COUT
    );

  -- Stimuli de test
```

```
    process
    begin
        -- Tester différentes combinaisons
        A <= "0000000000000000"; B <= "0000000000000000"; CIN <= '0'; wait for 100 ns;
        A <= "0000000000001111"; B <= "0000000000001111"; CIN <= '0'; wait for 100 ns;
        A <= "1111000011110000"; B <= "0000111100001111"; CIN <= '1'; wait for 100 ns;
        A <= "1111111111111111"; B <= "0000000000000001"; CIN <= '1'; wait for 100 ns;
        A <= "1010101010101010"; B <= "0101010101010101"; CIN <= '0'; wait for 100 ns;
        wait;
    end process;
end Test;
```

3.4 Simulation et Résultats

1. **Simulation :**
    - Compilez et simulez avec un outil comme **ModelSim** ou **Vivado**.
    - Vérifiez les signaux de sortie `S16` et `COUT_16` pour toutes les combinaisons.
2. **Résultats attendus :**
    - Les résultats de l'addition doivent être corrects pour chaque combinaison.
    - Les effets du temps de propagation (TP) seront visibles : les sorties `S16` et `COUT_16` peuvent avoir des transitions décalées avant stabilisation complète.

Problèmes Potentiels

1. **Propagation des retenues :**
    - Si le TP est grand, les bits de poids forts dans `S16` pourraient avoir des délais visibles.
    - Une mauvaise synchronisation entre les sorties des blocs pourrait entraîner des valeurs intermédiaires incorrectes avant stabilisation.
2. **Optimisation :**
    - Une alternative pourrait être d'utiliser une structure hiérarchique optimisée (comme un *carry-lookahead adder*) pour réduire les délais.

Ce modèle est fonctionnel. Souhaitez-vous des détails sur les optimisations possibles ou des ajustements ? 😊