Laboratoire
Codes redondants pour la détection d'erreurs

Sven Rouvinez & Yohann Meyer

2016-10-18

1 Additionneur 8 bits

Sur la base des additionneurs conçus précédemment, concevoir et implémenter un additionneur 8-bits.

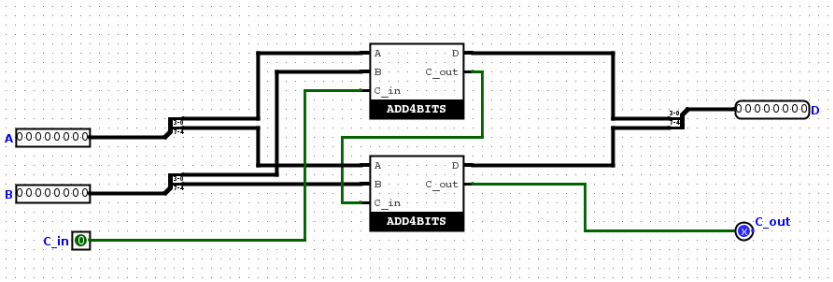


FIGURE *1 – Additionneur 8bits*

L'additionneur 8bits est composé de 2 additionneurs 4bits qui eux sont composés d'aditionneurs 1bit, cette séparation par couche nous a permis de comprendre le fonctionne à la base d'un aditionneur pour ensuite le transposer en 8bits.

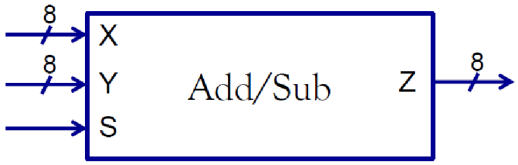
Les entrées sont définies par **A** et **B** sur 8bits et les nombres choisis transitent ensuite par 4 bus de 4 bits avant d'être réunis par un splitter (4-2) après les opérations.

Le fil de l'entrée **A** est connecté sur l'entrée de notre additionneur 4bits et le fil de l'entrée **B** est connecté sur l'entrée **B** de notre additionneur. Le pin **C_out** nous permet de reporter notre retenue dans l'entrée **C_in** du 2ème additionneur 4bits.

Le résultat de notre addition entre **A** et **B** se retrouve dans notre pin de sortie **D** et dans le cas d'un dépassement de mémoire, nous allons le retrouver dans le pin de sortie **C_out** .

2 Additionneur-Soustracteur 8 bits

Concevoir et implémenter un additionneur/soustracteur 8-bit à l'aide d'un bloc additionneur 8-bit et des portes logiques. Note : Ajoutez un bit de sélection d'opération **S** (1 = Soustraction, 0 = Addition). Implémentez et simulez le système sur Logisim. Donnez-en un exemple d'utilisation.



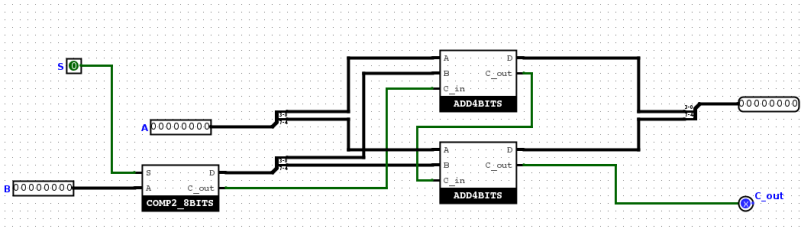


FIGURE *2 – Additioneur- soustracteur 8bits*

L'additionneur/soustracteur utilise le concept du complément à 2, c'est à dire qu'il convertit notre nombre en entrée **B** en remplacant les 0 et 1 et en faisant +1 sur ce nombre. Par exemple : le nombre en binaire 010010 deviendra dans un premier temps (complément à 1) 101101 et ensuite nous rajoutons +1 ce qui donnera 101110.

Ce complément est à 8bits est composé de lui de 2 compléments à 4bits.

