Additionneur binaire : niveau difficile

Première partie

- Question préliminaire : Poser et effectuer l'addition binaire ci dessous (faire apparaître les retenues):

```
  10011101
+ 00111001
=
```

Le but de cet exercice est de réaliser un circuit combinatoire permettant de réaliser une addition binaire

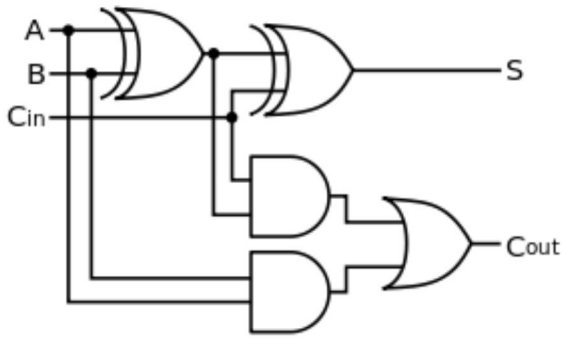
> On appelle demi-additionneur un circuit combinatoire réalisant une addition entre 2 bits (notés A et B) sans retenue entrante. Ainsi dans la question préliminaire, le demi-additionneur est le circuit permettant d'effectuer l'addition des bits de poids faibles (pas de retenue entrante)

1. Compléter ci-dessous la table de vérité du demi-additionneur. On note S le résultat et C la retenue

A	B	S	C
0	0		
0	1		
1	0		
1	1		

2. Donner l'expression booléenne de S et de C en fonction de A et de B puis dessiner le circuit combinatoire correspondant.

> Un **additionneur complet 1 bit** nécessite une entrée supplémentaire : une retenue. Dans la question préliminaire, l'additionneur complet est le ciruit permettant d'effectuer l'addition de n'importe quel bit. On donne ci-dessous, le circuit combinatoire correspondant à un additionneur complet 1 bit. On note Cin la retenue entrante et Cout la retenue sortante.



Source : wikipedia

> Dans la question précédente, nous avons réalisé un demi-additionneur en associant des portes logiques (eux-mêmes constitués de transistors). C'est l'idée de "lego" mentionnée dans le cours. De la même façon, on peut remarquer que l'additionneur complet 1 bit est basé sur l'association de 2 demi-additionneurs et d'une porte logique.

3. Dessiner un schéma de l'additionneur complet 1 bit faisant apparaître l'association de 2 demi-

additionneurs et d'une porte logique.

4.
   - Donner l'expression booléenne de S et de Cout en fonction de A, B et Cin.
   - A l'aide des expressions ci-dessus, montrer que le demi-additionneur est un cas particulier de l'additionneur complet 1 bit pour lequel Cin = 0

> Ainsi, on peut donc considérer que l'additionneur complet 1 bit est *"la brique de base"* permettant de réaliser nimporte quelle addition binaire (comme celle de la question préliminaire)

5. Dessiner le schéma d'un additionneur complet 4 bits montrant l'association de plusieurs "briques de base" additionneur complet 1 bit

> L'additionneur binaire complet ainsi réalisé est appelé *additionneur parallèle à propagation de retenue*. Même s'il donne des résultats justes, ce n'est pas ce circuit qui est utilisé dans les processeurs pour réaliser les additions car il a un inconvénient majeur.

> Même s'ils sont très rapides, les circuits logiques ne donnent pas leur sortie instantannément : il y a un temps de propagation (de l'ordre de la ns) entre le moment où on injecte les entrées et où les résultats apparaissent sur les sorties. On appelle **t** le temps de propagation d'un additionneur complet 1 bit (**t** = temps de calcul d'un additionneur complet 1 bit).

6.
   - Quel est le temps mis pour faire une addition binaire sur 4 bits. Donner la raison en une courte phrase.
   - En déduire la formule mathématique donnant le temps de calcul de l'addition en fonction de la longueur des nombres à additionner (nombre de bits de ces nombres)
   - Tracer l'allure de la courbe donnant le temps de calcul de l'addition en fonction du nombre de bits composant les nombres à additionner

Il existe des additionneurs de technologie plus complexes (à retenue anticipée) mais qui permettent de calculer un résultat en un temps logarithmique de la taille des entrées

7. Ajouter sur le graphique précédent la courbe pour les additionneurs à retenue anticipée.
8. A l'aide du graphique, expliquer en quelques lignes l'avantage des additionneurs à retenue anticipée par rapport aux additionneurs parallèles à propagation de retenue.

Deuxième partie

La suite de l'exercice devra être réalisé en mettant en oeuvre les résultats précédemment trouvés !

1. Ecrire une fonction `str_to_bool` répondant à la documentation ci-dessous. Ainsi `str_to_bool('0')` devra renvoyer `False`. `str_to_bool('1')` devra renvoyer `True`

Entrée [ ]:
```
def str_to_bool(ch):
    """
    Description de la fonction : convertit les chaînes de caractères '1' et '0
    ch (str)
    return (bool)
    """
    # à compléter
```

2. Ecrire une fonction `bool_to_str` répondant à la documentation ci-dessous. Ainsi `bool_to_str(True)` devra renvoyer `'1'`. `bool_to_str(False)` devra renvoyer `'0'`

Entrée [ ]:
```
def bool_to_str(booleen):
    """
```

```
    Description de la fonction : Convertit un booléen en chaîne de caractère '
    booleen (bool)
    return (str)
    """
    # à compléter
```

> On donne la fonction `xor` permettant de calculer le résultat `x xor y` (voir TD `premiere/bloc1/operateurs_booleens/TD_operateurs_booleens.ipynb`)

Entrée [ ]:
```
def xor(x,y):
    return (x and not(y)) or (y and not(y))
```

3. Ecrire une fonction `addComplet_1bit` répondant à la documentation ci-dessous. Pour cela, vous devrez utiliser les fonctions `bool_to_str` et `str_to_bool` ainsi que `xor` précédente

`addComplet_1bit('1','0','1')` devra renvoyer `('0', '1')`

Entrée [ ]:
```
def addComplet_1bit(A,B,Cin):
    """
    Description de la fonction : Additionneur complet 1 bit
    A (str) : 1er bit à additionner
    B (str) : 2eme bit à additionner
    Cin (str) : retenue entrante
    Préconditions sur les paramètres : A, B et Cin ne peuvent être que des cha
                                       égales à '0' ou '1'
    return (tuple) : (S, Cout) avec S et Cout de type (str) ne pouvant être qu
                                       égales à '0' ou '1'
    """
    # à compléter
```