



TP 00 : Simulation de schéma de portes logiques avec Quartus II

Création d'un schéma d'un additionneur avec les portes logique sous Quartus II

Objectifs pédagogiques

- Compréhension : Construire un additionneur complet 2 bits en cascade de quatre additionneurs complets 1 bit (ripple-carry).
- Méthodologie : Savoir décomposer un circuit complexe en blocs élémentaires réutilisables.
- Pratique Quartus : Créer des fichiers schéma (.bdf), générer des symboles réutilisables, câbler des bus, et simuler.
- Validation : Mettre en place un plan de test et interpréter les chronogrammes de simulation.

Partie théorique

- ♦ Expression logique d'additionneur complet

Il peut être exprimée uniquement avec des portes de base :

$$Y = A \oplus B \oplus C_{in}$$

$$C_{out} = (A \cdot B) + (A \cdot C_{in}) + (B \cdot C_{in})$$

Préparation et outils

- **Logiciel** : Quartus II installé (version classique).
- **Fichiers** : Schémas (.bdf), symboles générés (.bsf) pour réutiliser des blocs.
- **Éditeur de simulation** : Vector Waveform File (.vwf) pour tests sans testbench, ou ModelSim/Quartus NativeLink avec testbench VHDL si souhaité.

Étapes de réalisation dans Quartus II

- ♦ Étape 1 : Créer un nouveau projet
 1. Ouvrir Quartus II
 2. Fichier → Nouveau projet
 3. Nom du projet : FULL_ADD
 4. Choisir la famille de FPGA (ex. CMAX II)
 5. Ne pas ajouter de fichier VHDL
- ♦ Étape 2 : Créer un fichier de schéma (Schematic)
 - Fichier → Nouveau → Block Diagram/Schematic File
 - Enregistrer sous Full_Adder.bdf
- ♦ Étape 3 : Placer les composants pour un additionneur de 1 bit

- Menu : **Tool** → **Symbol Tool** ou clic sur l'icône de porte logique
- Rechercher et insérer :
- 2 portes XOR : xor2
- 3 portes AND : and2
- 1 portes OR : or3
- Entrées : 3 input
- Sortie : 2 output
- ♦ Étape 4 : Connecter les composants
 - **Câblage** : Relier soigneusement les portes aux ports (respect des équations).
 - **Nommer les nœuds** : Assigner des noms clairs aux fils internes si nécessaire.
- ♦ Étape 5 : Compilation
 - Fichier → Enregistrer
 - **Vérifier** : Processing → Start Compilation (optionnel pour vérifier syntaxe schéma).
 - Vérifier qu'il n'y a pas d'erreurs
 - **Générer symbole**: File → Create/Update → Create Symbol Files for Current File → Produit "full_adder_1b.bsf".
- ♦ Étape 6 : Simulation
 1. **Créer un fichier de simulation**
 - Fichier → Nouveau → Vector waveform File
 - Enregistrer sous full_test.vwf
 2. **Ajouter les entrées/sorties**
 - Menu: Edit → Insert → Node or Bus → Node find → filter : all list
 - Sélectionner les signaux A, B, Cin, Cout, S depuis le schéma
 3. **Définir les valeurs de test**
 - Pour chaque combinaison :
 - A=0, B=0, Cin = 0
 - A=0, B=1, Cin = 1
 - A=1, B=0, Cin = 0
 - A=1, B=1, Cin = 1
 - Utiliser l'outil "Value" pour définir les niveaux logiques
 4. **Définir le timing**
 - Menu : Edit → End Time → 100 ns
 - Chaque état dure 10 ns
 5. **Lancer la simulation**
 - Menu : Simulation → Start Simulation Fonctionnelle.
 - Observer les sorties et vérifier qu'elle correspond à la table de vérité

Conception d'un additionneur complet 2 bits avec retenue d'entrée sous Quartus II

Objectifs pédagogiques

- Construire un additionneur 2 bits en reliant 2 additionneurs 1 bit en cascade.
- Réaliser le schéma dans Quartus II.
- Vérifier le fonctionnement par simulation.