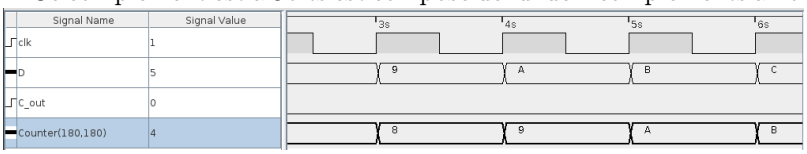


FIGURE *3 – chronogramme ADD_SUB8BITS*

Le chronogramme suivant ce compose d'un élément d'horloge **clk** et d'un autre **sysclk** qui permettent d'incrémenter les valeurs qui rentrent. Il y a un pin de sortie **D** qui affiche le résultat et l'autre affiche les débordement de mémoire. Et l'élément compteur fait office de mémoire afin de stocker toutes les valeurs calculées.

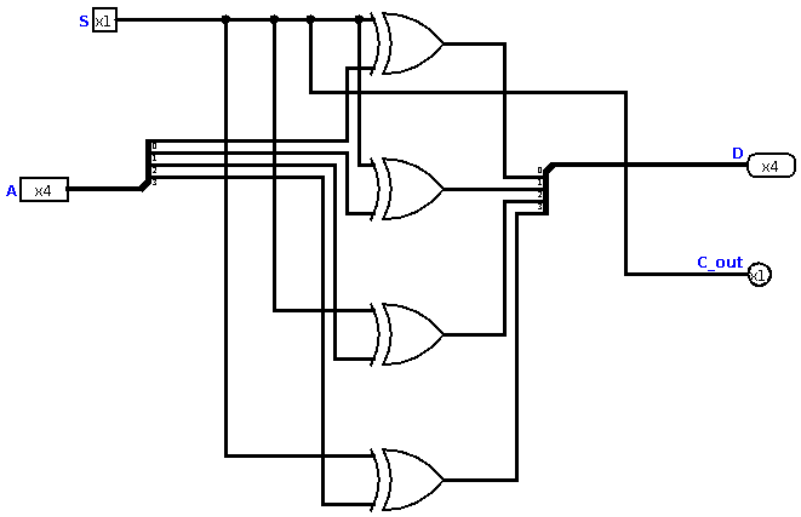


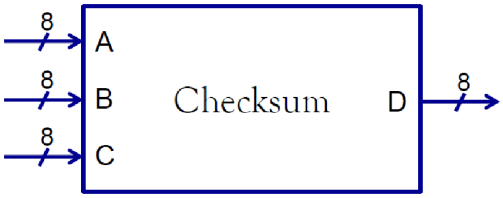
FIGURE 4 – *Complément à 2 - 4bits*

Dans cette image, se trouve la structure de base de nos compléments. Le pin **S** définit s'il s'agit d'une addtion ou d'une soustraction. En effet les portes choisies sont des portes XOR qui on comme propriétés de n'être vrai que lorsque qu'une des entrée est vraie.
Exemple :
Pin S = 1
Pin A = 0100

Pin#	Pin S	Pin A	XOR
0	1	0	1
1	1	1	0
2	1	0	1
3	1	0	1

3 Calcul du Checksum

Concevoir un système combinatoire qui calcule le checksum du type modular sum de trois octets données en entrée. Utiliser des additionneurs-soustracteurs 8-bits et d'autres composants, si nécessaire. Vous devez implémenter et simuler le système à l'aide de Logisim, en donnant quelques exemples d'utilisation.



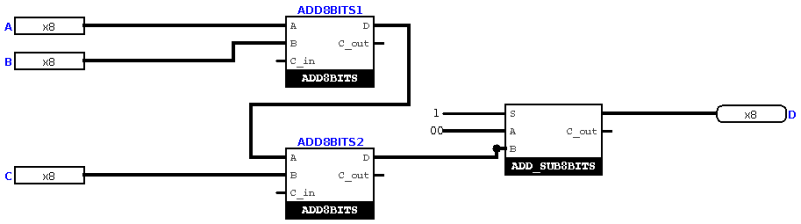


FIGURE 5 – *Checksum 8bits*

Ce circuit se compose de 2 additionneurs 8bits et d'un additionneur/soustracteur 8bits et permet de calculer le checksum de 3 valeurs. Le checksum permet de vérifier l'intégriter des données car doit donner 0 lors de l'addition avec les valeurs qui l'ont produit. Les exemples d'utilisations sont divers, comme par exemple valider un numéro de carte de crédit ou pour assurer l'intégrité d'un fichier téléchargé.
La structure de ce checksum est : 2 additionneurs 8bits et 1 additionneur/soustracteur 8bits

1.

