BUT 1, SAE 1.03, architecture des ordinateurs

Vincent Risch

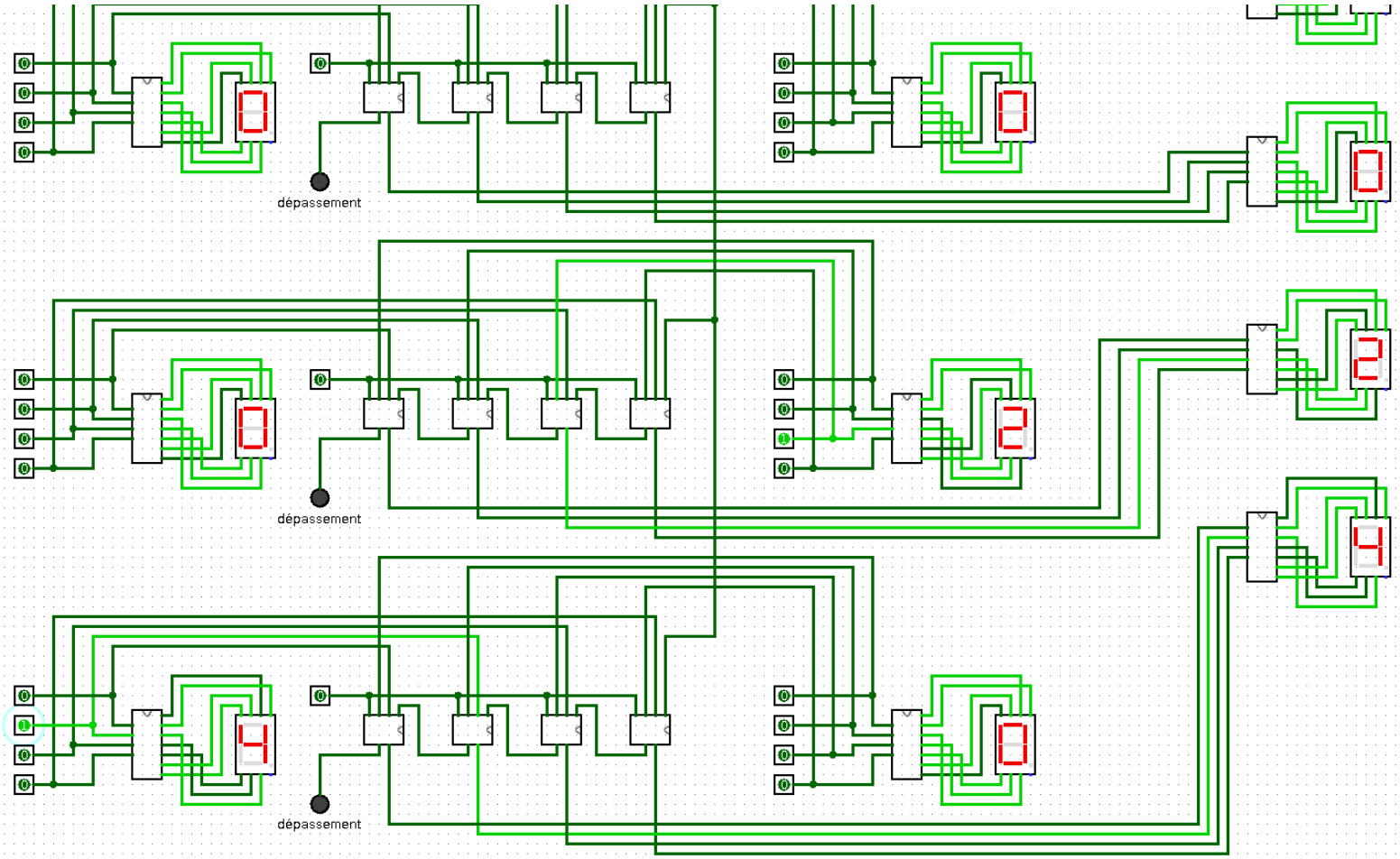
octobre 2022

Projets proposés

1. Conception d'une *calculette (addition/soustraction à quatre chiffres)*
2. Conception d'une *machine programmable RAM*

→ *Sur simulateur de circuit LOGISIM*
→ *Cette année, uniquement le projet 1*

Calculette



Calculette

- Première partie (addition/soustraction de deux chiffres)
- Seconde partie (addition/soustraction de deux entiers)

