Tricks de FDS (part I)

1. Calculer l’exposant IEEE-754 plus rapidement

Dans tous les cas, peu importe le nombre de bits :
- `0111111111` représente un exposant de 0.
- `1000000000` réprésente un exposant de 1.

Si on veut un exposant de −3, on part de `0111111111` et on enlève 3 → `0111111100`. Si on veut un exposant de 3, on part de `1000000000` (1) et on ajoute **2** → `1000000010`.

2. Calculer les nombres binaires plus rapidement

On s’approche très près de n’importe quel nombre avec la table de 16. On écrit ensuite le nombre en hexadécimal, puis on convertit en binaire.

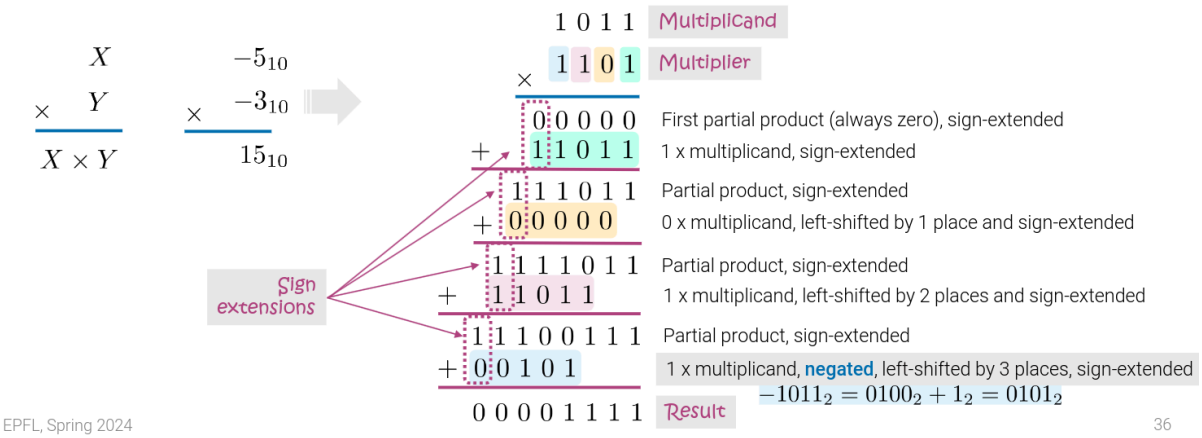
Table de 16

Multiple	16 * Multiple
1	16
2	32
3	48
4	64
5	80
6	96
7	112
8	128
9	144
10	160

3. Convertir de Gray Code ⇔ Binary

Binary vers Gray Code :
→ on copie le MSB, puis on XOR chaque couple de bits qui suit. Gray Code vers Binary :
→ on copie le MSB, puis à chaque 1 dans le gray code (ce qui signifie qu’il y avait un changement dans la chaîne originale) on switch entre 0 et 1.

4. Comprendre la multiplication en 2′s Complement



Pourquoi est-ce qu'on doit negate le dernier multiplicant ? On veut par exemple calculer −17x14.

- Si on fait −17x14 = 101111 x 001110, on fait 2x-17 + 4x-17, etc. jusqu'à avoir 14x-17. Pas besoin de negate !
- Par contre 14x-17 = 001110 x 101111, on fait 14x2 + 14x4 + 14x8 **- 14x32** (c'est pour ça qu'on doit negate le multiplicant).

5. Pourquoi on invert, +1 pour changer de signe en 2′s complement

Imaginons qu'on ait 17. On cherche donc −17. Inverser les bits de 17 revient à trouver les bits qui, sommés à ceux de 17, donne uniquement des 1. Or, uniquement des 1, en 2′s complement, donne −1.

Donc en inversant 17, on a trouvé C tel que 17 + C = −1. Mais nous, on cherche N tel que N = −17 soit 17 + N = 0. Donc on ajoute +1 au complémentaire de C !