A	00000001
B	00000001
ADD8BITS 1	00000010

2.

ADD8BITS 1	00000010
C	00000001
ADD8BITS 2	00000011

3.

ADD8BITS 2	00000001
ADD8BIT_SUBS 2	00000011
Sortie	11111101

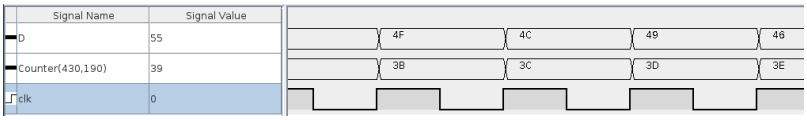


FIGURE 6 – *Chronogramme Checksum*

4 Vérification d'intégrité en réception

Concevoir un circuit combinatoire qui permet de vérifier l'intégrité de quatre octets reçus en entrée (c.-à.d trois octets et leur checksum) et de calculer un bit de sortie valide égal à 1 lorsque aucune altération des données est détectée à la réception. Note : Ce circuit reçoit quatre octets et ne peut pas savoir quel est l'octet de checksum.

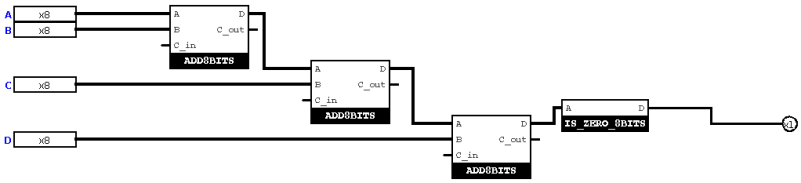


FIGURE 7 – *Verificateur d'intégrité*

Ce circuit permet de contrôler l'intégrité d'une donnée représentée ici avec 3 octets et 1 checksum pour assurer que la transmission c'est bien déroulée. Comme le checksum représente la valeur qui additionnée avec les valeurs d'entrées vaut zéro et que l'addition est **commutative**, il n'est pas nécessaire de connaître l'ordre. L'addition des 4 octects doit être égal à 0 afin de garantir l'intégrité des données, dans le cas contraire , nous nous retrouvons avec des données altérées et donc potentiellement des données fausses.

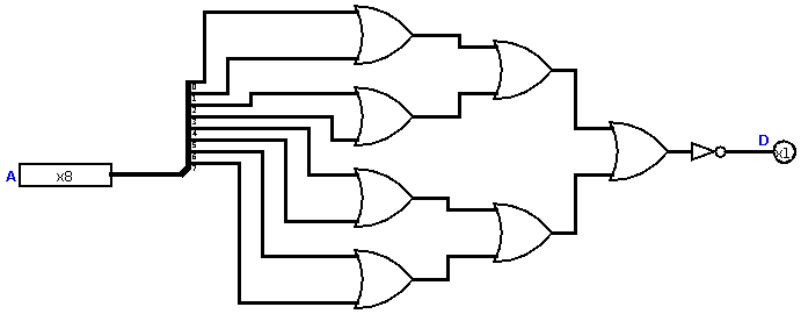


FIGURE 8 – *IsZero*

Le circuit ci-dessus est utilisé pour retourner que la valeur entrée vaut bien zéro.
Les portes utilisées sont des OR qui ont comme propriétés de retourner faux si les 2 valeurs sont égales à 1, et étant donné que nous avons besoin d'avoir vrai si l'entrée vaut 0 il faut une porte NOT qui va inverser le résultat de nos portes et donc nous afficher un si l'entrée **A** vaut 0.

5 Vérification du 1er groupe de données

Supposez que les données 0x45, 0x68 et 0x73 sont envoyés avec l'octet de checksum correspondant, mais que par des aléas de transmission les données reçues sont 0x4D, 0x68, 0x73 et le checksum sans erreur. Calculez le checksum transmis et utilisez le circuit combinatoire développé pour vérifier l'intégrité de trois octets reçus. Que se passe-t-il ? Pourquoi ?