Calculette

- Première partie (addition/soustraction de deux chiffres)
  1. Réalisation d'un décodeur 7-segments utilisant l'afficheur 7-segments de LOGISIM *(Degré de difficulté : facile)*

- Seconde partie (addition/soustraction de deux entiers)

Calculette

- Première partie (addition/soustraction de deux chiffres)
  1. Réalisation d'un décodeur 7-segments utilisant l'afficheur 7-segments de LOGISIM *(Degré de difficulté : facile)*
  2. Conception d'un additionneur-soustracteur 4 bits $\times$ 4 bits *(Degré de difficulté : difficile)*
- Seconde partie (addition/soustraction de deux entiers)

Calculette

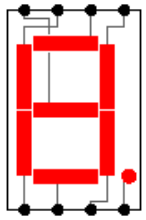
- Première partie (addition/soustraction de deux chiffres)
  1. Réalisation d'un décodeur 7-segments utilisant l'afficheur 7-segments de LOGISIM *(Degré de difficulté : facile)*
  2. Conception d'un additionneur-soustracteur 4 bits $\times$ 4 bits *(Degré de difficulté : difficile)*
- Seconde partie (addition/soustraction de deux entiers)
  1. Extension aux entiers de quatres chiffres en *décimal codé binaire* *(Degré de difficulté : difficile)*

Calculette

- Première partie (addition/soustraction de deux chiffres)
  1. Réalisation d'un décodeur 7-segments utilisant l'afficheur 7-segments de LOGISIM *(Degré de difficulté : facile)*
  2. Conception d'un additionneur-soustracteur 4 bits $\times$ 4 bits *(Degré de difficulté : difficile)*
- Seconde partie (addition/soustraction de deux entiers)
  1. Extension aux entiers de quatres chiffres en *décimal codé binaire* *(Degré de difficulté : difficile)*
  2. Réalisation et intégration au circuit réalisé d'un comparateur servant à la génération d'une retenue héxadécimale *(Degré de difficulté : difficile)*

Partie 1, 1 : Décodeur 7 segments

But : Concevoir le circuit qui à la représentation binaire d'un entier prise en entrée sur quatre bits $A, B, C, D,$, associe l'allumage de sept segments a, b, c, d, e, f, g permettant de représenter cet entier en base 10. L'afficheur sept segments disponible dans les entrées-sorties de LOGISIM est le suivant :



On nomme chacun des segments de la façon suivante (en oubliant le point) :

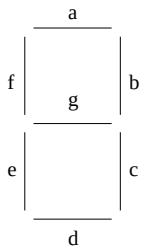


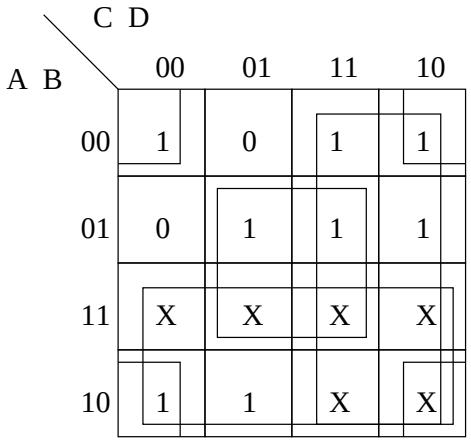
Table de vérité du décodeur 7 segments

Chacun des segments peut être vu comme une fonction booléenne des entrées, ce qui permet de générer une table de vérite décrivant complètement la valeur qui doit leur être affectée en fonction de la configuration des entrées. Les lignes comportant un x sont des lignes *a priori* inutilisées (puisqu'au delà de neuf, il n'est plus possible de représenter un entier sur un seul afficheur). La table suivante est donc à compléter :

A	B	C	D	a	b	c	d	e	f	g
0	0	0	0	1						
0	0	0	1	0						
0	0	1	0	1						
0	0	1	1	1						
0	1	0	0	0						
0	1	0	1	1						
0	1	1	0	1						
0	1	1	1	1						
1	0	0	0	1						
1	0	0	1	1						
1	0	1	0	x	x	x	x	x	x	x
1	0	1	1	x	x	x	x	x	x	x
1	1	0	0	x	x	x	x	x	x	x
1	1	0	1	x	x	x	x	x	x	x
1	1	1	0	x	x	x	x	x	x	x
1	1	1	1	x	x	x	x	x	x	x