6. Comment savoir combien de bits de fractions nécessaires après une opération ?

Pour une addition, ce sera toujours le max(f_x1, f_x2). Parce qu'il ne peut pas y avoir de débordement à droite, toujours à gauche.

Oour une multiplication, ce sera toujours f_x1 + f_x2. Pourquoi ? P. ex. en décimal quand on fait ax10^-3 x bx10^-5 ça donne toujours un résultat le plus faible en 10^-8.

dynamic range -> te dit si le système de nombre peut gérer des très grands nombres et de très petits nombres à la fois.

7. Propriétés en floating point

Accuracy : max diff entre 2 nb dans le système divisée par 2 (pour avoir l'erreur potentielle)

① on maximise l'exposant $2^{e_{max}}$

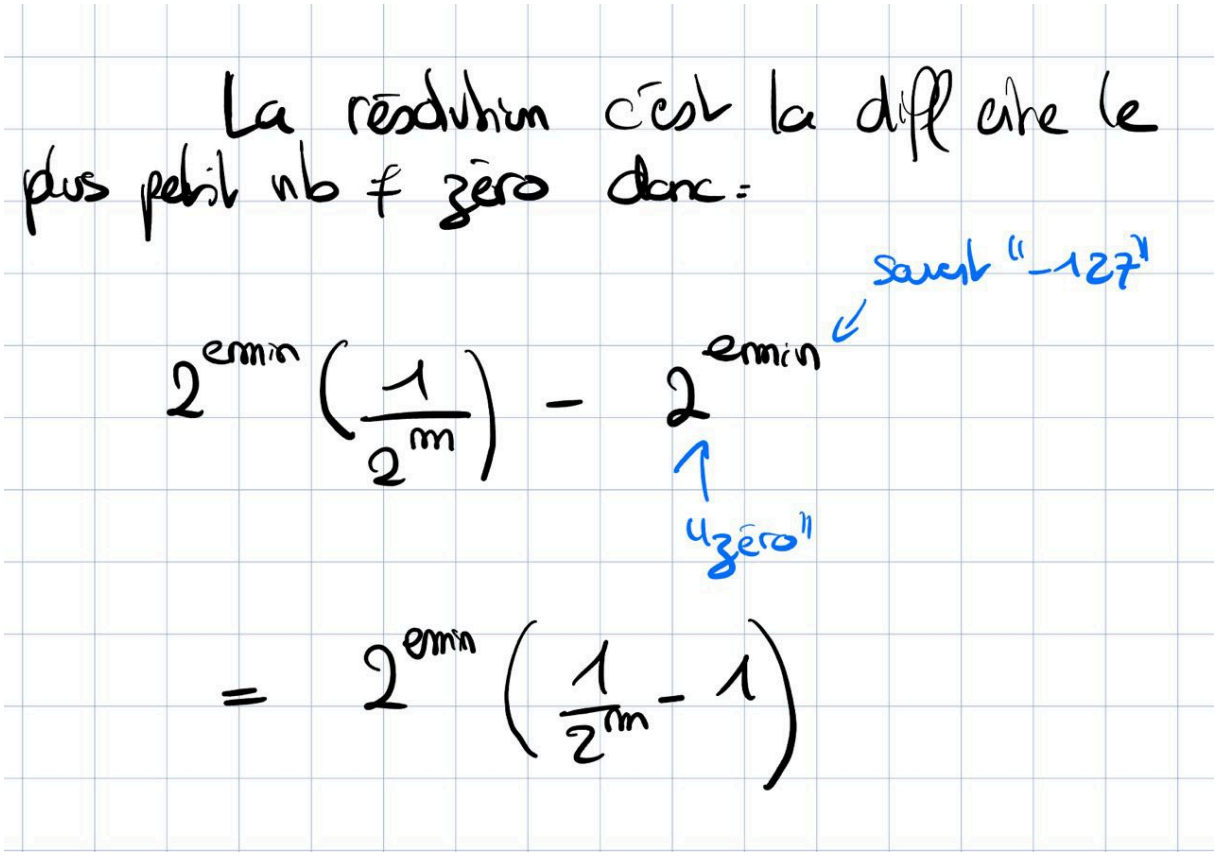
② avec le même exposant, il y aura toujours le même écart entre deux nombres

