Architecture des Circuits
Travaux Dirigés et Travaux Pratiques
Année 2025-2026

Préambule

Ce document contient les sujets des deux séances de Travaux Dirigés (TD) et des trois séances de Travaux Pratiques (TP) du cours d'Architecture des Circuits Numériques (3IF-AC). Il est rédigé de façon à vous permettre de réaliser les TD/TP pas à pas de façon autonome (ce qui, bien évidemment, ne vous interdit en rien de poser un maximum de questions à vos enseignants lors des séances, ils sont là pour ça).

Comme vous allez rapidement le constater, les sujets sont (bien) trop longs pour des séances de 2 (TD) ou 4 (TP) heures. C'est voulu car ce cours adopte un mode de fonctionnement particulier librement inspiré de la « pédagogie inversée ». Nous ne demandons aucun compte-rendu en TD/TP AC et les TD/TPs proprement dits ne sont pas notés. En revanche, nous vous demandons de **préparer les TD/TPs avant la séance** de façon à ce que vous puissiez vous concentrer, en séance, sur les points les plus difficiles, profiter au maximum de la présence de l'encadrant et, ainsi, aller jusqu'au bout de chaque sujet en séance : à l'évidence, les questions sont plus compliquées à la fin qu'au début ; en préparant les TD/TP avant, vous aurez la possibilité de profiter de l'expertise de vos enseignants sur les questions compliquées de la fin plutôt que sur les questions simples du début ...

Si l'on excepte les deux séances de TD initiales pour lesquelles une séance de TD correspond strictement à un chapitre de ce fascicule (respectivement aux chapitres 1 et 2), les trois séances de TP de correspondent pas strictement à des chapitres. L'objectif est de permettre à chacune et à chacun d'avancer à son rythme avant et pendant le TP. Le déroulement idéal des trois séances est le suivant :

Première séance de TP (semaine 45)

- Chapitre 3, *Circuits combinatoires*,
- Chapitre 4, *Circuits séquentiels simples*
- et Chapitre 5, *Circuits séquentiels plus complexes*.

Deuxième séance de TP (semaine 46)

- Chapitre 6, *Premières briques pour un afficheur défilant*
- Chapitre 7, *Implémentation d'automates — Afficher des motifs réguliers*
- et Chapitre 8, *Automate Parcourant La Mémoire*

Troisième séance de TP (semaine 47)

- Fin du chapitre 8, *Automate Parcourant La Mémoire*
- et Chapitre 9, *Une machine de Von Neumann Simple*

Si vous constatez que vous êtes en retard par rapport à ce déroulement, il est conseillé de rattraper ce retard entre les séances, les circuits réalisés étant réutilisés lors des séances suivantes. Vous pouvez aussi en discuter avec vos enseignants pour identifier et résoudre vos difficultés. Si, au contraire, vous êtes en avance, le sujet comporte plusieurs questions « bonus » qui vous permettront de renforcer certains points ou d'explorer des sujets plus avancés.

Vous remarquerez rapidement que les sujets comportent des questions théoriques et/ou des rappels de cours. Ces questions et ces rappels sont là pour vous permettre de vérifier régulièrement que vous avez bien la connaissance théorique nécessaire à la bonne réalisation des TPs. Veillez donc à ne pas les négliger (et même, si vous ne savez pas y répondre, à demander à un enseignant) sous peine de vous retrouver dans la panade quelques minutes plus tard ...

Table des matières

Chapitre 1

Représentation des nombres et arithmétique entière

Ce premier TD doit vous permettre d'acquérir quelques bases en arithmétique telle qu'elle se trouve implémentée au cœur des ordinateurs, dans le processeur même. Pour ce faire, nous allons discuter de la représentation des nombres entiers, d'abord naturels (ie positifs ou nuls, notés $\mathbb{N}$) puis relatifs (c'est à positifs ou négatifs, notés $\mathbb{Z}$). Ensuite, nous discuterons de quelques opérations arithmétiques de base sur ces entiers : addition, calcul de l'opposé, soustraction et enfin multiplication. Ces opérations représentent un sous-ensemble significatif du cœur de calcul d'un processeur : un processeur n'est bien qu'une grosse machine à calculer programmable.

1 Entiers représentables, ordres de grandeur et conversions

QUESTION 1 ► Quels sont les plus grands nombres codables en base 2 sur les tailles suivantes. On demande une valeur exprimée en décimal.

1 bit (valeur exacte)

4 bits (valeur exacte)

8 bits (valeur exacte)

1 byte (valeur exacte)

2 bytes (valeur exacte)

4 bytes (valeur approchée)

8 bytes (valeur approchée)

QUESTION 2 ► Pour chacune des paires de valeurs ci-dessous, laquelle est la plus grande ?