Simplification de la fonction *a*

A partir de l'expression obtenue pour chacune des fonctions (par exemple *a*), il est possible de procéder à une simplification. Cela peut être réalisé *via* une table de Karnaugh dans laquelle la prise en compte des *x* permet de maximiser la simplification, voir transparent du cours intitulé *Circuits Booléens*). On obtient par exemple pour la fonction *a* :



$$\begin{aligned} a &= A + C + B.D + \overline{B}.\overline{D} \\ &= A + C + \overline{B \oplus D} \end{aligned}$$

Rappel utile : Relations fondamentales de l'algèbre de Boole

Relation	*Intitulé*
$x + \overline{x} = 1$	tautologie
$x.\overline{x} = 0$	contradiction
$\overline{\overline{x}} = x$	involution
$x + x = x$ $x.x = x$	idempotence
$x + 1 = 1$ $x.0 = 0$	élément absorbant
$x + x.y = x$ $x.(x + y) = x$	absorption
$x + \overline{x}.y = x + y$ $x.(\overline{x} + y) = x.y$	simplification
$\overline{x + y} = \overline{x}.\overline{y}$ $\overline{x.y} = \overline{x} + \overline{y}$	formules de De Morgan
$x.(y + z) = (x.y) + (x.z)$	distributivité de . sur +
$x + (y.z) = (x + y).(x + z)$	distributivité de + sur .
$x \oplus y = \overline{x}.y + x.\overline{y}$	expression de $\oplus$ en fonction de . et +
$x \oplus 0 = 0 \oplus x = x$	élément neutre pour $\oplus$
$x \oplus x = 0$	élément symétrique pour $\oplus$
$x.(y \oplus z) = (x.y) \oplus (x.z)$	distributivité de . sur $\oplus$
$\overline{x \oplus y} = \overline{x} \oplus y = x \oplus \overline{y}$ $\overline{x} \oplus \overline{y} = x \oplus y$	relations avec l'inversion

Partie 1, 2 : Additionneur-soustracteur

- Le circuit correspondant se base sur la construction de l'additionneur décrite dans le document *Circuits Booléens*. Ce circuit se décompose sous la forme de deux demi-additionneurs reliés par une porte « ou »
- rappel :
  - on appelle *demi-additionneur* un circuit capable de réaliser l'addition deux bits et de générer une somme et une retenue de sortie
  - on appelle *additionneur* un circuit capable de réaliser l'addition de deux bit *en tenant compte d'une retenue d'entrée* et de générer une somme et une retenue de sortie
  - un additionneur se réalise « facilement » en combinant de demi-additionneurs
- Le protocole à suivre pour la construction d'un additionneur-soustracteur est le suivant : étude d'un demi-soustracteur (important !), comparaison avec le demi additionneur pour réaliser un unique demi-additionneur-soustracteur, combinaison de deux demi-additionneur-soustracteurs pour réaliser un additionneur-soustracteur

Partie 2, 1 : opérations en DCB

La conception d'une calculatrice repose sur l'emploi d'une représentation des entiers en *Décimal Codé Binaire* (ou codage DCB). Dans ce code chacun des chiffres de la représentation décimale d'un entier est donné par son équivalent binaire sur quatre bits. Par exemple, la représentation de 9857_{10} est donnée par la séquence de quatre fois quatre bits :

9	8	5	7
1 0 0 1	1 0 0 0	0 1 0 1	0 1 1 1

Du coup, addition et soustraction en binaire doivent être adaptées à cette représentation...

