Labo 1 : Checksum



École : HEIG-VD

Domaine : Système logique

Auteurs : Brodo Christopher  
Graf Calvin

Date de rendu : 30.12.2022

Table des matières

[Etape 1-a : 2](#_Toc118055381)

[Etape 1-b : 3](#_Toc118055382)

[Etape 1-c : 3](#_Toc118055383)

[Etape 1-d : 4](#_Toc118055384)

[Etape 1-e : 4](#_Toc118055385)

[Etape 2-a : 5](#_Toc118055386)

[Etape de la carte MAX-V 6](#_Toc118055387)

[Etape 3-a : 7](#_Toc118055388)

# Etape 1-a :

Pour créer l’additionneur 4 bits, nous avons simplement repris le tutoriel de Logisim réalisé au début de l’année et récupéré les additionneurs 1 bit et 4 bits sans y apporter de modification puisque ceux-ci fonctionnaient.

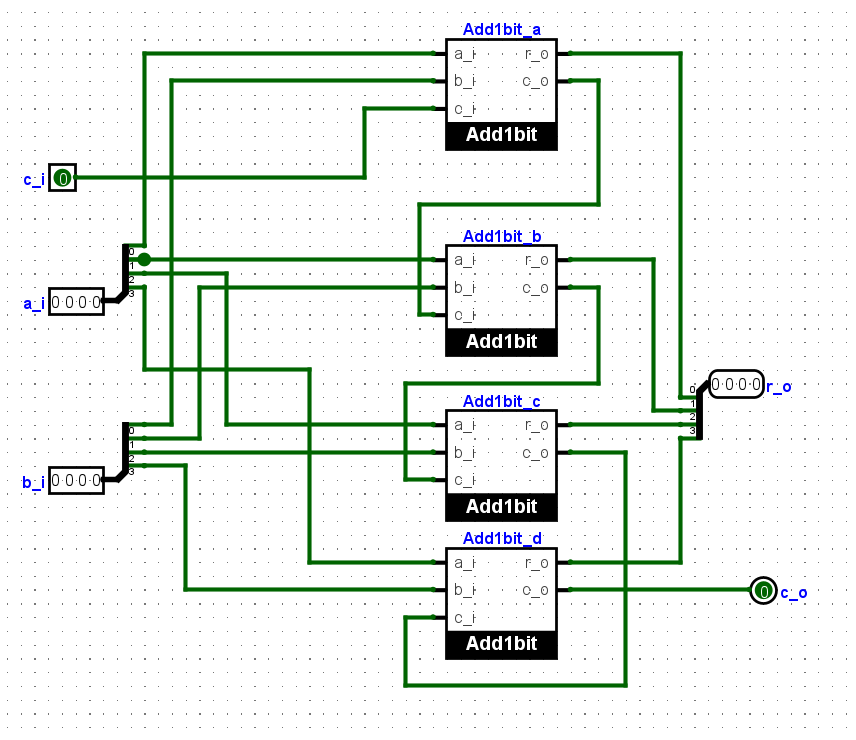


Figure – Add 1 bit

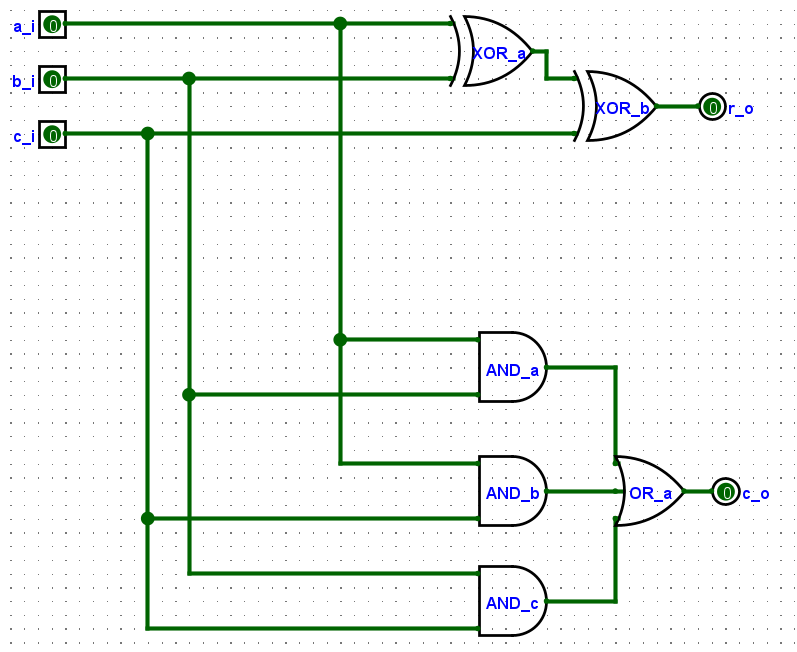


Figure - Add 4 bits

# Etape 1-b :

Pour créer le soustracteur, nous avons repris l’additionneur en utilisant le bit d’overflow pour la retenue de la soustraction. En soi, nous ne pouvons pas soustraire, ce que nous faisons donc c’est d’additionner un nombre que l’on aura préalablement inversé pour l’obtenir en négatif. Exemple : au lieu de faire 7 – 3, nous faisons 7 + (-3).

