Corrigé

ENSICAEN – 1ère année Electronique
Année 2019-2020 – Semestre 1

Electronique numérique

Lundi 25 novembre 2019 – 11h-12h30 – durée 1h30 – Matthieu Denoual
Aucun document autorisé

Consignes
Les documents, calculatrices et téléphones portables ne sont pas autorisés.
Les réponses seront données sur ces feuilles à l'intérieur des espaces prévus à cet usage.

Partie 1 : Numération et codage [/6]

Exercice 1 [/3] Complétez le tableau ci-dessous

Base 2 (12 bits)*	Base 10 **	Démarche, erreur de représentation
1111 0011 1010	-12,40	|-12,40| = 12,40 ; 0,25+0,125 = 0,375 ; erreur = 0,025 < 2^{-5} = 0,03125
0000 0100 0100	4,25	pondération des bits $2^{2}+2^{-2}$; erreur = 0 ; 4+0,25

* les nombres binaires seront représentés en complément à deux sur 12 bits virgule fixe $Q_{8,4}$. (Rappel représentation $Q_{m,k}$ sur N bits: $b_{m+k-1}b_{m+k-2}.....b_k,b_{k-1}.....b_2b_1b_0$; N=m+k)

** les nombres non entiers en base 10 seront représentés avec 2 chiffres significatifs derrière la virgule.

Exercice 2 [/3]

Codez les valeurs suivantes en format 32 bits virgule flottante selon la norme IEEE 754 rappelée en bas de page.

A = -5

Nombre négatif S = 1.

$A = -1 \times 1,25 \times 2^{2}$

$E - 127 = 2$

$E = 129 = 128 + 1$

E : 10000001

0,25 × 2 = 0,5 ; 0,5 × 2 = 1,0 ; 0. . . .

S | E_7 E_6 E_5 E_4 E_3 E_2 E_1 E_0 | f_{22} f_{21} f_{20} f_{19} . . . f_1 f_0
1 | 1 0 0 0 0 0 0 1 | 0 1 0 0 . . . 0 0

Rappel : représentation en virgule flottante suivant la norme IEEE 754.

La valeur X est représentée suivant la forme : $X = (-1)^{S} \cdot 2^{E-127} \cdot 1,F$

X s'écrit alors en binaire virgule flottante : $\underbrace{S}_{\text{signe}} \underbrace{e_7e_6\ldots e_1e_0}_{E} \underbrace{f_{22}f_{21}\ldots f_2f_1f_0}_{F}$; E et F sont codés en binaire non signé.

Partie 2 : Transcodeur 4B6B [/6]

Exercice 3 [/6]

Le transcodage 4B6B étend un code de 4 bits en un code de 6 bits contenant le même nombre de "1" et de "0".

L'objectif de cet exercice est la synthèse d'un transcodeur 4B6B selon la table ci-dessous.

	Code binaire 4B $b_3\ b_2\ b_1\ b_0$	Code 6B $g_5\ g_4\ g_3\ g_2\ g_1\ g_0$
0	0 0 0 0	0 0 1 1 1 0
1	0 0 0 1	0 0 1 1 0 1
2	0 0 1 0	0 1 0 0 1 1
3	0 0 1 1	0 1 0 1 1 0
4	0 1 0 0	0 1 0 1 0 1
5	0 1 0 1	1 0 0 0 1 1
6	0 1 1 0	1 0 0 1 1 0
7	0 1 1 1	1 0 0 1 0 1
8	1 0 0 0	0 1 1 0 0 1
9	1 0 0 1	0 1 1 0 1 0
A	1 0 1 0	0 1 1 1 0 0
B	1 0 1 1	1 1 0 0 1 0
C	1 1 0 0	1 1 0 0 1 0
D	1 1 0 1	1 0 1 0 0 1
E	1 1 1 0	1 0 1 0 1 0
F	1 1 1 1	1 0 1 1 0 0

Question 3.1 [/0.5]

Le code 6B est-il pondéré ?

non (pas de pondération associée).