on prend x_1 et x_1"+1", on fait la différence /2

$$x_1 = 2^{e_{max}} \cdot (1.000\ldots)$$

$$x_1\text{"+1"} = 2^{e_{max}} \cdot (1.00\ldots01)$$
$$= 2^{e_{max}} \cdot \left(1 + \frac{1}{2^m}\right)$$

$$\frac{1}{2}(x_1\text{"+1"} - x_1) = \frac{1}{2} \cdot 2^{e_{max}} \cdot \frac{1}{2^m}$$

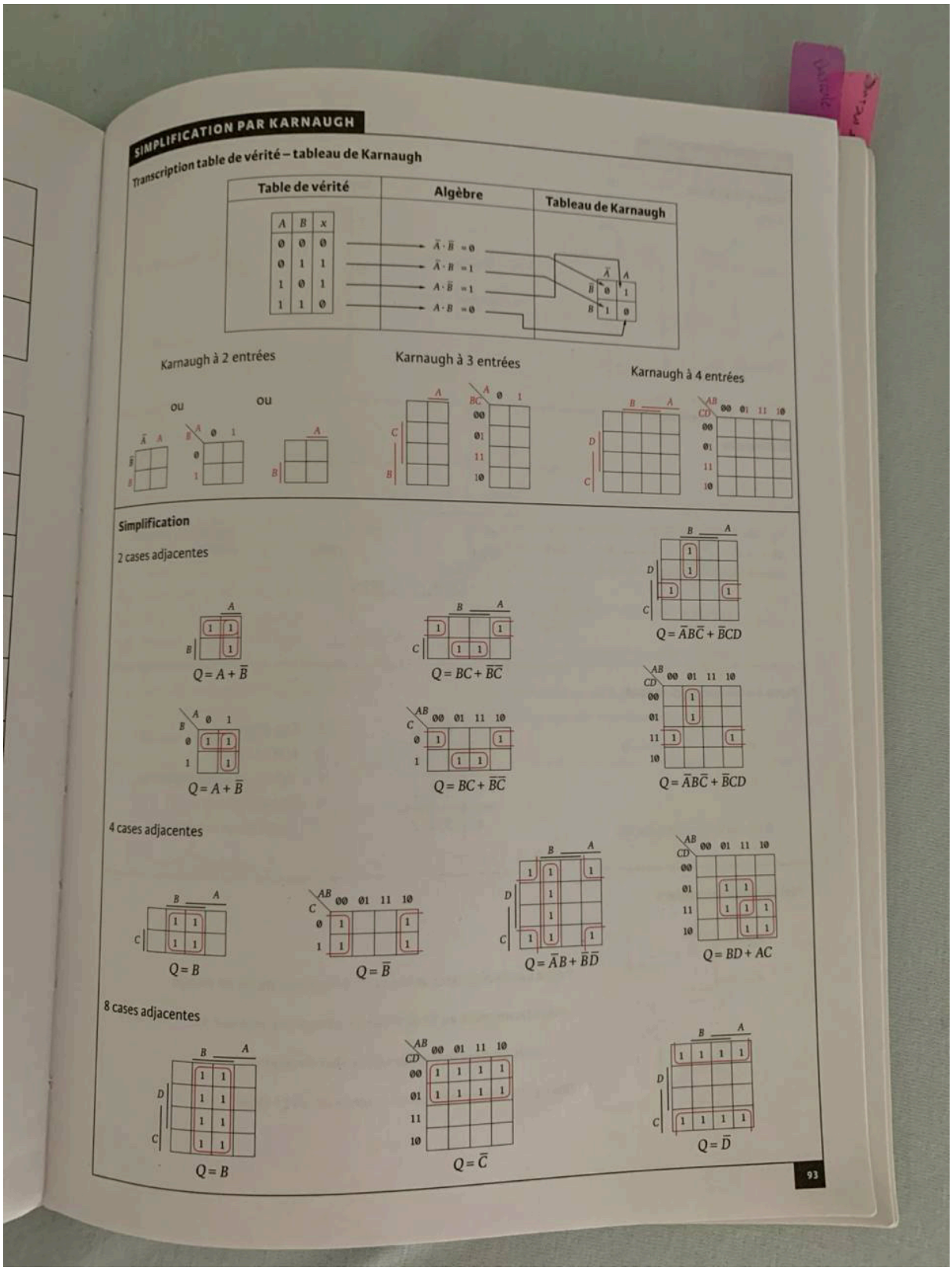


8. Pourquoi est-ce qu'on fait un tie to even ?

Si on veut faire une division après l'arrondi (un shift), on perd le risque de faire une nouvelle erreur. Par exemple 3.5 → on peut soit faire 3 soit faire 4. Si on fait 3/2 = 1 et 4/2 = 2. 3.5/2 = 1.75, 2 est plus proche.

9. Map de Karnaugh