Pour cela, nous avons mis une porte XNOR qui lie le carry out au c\_in de l’additionneur 4 bits.

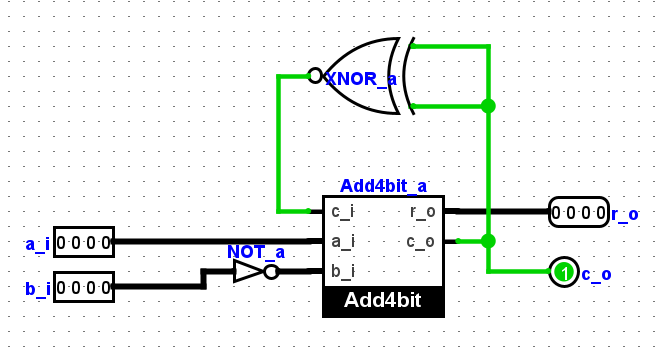


Figure - Sub 4 bits

# Etape 1-c :

Pour l’additionneur / soustracteur nous avons placé une porte XOR qui utilise l’entrée B et le bit de sélection. On le transforme en 4 bits à l’aide d’un splitter ce qui nous permet de réaliser une éventuelle soustraction si c’est le choix de l’utilisateur, sans créer de problème pour l’addition.

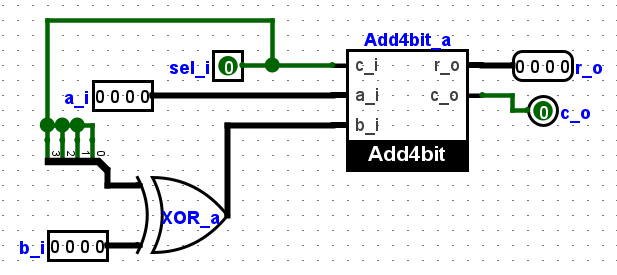


Figure – Add / Sub 4 bits

# Etape 1-d :

Pour l’additionneur 8 bits, nous avons réutilisé la même méthode que pour l’additionneur 4 bits, à l’exception que nous utilisons cette fois directement des additionneurs 4 bits au lieu de 1 bit mais le principe reste le même.

Notre c\_in est relié uniquement aux premiers additionneurs mais nos deux entrées a\_in et b\_in sont reliés chacun à un des deux additionneurs, puis le c\_out du premier se transmet au deuxième ce qui nous donne notre résultat final.

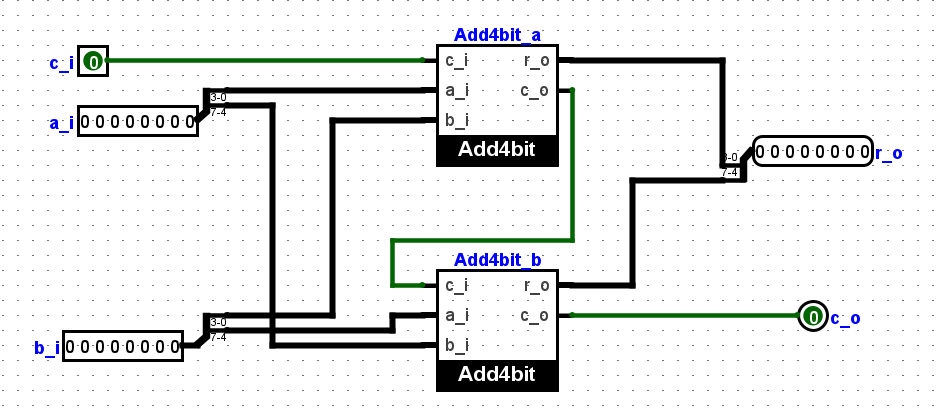


Figure - Add 8 bits

# Etape 1-e :

L’Add / Sub 8 bits a le même principe que les 4 bits, il a simplement fallu ajouter un splitter à 8 bits d’entrée au lieu de 4. C’est le seul changement que nous devions apporter.