Question 3.2 [/0.5]

Le code 4B est-il cyclique ?

non (la dernière représentation du code n'est pas adjascente à la première)

Question 3.3 [/1]

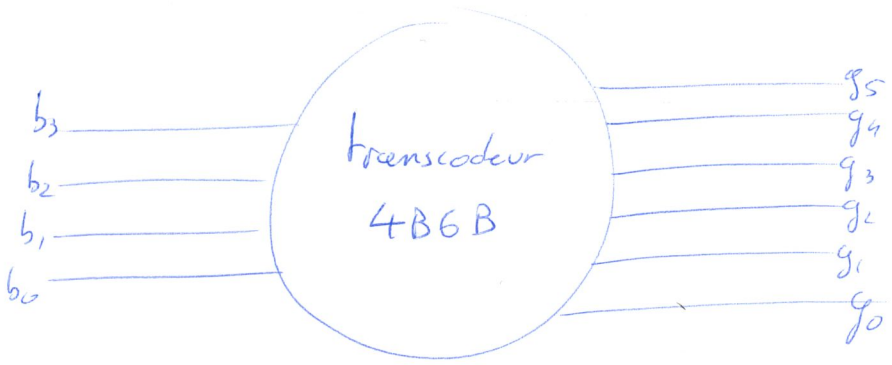
Quel(s) avantage(s) voyez-vous au code 6B ?

Il permet la détection d'erreur de transmission si l'on reçoit un nb de "0" différent du nb de "1" c'est qu'il y a une erreur.
Le code permet d'avoir la même puissance moyenne par tous les [illegible]

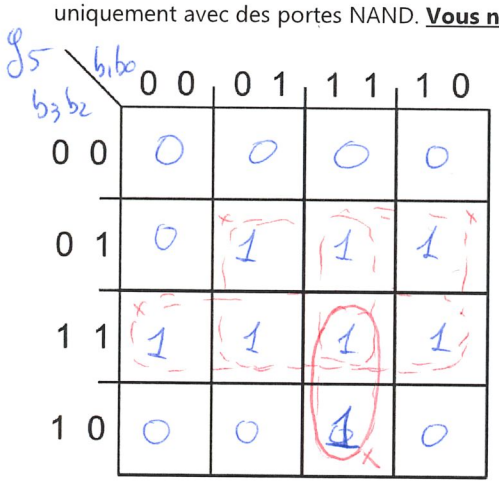
Question 3.4 [/4]

L'objectif est de synthétiser un transcodeur 4B6B. La synthèse conduira aux expressions des équations logiques pour une implémentation avec uniquement des portes NAND. Il n'est pas nécessaire de dessiner des schémas avec des portes.

Schématiser le transcodeur en identifiant ses entrées et ses sorties :

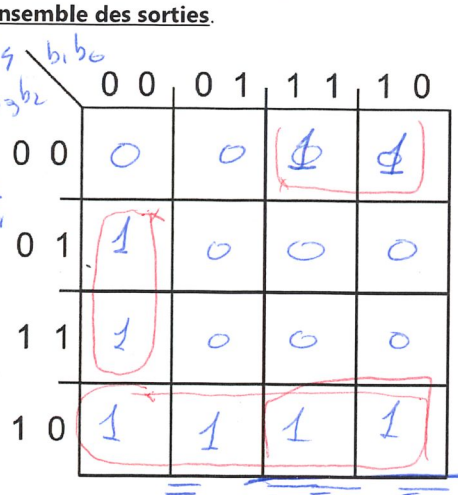


Utilisez les tables de Karnaugh ci-après pour réduire les expressions logiques donnant les sorties en fonction des entrées. Ecrivez sous chaque table l'expression logique réduite ainsi que l'expression mise en forme pour une implémentation uniquement avec des portes NAND. **Vous ne traiterez que 2 sorties sur l'ensemble des sorties**.



$q_5 = b_3b_1b_0 + b_3b_2 + b_2b_0 + b_2b_1$

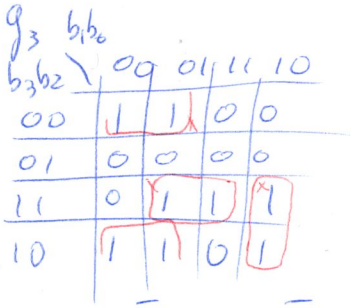
$\overline{q_5} = \overline{b_3b_1b_0 \cdot b_3b_2 \cdot b_2b_0 \cdot b_2b_1}$



$q_4 = b_3\bar{b}_1\bar{b}_0 + \bar{b}_2b_1 + b_3\bar{b}_2$

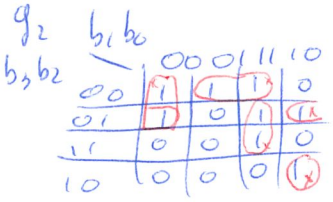
$\overline{q_4} = b_3\bar{b}_1\bar{b}_0 \cdot \bar{b}_2b_1 \cdot b_3\bar{b}_2$

$\overline{\overline{q_4}} = \overline{b_3\bar{b}_1\bar{b}_0 \cdot \bar{b}_2 \cdot b_1 \cdot b_3\bar{b}_2}$