pour faire une karnaugh map → mettre bien les nombres qui ont une unique différence entre eux à côté 10 11 01 00 (10 et 01 sont à côté parce qu'on considère un tableau comme une boule 3D).



10. Pourquoi on écrit la P.O.S en prenant les lignes à zéro et en complémentant ?

En fait on “fuit les zéros”, c-a-d qu’on prend toutes les lignes qui donnent zéro, et pour chaque ligne, pour éviter de tomber dedans, on sait qu’il suffit qu’au moins

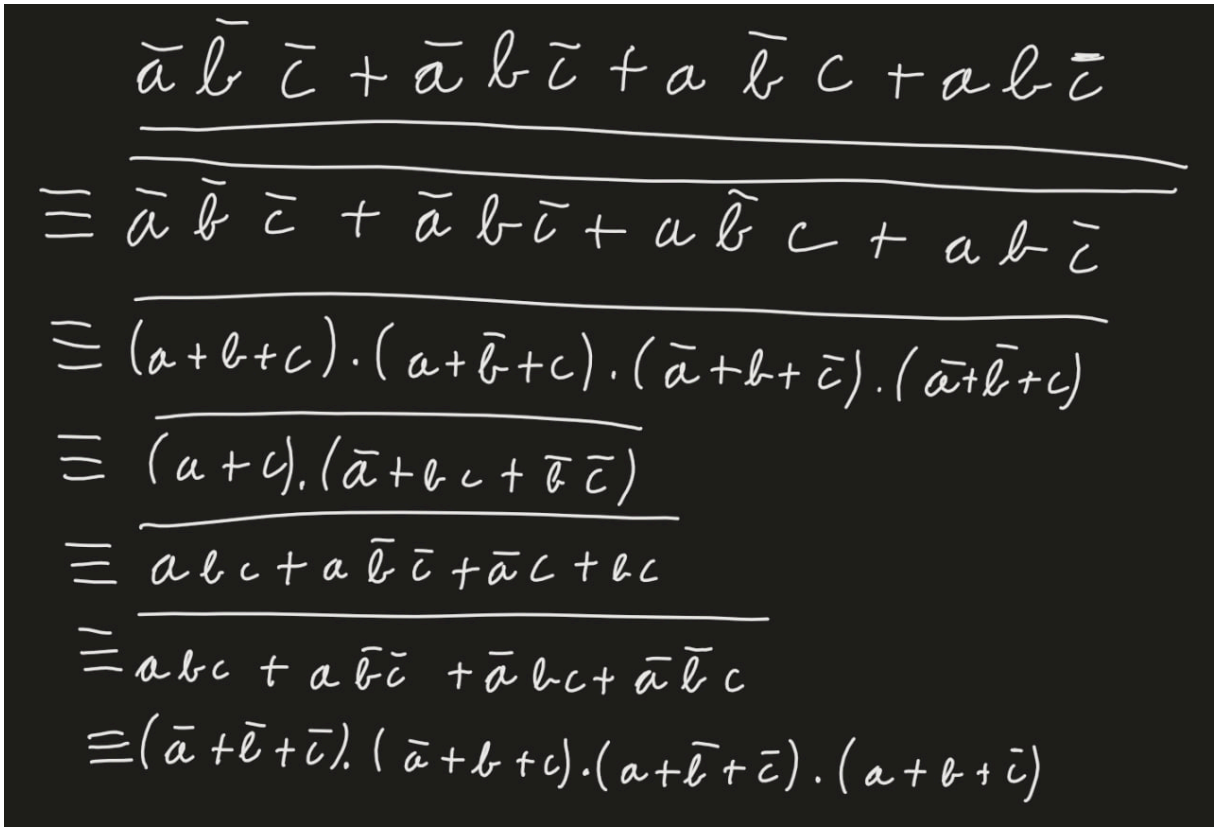
une variable ait une valeur différente. Et on combine le tout avec un “et” pour contrôler qu’on a une variable différente nous permettant de nous “échapper” de tous les zéros.