Partie théorique

- ♦ Additionneur 2 bits en cascade (ripple-carry):
 - **Principe** : Quatre full adders 1 bits reliés en série. Le Cout de poids faible devient le Cin (C1) du full adder suivant ainsi de suite.
 - Entrées: A(1 downto 0), B(1 downto 0), Cin
 - Sorties : S(1 downto 0), Cout
 - Fonctionnement :
 - Le bit 0 additionne A0, B0, Cin → produit S0 et Cout1.
 - Le bit 1 additionne A1, B1, Cout1 → produit S1 et Cout.
 - Cin du bit suivant.
 - **Chaînage** : Cin → FA0 → Cout1 → FA1 → Cout.

Étapes pratiques

- ♦ Étape 1 : Création du projet
 1. Ouvrir Quartus II.
 2. Créer un nouveau projet nommé adder2.
 3. Choisir la technologie cible (FPGA/CPLD).
- ♦ Étape 2 : Création du schéma de l'additionneur 2 bits
 1. Créer un nouveau fichier Schematic nommé adder2.
 2. Ports vectoriels :
Entrées : A[1..0], B[1..0], Cin.
Sorties : S[1..0], Cout.
 3. Bus et attachements :
Créer bus : Utiliser l'outil "Bus" pour tracer des bus pour A, B, S.
Attacher les bits : Insérer "Bus Pins" pour A[0], A[1] (idem pour B et S).
 1. Insérer 2 instances du schéma full_adder. (FA0, FA1).
 4. Relier :
 - Cin → premier additionneur (FA1).
 - Cout1 de chaque additionneur (FA1) → Cin du suivant (FA2).
 - Les sorties: S0..S1 → vecteur S(1 downto 0).
 - Le dernier Cout → sortie finale Cout.
- ♦ Étape 4 : Simulation

1. **Fichier. vwf:** File → New → vector waveform File → ajouter signaux A[1..0], B[1..0], Cin, S[1..0], Cout.
 2. **Stimuli :**
 - **Cas 1:** A=01, B=11, Cin=0 → Attendu : S=00, Cout=1.
 - **Cas 2:** A=11, B=10, Cin=0 → Attendu : S=10, Cout=1.
 3. **Lancer :** Simulation → Run Functional Simulation.
 4. **Observer :** Vérifier transitions et stabilité des retenues C1, et résultat S/Cout.
- ◆ Étape 5 : PIN Planner
- Associer les signaux logiques du projet (ex A[1..0], B[1..0], Cin, Cout) aux broches physiques du FPGA/CPLD afin de relier correctement le design au matériel (boutons, LEDs, switch...).
- ◆ Étape 6 : Programmer
1. Dans la fenêtre Programmer, cocher **Program/Configure**.
 2. Cliquer sur **Start**.
 3. Attendre le message **Success**.



Schéma attendu

- 2 blocs Full Adder en cascade.
- Entrées : A(1..0), B(1..0), Cin.
- Sorties : S(1..0), Cout.
- Chaque bloc reçoit un bit de A, un bit de B, et la retenue du bloc précédent.

ANNEX

#-----LED-----#

set_location_assignment	PIN_67	-to	led[1]
set_location_assignment	PIN_66	-to	led[2]
set_location_assignment	PIN_61	-to	led[3]
set_location_assignment	PIN_58	-to	led[4]
set_location_assignment	PIN_57	-to	led[5]
set_location_assignment	PIN_56	-to	led[6]
set_location_assignment	PIN_55	-to	led[7]
set_location_assignment	PIN_54	-to	led[8]

#-----Bouton-----#

set_location_assignment	PIN_49	-to	key[1]
set_location_assignment	PIN_48	-to	key[2]
set_location_assignment	PIN_47	-to	key[3]
set_location_assignment	PIN_44	-to	key[4]
set_location_assignment	PIN_43	-to	key[5]
set_location_assignment	PIN_42	-to	key[6]

#-----Switch-----#

set_location_assignment	PIN_53	-to	ckey[1]
set_location_assignment	PIN_52	-to	ckey[2]
set_location_assignment	PIN_51	-to	ckey[3]
set_location_assignment	PIN_50	-to	ckey[4]