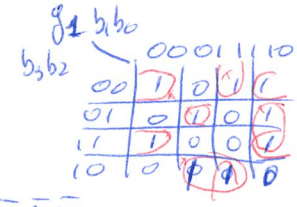
$q_3 = \bar{b}_2\bar{b}_1 + b_3b_1\bar{b}_0 + b_3b_2b_0$

$\overline{q_3} = \overline{\bar{b}_2\bar{b}_1 \cdot b_3b_1\bar{b}_0 \cdot b_3b_2b_0}$



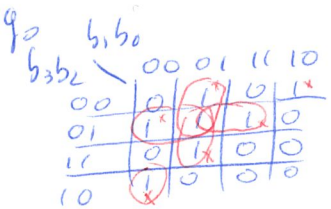
$q_2 = b_3\bar{b}_2b_1\bar{b}_0 + b_2b_1b_0 + \bar{b}_3b_2\bar{b}_0 + \bar{b}_3\bar{b}_2b_0 + \bar{b}_3\bar{b}_1\bar{b}_0$

$\overline{q_2} = \overline{b_3\bar{b}_2b_1\bar{b}_0 \cdot \bar{b}_2b_1b_0 \cdot \bar{b}_3b_2\bar{b}_0 \cdot \bar{b}_3\bar{b}_2b_0 \cdot \bar{b}_3\bar{b}_1\bar{b}_0}$



$q_1 = \bar{b}_3\bar{b}_2\bar{b}_0 + \bar{b}_3b_2\bar{b}_1b_0 + b_3\bar{b}_2b_0 + b_3b_1\bar{b}_0 + \bar{b}_2b_1b_0 + b_2b_1\bar{b}_0$

$\overline{q_1} = \overline{\bar{b}_3\bar{b}_2\bar{b}_0 \cdot \bar{b}_3b_2\bar{b}_1b_0 \cdot b_3\bar{b}_2b_0 \cdot b_3b_1\bar{b}_0 \cdot \bar{b}_2b_1b_0 \cdot b_2\bar{b}_1b_0}$



$q_0 = b_3\bar{b}_2\bar{b}_1\bar{b}_0 + \bar{b}_3b_2b_1\bar{b}_0 + \bar{b}_3b_2b_0 + b_2\bar{b}_1b_0 + \bar{b}_3b_2\bar{b}_1 + \bar{b}_3\bar{b}_1b_0$

$\overline{q_0} = \overline{b_3\bar{b}_2\bar{b}_1\bar{b}_0 \cdot \bar{b}_3b_2\bar{b}_1b_0 \cdot \bar{b}_3b_2\bar{b}_0 \cdot b_2\bar{b}_1b_0 \cdot b_3b_2\bar{b}_1 \cdot b_3\bar{b}_1b_0}$

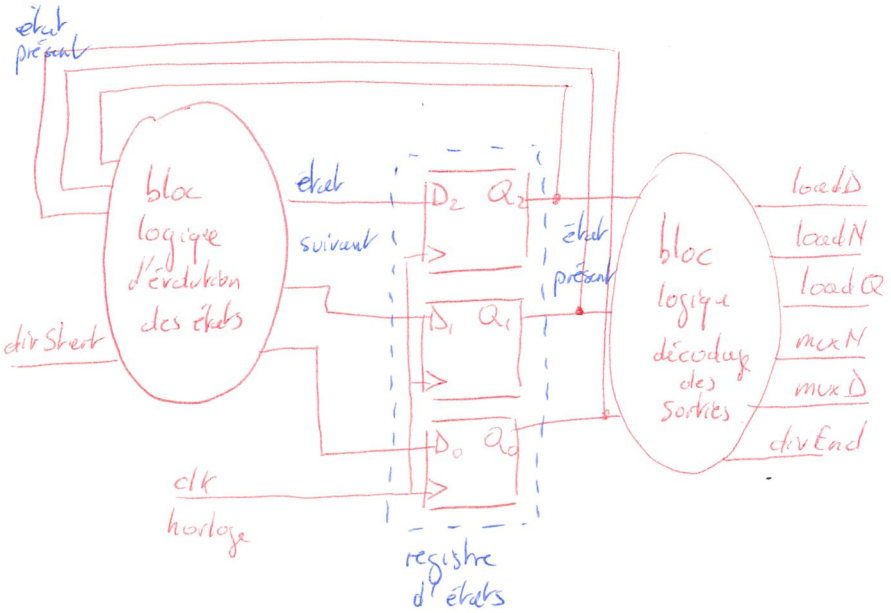
Question 4.1 [/1]

Quelle est la taille nécessaire et suffisante du registre d'états ?

il y a 8 états. En utilisant un codage binaire classique, 3 bascules sont nécessaires pour le registre d'états.