11. Comment faire une soustraction SIMPLE en binaire ?

- on choisit toujours le nombre le plus grand au-dessus (especially en IEEE-754), et si au lieu de faire a-b on fait b-a, on ajoute automatiquement un signe - au résultat.
- sinon, en 2’s complement il vaut mieux obtenir la représentation négative de notre nombre et **sommer** les deux.
- sinon, si on obtient une retenue infinie, on ne tient compte **que du premier 1**.

12. Comment passer d’une S.O.P à une P.O.S ?

pour passer d’une SOP à une POS, double négation, puis développement, puis application de la négation

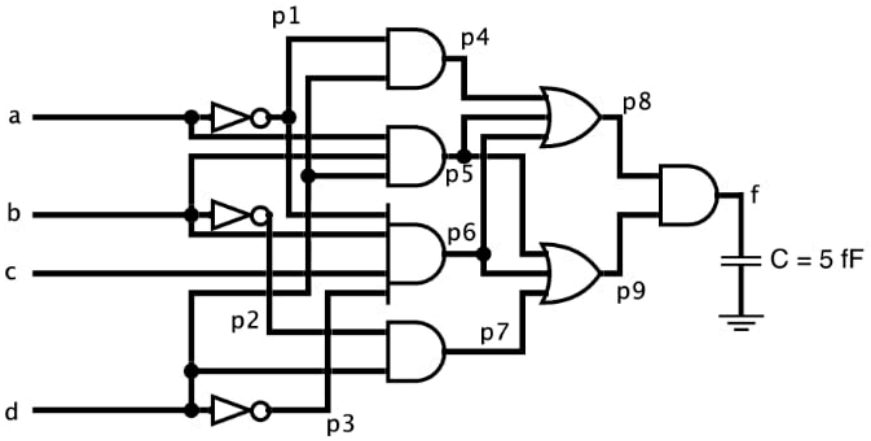


13. Comment comparer rapidement deux expressions

- si on a les minterms (les S.O.P), on peut les numéroter en fonction des termes positifs (par ex. ab(!c) représente 110, 6).
- si on a les maxterms (les P.O.S), on peut aussi les numéroter en fonction des termes négatifs (par ex. ab(!c) représente 001, 1).

Pour passer de maxterms M1, M4 à la liste des minterms, on prend les complément (m0, m2, m3, m5, etc.).

14. Comprendre les delays et les fan-in/fan-out



Pourquoi est-ce que l'inverter p2 prend 20ns à charger le gate AND à p7 et pas juste 10ns ? (comme il est branché à un seul input ?)