Obtention du checksum. Le calcul se fait avec les données 0x45, 0x68 et 0x73 afin d'obtenir le résultat correspondant à ces données.

1. Addition du pin **A** et **B**
   `01000101 + 01101000 = 10101101`
2. Addition du résultat de l'étape 1 et pin **C**
   `10101101 + 01110011 = 00100000`
3. Addition du résultat de l'étape 2 et pin **D**
   `00100000 -> 11011111 + 00000001 = 11100000`
4. Passage par IS_ZERO pour définir le bit de sortie.

Notre checksum vaut donc 0xE0. Nous allons pouvoir l'utiliser dans le vérificateur d'intégrité.

Vérification de l'intégrité

Pour simuler une altération des données nous allons utiliser les octects 0x4D, 0x68 et 0x73.

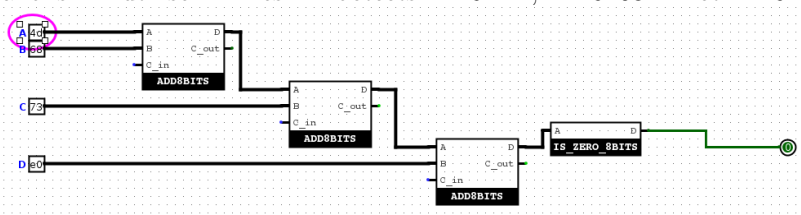


FIGURE 9 – *Vérificateur d'intégrité*

1. Addition du pin **A** et **B**
   `11100101 + 10011110 = 10000011`
2. Addition du résultat de l'étape 1 et pin **C**
   `10110101 + 01110011 = 10101000`
3. Addition du résultat de l'étape 2 et du pin **D**
   `10101000 + 11100000 = 1000100`
4. On compare la valeur obtenue à 0. Si elles ne sont pas égales, il y a eu corruption des données.

6 Vérification du 2ième groupe de données

Supposez que les données 0xA5, 0x9E et 0x6F sont envoyés avec l'octet de checksum correspondant, mais que par des aléas de transmission les données reçues sont 0xE5, 0x9E, 0x2F et le checksum sans erreur. Calculez le checksum transmis et utilisez le circuit combinatoire développé pour vérifier l'intégrité de trois octets reçus. Que se passe-t-il ? Pourquoi ?

Obtention du checksum. Le calcul se fait avec les données 0xA5, 0x9E et 0x6F afin d'obtenir le résultat correspondant à ces données.

1. Addition du pin **A** et **B**
   `10100101 + 10011110 = 01000011`
2. Addition du résultat de l'étape 1 et pin **C**
   `01000011 + 01101111 = 10110010`
3. Complément à 2 du résultat de l'étape 2
   `10110010 -> 01001101 + 00000001 = 01001101`
4. On compare la valeur obtenue à 0. Si elles ne sont pas égales, il y a eu corruption des données.

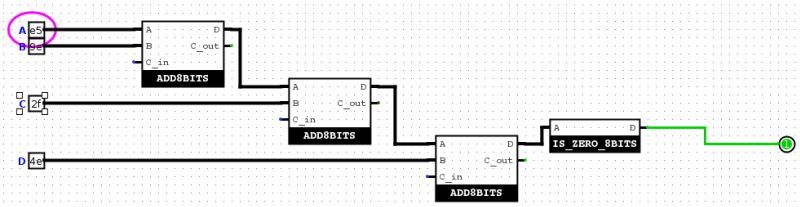


FIGURE *10 – Vérificateur d'intégrité 2*

1. Addition du pin **A** et **B**
   `11100101 + 10011110 = 10000011`
2. Addition du résultat de l'étape 1 et pin **C**
   `10000011 + 00101111 = 10110010`
3. Complément à 2 du résultat de l'étape 2
   `10110010 + 01001110 = 00000000`
4. Contrôle si le résultat vaut 0x00. C'est le cas

Il semblerait que la transmission se soit correctement faite vu que nous avons un bit de vérification égal à 0.
En réalité, nous sommes confrontés à un overflow. Tout les overflow ne vont pas