Question 4.2 [/2]

Représentez la machine à états finis de type Moore réalisée pour la synthèse de l'unité de contrôle. Précisez les entrées et les sorties de la machine à états finis.



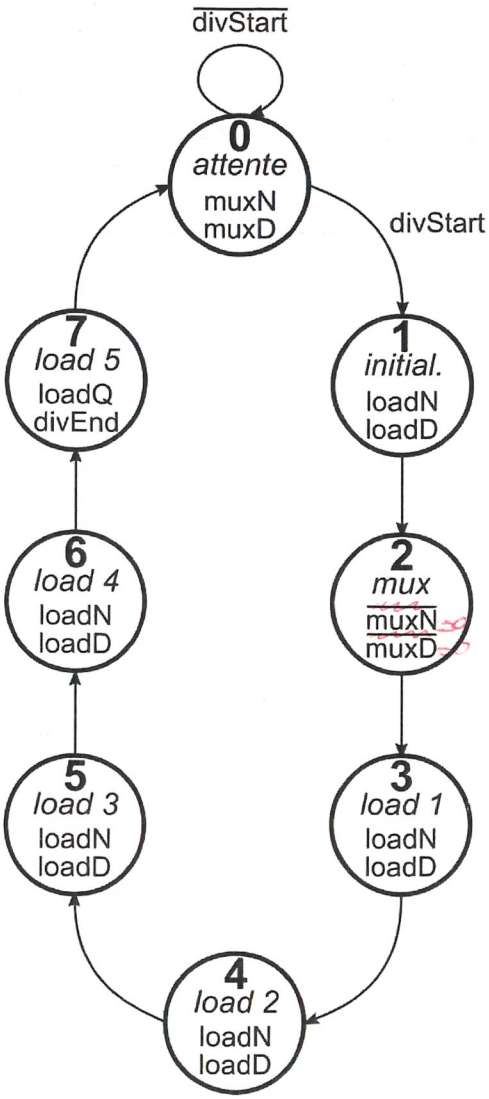


Figure 4: diagramme d'état du calcul de division

Question 4.3 [/5]

Synthétisez la machine à états finis de type Moore correspondant au diagramme d'états de la figure 4. La synthèse ira jusqu'aux expressions logiques.

Remarque : veillez à expliquer votre démarche.

Le registre d'état à 3 bascules est défini
il reste à obtenir les expressions des blocs logiques combinatoires.

états	code associé Q_2	Q_1	Q_0
0 attente initialisation	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0
7	1	1	1

Table de vérité du bloc logique d'évolution des états.

→ 4 entrées
3 sorties

	entrées				sorties		
	entrée système divStart	état présent Q_2	Q_1	Q_0	état suivant D_2	D_1	D_0
attente tant que divStart	0	0	0	0	0	0	0
démarrage →	1	0	0	0	1	0	1
indépendant de divStart	X	0	0	1	2	0	1	0
	X	0	1	0	3	0	1	1
	X	0	1	1	4	1	0	0
	X	1	0	0	5	1	0	1
	X	1	0	1	6	1	1	0
	X	1	1	0	7	1	1	1
	X	1	1	1	0	0	0	0

On utilise des tables de Karnaugh pour obtenir et simplifier les équations.

D_2

divStart Q_2 \ Q_1Q_0	00	01	11	10
00	0	0	1	0
01	1	1	0	1
11	1	1	0	1
10	0	0	X	0

$D_2 = \bar{Q}_2 Q_1 Q_0 + Q_2 \bar{Q}_0 + Q_2 \bar{Q}_1$

D_1

divStart Q_2 \ Q_1Q_0	00	01	11	10
00	0	1	0	1
01	0	1	0	1
11	0	1	0	1
10	0	1	0	1

$D_1 = \bar{Q}_1 Q_0 + Q_1 \bar{Q}_0 = Q_1 \oplus Q_0$

D_0

divStart Q_2 \ Q_1Q_0	00	01	11	10
00	0	0	0	1
01	1	0	0	1
11	1	0	0	1
10	1	0	0	1

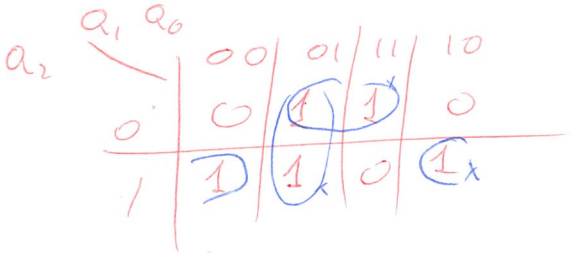
$D_0 = Q_2 \bar{Q}_0 + \text{divStart}\,\bar{Q}_0 + Q_1 \bar{Q}_0$