Partie 2, 2 : addition et soustraction en DCB

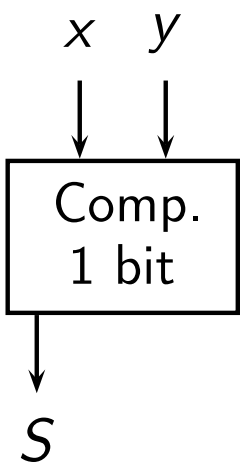
Le résultat d'une l'opération arithmétique sur deux chiffres *décimaux* doit être suivie d'une comparaison avec 9 (dernier chiffre représentable en base 10), en commençant par les poids faibles de la représentation binaire et en se déplaçant vers les poids forts. Quelque soit l'opération considérée sur deux chiffres décimaux, si le résultat de celle-ci sur un rang donné est supérieur à 9 (c'est-à-dire 1001_2) :

- pour l'*addition*, il est nécessaire d'ajouter 6 (pour dépasser 15, entier maximum représentable sur quatre bits) et générer ainsi la retenue propagée en décimal ;
- pour la *soustraction*, il est nécessaire de soustraire 6 (pour rester inférieur à 15) et générer ainsi la retenue propagée en décimal.

La circuiterie booléenne sous-jacente devra donc intégrer la conception d'un comparateur avec la représentation binaire de 9.

Partie 2, 2 : réalisation d'un comparateur

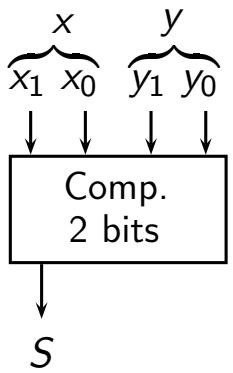
Etapes (possibles) de conception :
Réalisation d'un circuit (comparateur un bit) prenant en entrée un bit x et un bit y, et rendant en sortie un bit S résultats de la comparaison de x et de y :



- si $x > y$ alors S vaudra 1 ;
- si $x \leq= y$ alors S vaudra 0.

Partie 2, 2 : extension du comparateur 1 bit

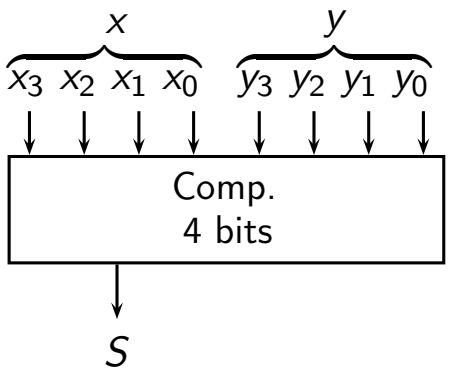
En utilisant plusieurs comparateurs un bit et des portes logiques usuelles, réalisation d'un circuit (comparateur deux bits) comparant deux entiers positifs x et y codés sur deux bits chacun :



On tiendra compte du fait que quels que soient deux entiers x et y, si $x = x_1x_0$ (où x_i représente un bit) et $y = y_1y_0$, alors $x > y$ ssi $(x_1 > y_1)$ ou $(x_1 = y_1$ et $x_0 > y_0)$. Ceci est généralisable quel que soit le nombre de bits constituant les x_i, pourvu que tout x_i ait le même nombre de bits que y_i.

Partie 2, 2 : comparateur 4 bits

A partir de trois comparateurs deux bits, réalisation d'un comparateur quatre bits :



La comparaison étant à effectuer avec 9 en binaire non-signé, le comparateur correspondant sera intégré à l'ensemble du circuit effectuant addition/soustraction en binaire pour réaliser les opérations correspondantes sur 8 chiffres décimaux, en tenant compte de la possibilité d'un dépassement de la représentation.

Partie Bonus et compléments

- Intégrer à la calculette la possibilité d'un affichage d'erreur (par exemple « E » sur un afficheur segment) pour signaler un dépassement de la capacité d'affichage de la calculette
- Implémenter la mutiplication
- Implémenter la division entière (sans affichage de reste)...

Format de rendu

A rendre : une archive .zip, nommée

`ProjetCalc.GR<NUMERO_DE_GROUPE>.<NOM_1>.<NOM_2>.<NOM_3>.<NOM_4>.zip`

<NOM_1>, <NOM_2>, ... étant les noms respectifs, en majuscules, des participants) et <NUMERO_DE_GROUPE> le numéro de votre groupe. L'archive en question devra contenir

- une description de votre implémentation comprenant la justification des choix opérés pour la réalisation du circuit, sous la forme de 3 à 5 pages max au format `.pdf` ;
- le circuit réalisé à l'aide du simulateur *logisim* sous la forme d'un fichier au format `.circ` contenant tous les sous-circuits réalisés.

Date et heure de rendu, à quel endroit : lundi 10/10 au soir, heure précise et url de dépot seront précisées rapidement.