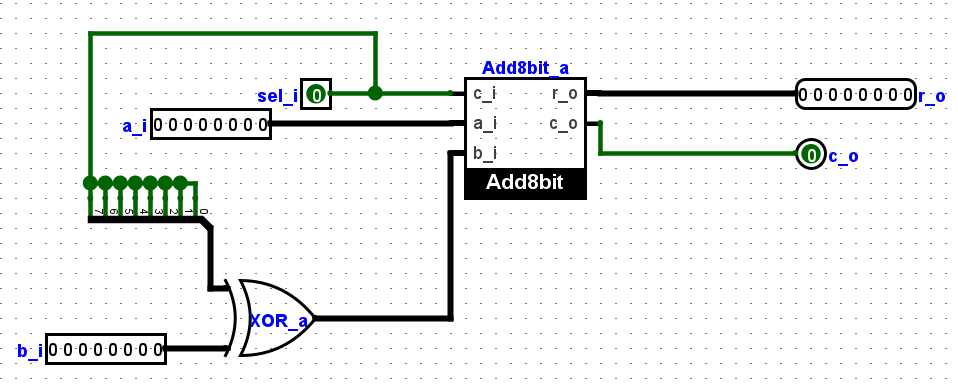


Figure 6 - Add / Sub 8 bits

# Etape 2-a :

Pour le checksum nous avons raisonné de cette manière. Nous avons mis 3 additionneurs / soustracteurs 8 bits qui s’occupent d’additionner les trois nombres. Puis nous avons ajouté une porte logique NOT reliée directement à un bit constant de 0 qui nous permet de trouver le complément à deux du nombre.

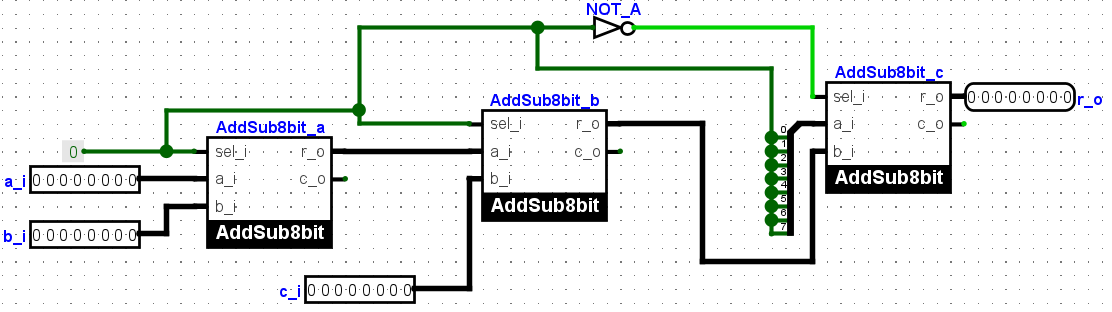


Figure - Checksum

# Etape de la carte MAX-V

Une fois la carte MAX-V installée et connecter à l’ordinateur, lancer Logisim, et aller dans « FPGA MENU » puis « FPGA COMMANDER ».

Penser à réduire le nombre d’entrées à 8, remplacez le reste par des constantes.

Une image contenant texte

Description générée automatiquement

Bien vérifier que le « main circuit » soit le circuit qui soit voulu et vérifier aussi que la « target board » soit la MAX\_V Console et non la MAX\_V 80.

Puis, cliquez sur annotate et download. Une fois ceci fait, placez vos pins d’entrées sur les sticks de la console et le pin de sortie sur les clignoteurs tout en haut.

Une fois tous les pins placés, cliquer sur « Done » pour envoyer les informations sur la carte. Il suffit d’attendre quelques secondes, puis de vérifier sur la carte en activant les leviers que tout fonctionne.

**(RAPPEL)**: Pour calculer un nombre en complément à deux, il faut inverser les bits et rajouter + 1. Exemple ;

22 = 0001 0110 Il faut donc l’inverser -> 1110 1001

Une fois ce nombre obtenu, rajouter simplement + 1 => 1110 1001 + 0000 0001 = 1110 1010

# Etape 3-a :

Nous avons ensuite terminé avec le circuit permettant de contrôler la validité du checksum. Nous recevons donc 3 entrées qui permettent de réaliser un checksum, puis nous allons contrôler que le résultat est bien égal à la quatrième entrée. Pour cela, on utilise une porte XNOR entre le résultat du checksum et l’entrée d\_i, puis on vérifie que tout est bien égal avec une porte AND. Si on obtient 1 comme bit de vérification, alors c’est que tout fonctionne.

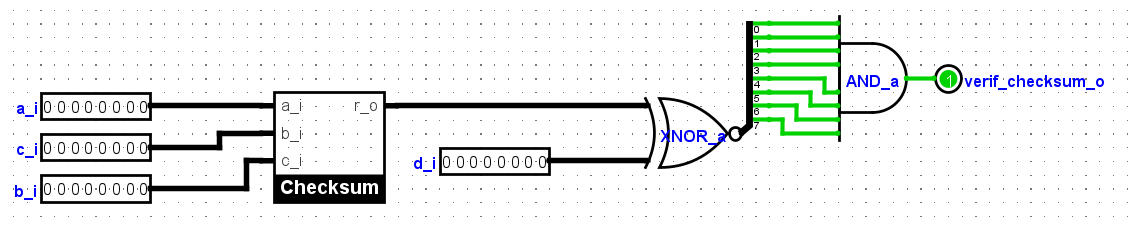


Figure – VerifiyIntegrity