Table de vérité du bloc de décodage des sorties.
↳ 3 entrées
6 sorties.

entrées état présent Q_2	Q_1	Q_0	sorties. loadD	loadN	loadQ	muxN	muxD	divEnd
0	0	0	0	0	0	1	1	0
1	0	0	1	1	0	1	1	0
2	0	1	0	0	0	0	0	0
3	0	1	1	1	0	0	0	0
4	1	0	0	0	0	0	0	0
5	1	0	1	1	0	0	0	0
6	1	1	0	0	0	0	0	0
7	1	1	1	0	0	0	0	1

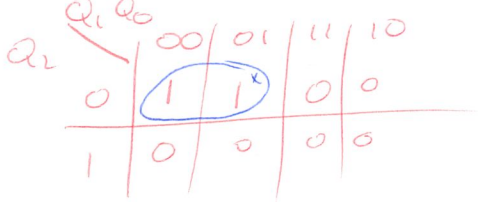
loadD = loadN = loadCalc ; muxN = muxD = muxData ; loadQ = divEnd

loadCalc



$loadD = loadN = Q_2\bar{Q}_0 + \bar{Q}_2 Q_0 + \bar{Q}_1 Q_0$

muxData



$muxData = \bar{Q}_2 \bar{Q}_1$

$divEnd = Q_2 Q_1 Q_0$

Question 4.4 [BONUS +2]

Quel(s) avantages(s)/inconvénient(s) voyez-vous à l'algorithme de division de type Godlschmidt comparé à celui que vous avez mis en œuvre en TP ?

Avantage : temps de calcul plus rapide.
moins de temps de cycle : $\log_2(m)$ au lieu de m pour l'algorithme de TP.

Par exemple pour un format 32 bits, avec l'algorithme de TP il faut 32 plus les cycles de chargement initiaux, ici il faut 5 cycles plus le chargement initial.

Inconvénient : l'unité de traitement est beaucoup plus grosse avec 2 multiplieurs et 1 soustracteur au lieu d'un ADD/SOUS pour l'unité de traitement en TP.

Question 4.5 [BONUS +3]

Proposer une méthode et éventuellement une structure permettant de normaliser les valeurs binaires du diviseur et du dividende.

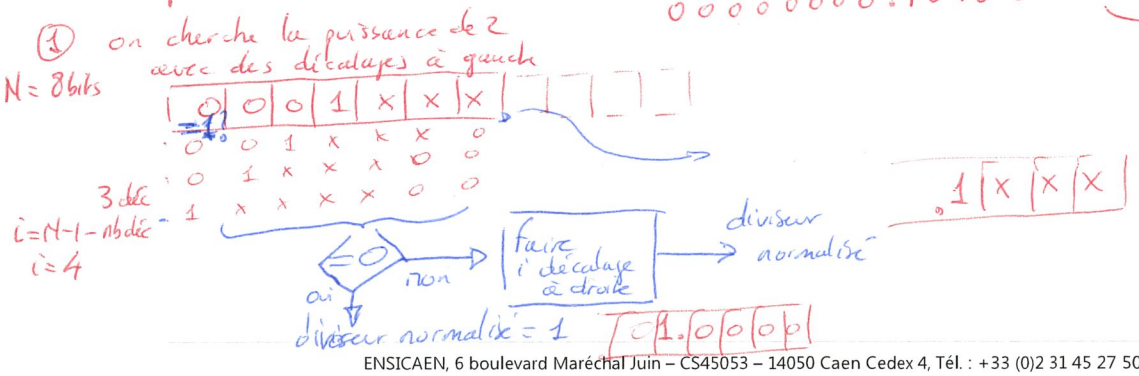
On veut normaliser le diviseur entre]0,5 et 1[.
Cela revient à diviser le diviseur par la puissance de 2 supérieure à sa valeur.

$2^2 < 5 < 2^3$
$4 < 5 < 8$

Une division par une puissance k de 2 correspond à k décalage à droite du nombre.

exemple 5 sur 16 bits Q8,8

00000101.0000 0000 → /8 : 3 décalages
00000000.1010 0000 → 0,625



Remarque : pour les diviseurs négatifs, on traite la valeur absolue.