En fait c'est parce que les gate AND à 2 inputs ne sont pas construits de la même façon que les gates à 3 ou 4 inputs, et on affirme dans ce cours que le temps pour charger un input d'un gate est proportionnel au nombre d'inputs que ce gate doit gérer.

Ainsi charger un unique input (donc fan-out de 1) d'un gate à 3 inputs prend 3 fois plus de temps que charger un unique input d'un gate à 1 input.

15. Trick cool pour obtenir une full P.O.S

Quand on part d'une P.O.S pas complète, on ajoute juste les termes manquants en (a)(!a) par exemple, puis on factorise, et on utilise que a + (x)(y) = (a+x)(a+y)

$$(a+b)(b+c)(\overline{a}+\overline{c})$$
$$\equiv (a+b+\overline{c}c)(\overline{a}a+b+c)\Big(\overline{a}+\overline{b}b+\overline{c}\Big)$$
$$\equiv (a+(b+\overline{c})(b+c))((\overline{a}+b)(a+b)+c)\Big(\Big(\overline{a}+\overline{b}\Big)(\overline{a}+b)+\overline{c}\Big)$$
$$\equiv (a+b+\overline{c})(a+b+c)(\overline{a}+b+c)(a+b+c)\Big(\overline{a}+\overline{b}+\overline{c}\Big)(\overline{a}+b+\overline{c})$$
$$\equiv (a+b+\overline{c})(\overline{a}+b+c)(a+b+c)\Big(\overline{a}+\overline{b}+\overline{c}\Big)(\overline{a}+b+\overline{c})$$

ça donne une technique très simple pour obtenir une full P.O.S. : quand on a

$$(a+b) \equiv (a+b+\overline{c})(a+b+c)$$

16. Arrondir rapidement en floating point

On écrit la mantisse avec un bit de plus. On ajoute +1. On prend le résultat tronqué.

17. Addition/Soustraction en floating point

On aligne les exposants. On additionne les mantisses **en n'oubliant surtout pas de faire toujours apparaître le hidden bit!** On normalise.

18. Convertir un circuit en NAND et NOR

pour les NAND : écrire la Sum of Products (elle n'a pas besoin d'être une full SOP!), puis appliquer 2 fois la De Morgan.

pour les NOR : écrire la Product of Sums (elle n'a pas besoin d'être une full POS!), puis appliquer 2 fois la De Morgan.

ne pas oublier qu'on peut créer un inverter en mettant en chaîne deux NAND ou deux NOR, donc on peut toujours obtenir un OR ou un AND avec des NAND ou des NOR.

19. Checklist verilog

- ne pas oublier de déclarer en `reg` quand on fait du combinatoire
- le `case` n'a pas de `begin` ni de point-virgule mais un `endcase`
- ne pas oublier de faire un cas `default` dans les `case`
- ne pas oublier d'attribuer des valeurs par défaut aux variables modifiées en haut de tous les blocs `always`
- ne pas oublier la syntaxe `always @(variable1 or variable2)` (exécuté quand une des deux variables **change**) ou `always @(*)`
- ne pas oublier qu'on fait `[4:0]` pour déclarer un input à **5 variables** et pas `[0:5]` (mauvais sens + un bit de trop)

20. Un peu de cours

- in 2′s complement : complement and +1 to negate
- detect overflow : wrong sign (summing 2 nb of different signs never produces an overflow)
- PMOS : s’allume quand l’input est 0
- NMOS : s’allume quand l’input est 1
- (f est la fréquence, C la capacitance du condensateur, et V_DD la tension du circuit)

$$P_D = f * C * V_D^2$$

- weighted number systems:

$$x = \sum W_i * X_i$$

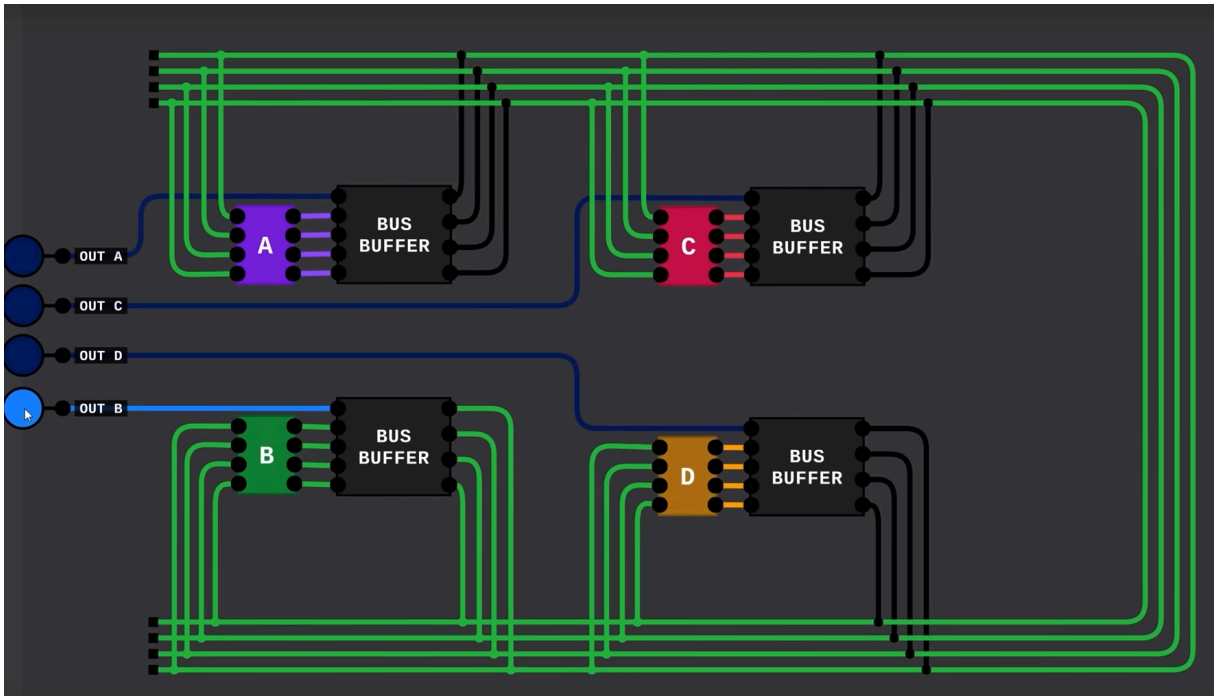
- radix number systems:

$$W_i = W_{i-1} * R_i$$

- canonical number system D means that each digit can take all the values between 0 and R (actually I don’t know any other representation)
- precision (fixed-point) : the maximum number of non-zero bits (m + f)
- resolution : the smallest non-zero magnitude representable
- range : the difference between the most positive and the most negative number representable
- accuracy : the magnitude of the maximum difference between a real value and its representation
- dynamic range : ratio of the max absolute value representable an the minimum non-zero positive value representable
- quand on additionne deux nombres en floating point, on garde l’exposant le plus grand (et on modifie la mantisse du plus petit pour qu’il ait le même exposant), pour minimiser l’erreur **note: c’est le même fonctionnement pour le block floating point!**

20.1. À quoi servent les tristate buffers?

Quand on a un bus de données (c'est à dire une sorte de voie sur laquelle des modules peuvent se connecter), on veut que ceux-ci puissent lire et écrire dedans. Sauf qu'on doit décider qui peut écrire à un moment donné, ils ne peuvent pas tous écrire en même temps :



Les tristate buffers permettent de résoudre ce problème. Ils ont 3 états : 0, 1, et Z (pour high impedance). Quand le tristate est à Z, il est déconnecté du bus (partie écriture), et les autres modules peuvent écrire dessus. Quand il est à 0 ou 1, il peut écrire sur le bus. (voir https://www.youtube.com/watch?v=_3cNcmli6xQ)

versus implémentation MUX (beaucoup moins propre) :