10^{33} et 2^{80}

10^{10} et 2^{35}

QUESTION 3 ► Une mole d'atomes de carbone contient 6.02×10^{23} atomes et pèse 12 grammes ...
En supposant qu'on trouve une technologie qui stocke un bit de mémoire par atome de carbone, est-ce que 2^{64} bits de mémoire pèseront plus ou moins que un gramme ?

QUESTION 4 ► En utilisant la technique décrite en cours, convertissez $(11011)_2$ en décimal.

QUESTION 5 ► Utilisez la méthode à base de divisions euclidiennes décrite en cours pour convertir $n = (414)_{10}$ en binaire.

QUESTION 6 ► En informatique on utilise très couramment la base 16 pour manipuler les nombres binaires de façon plus « humaine ». De fait, entre les bases 2 et 16, une méthode très directe peut être utilisée : tout chiffre hexadécimal est représenté par un entier sur quatre bits, et tout entier sur quatre bits est représenté par un chiffre hexadécimal. En utilisant cette méthode, convertissez $(34521)_{16}$ en base 2. Si vous n'êtes pas convaincu de l'intérêt de la base 16 ou si vous ne voyez pas à quoi elle sert, c'est le moment de poser la question à un enseignant !

2 Nombres entiers relatifs, codage en complément à 2

On s'intéresse maintenant à des entiers relatifs (donc positifs ou négatifs, ie dans $\mathbb{Z}$).

QUESTION 7 ► Écrire la table des correspondances binaire↔décimal pour tous les nombres entiers signés codés sur trois bits en complément à deux.

QUESTION 8 ► Comment pouvez-vous savoir si l'entier suivant, codé en complément-à-deux sur 16 bits, est positif ou négatif ?

$$1010\ 1110\ 1010\ 1111_{\bar{2}}$$

Donnez le code binaire de son opposé. Quelle est sa valeur ?

QUESTION 9 ► Donner la représentation binaire, en complément à 2 sur huit bits, des nombres suivants (tous négatifs). Attention : il n'est pas utile de faire systématiquement le calcul de conversion binaire ...

-11_{10}, -22_{10}, -44_{10}, -47_{10}, -125_{10}

QUESTION 10 ► Convertir en base 10 les nombres suivants codés en complément à deux :

$11111111_{\bar{2}}$

$10011111_{\bar{2}}$

$01001111_{\bar{2}}$

$111\ldots111110_{\bar{2}}$

3 Operations sur les nombres binaires

3.1 Addition

QUESTION 11 ► Dans un premier temps, nous allons supposer que les entiers ci-dessous sont des entiers naturels (donc non signé). Posez, sur 8 bits, les additions suivantes.

$$(10001010)_2 + (00001011)_2$$
$$(10001010)_2 + (10001011)_2$$
$$(01001010)_2 + (11001010)_2.$$

— Parmi ces additions, laquelle/lesquelles propage(nt) une retenue à gauche ?

— Pour chaque addition, comparez les ordres de grandeur des opérandes et du résultat.

— Comparez les réponses aux deux questions précédentes ; que pouvez-vous en déduire ?

QUESTION 12 ► Nous considérons maintenant des entiers relatifs codés en complément à deux. Posez, en complément à deux sur 8 bits, les additions suivantes.

$$(10001010)_{\bar{2}} + (00001011)_{\bar{2}}$$
$$(10001010)_{\bar{2}} + (10001011)_{\bar{2}}$$
$$(01001010)_{\bar{2}} + (11001010)_{\bar{2}}$$

— Parmi ces additions, laquelle/lesquelles propage(nt) une retenue à gauche ?

— Pour chacune de ces additions, quel est le signe des opérandes ? Quel est le signe du résultat ? En comparant ces trois signes, que pouvez-vous déduire sur ces additions ?

— Comparez les réponses aux deux questions précédentes ; que pouvez-vous en déduire ?

QUESTION 13 ► Faites le bilan des deux questions précédentes. Quelle est, d'après vos réponses, la différence entre une retenue (*Carry*, généralement notée "C") et un dépassement de capacité (*Overflow*, généralement noté "V" pour oVerflow) ?

3.2 Calcul de l'opposé en complément à deux

QUESTION 14 ► Calculez l'opposé de $(10001010)_{\overline{2}}$ en complément à 2 sur 8 bits, et vérifiez que votre résultat est correct.

QUESTION 15 ► En complément à 2 sur p bits, quel est le seul cas produisant un dépassement de capacité pour le calcul de l'opposé ?

3.3 Soustraction

Pour calculer une soustraction $A - B$, on commence par calculer l'opposé en complément-à-deux de B, puis on calcule l'addition $A + (-B)$. Comme l'addition, cette opération binaire peut se réaliser à l'identique, que A et B soient codés en complément à deux ou non. Attention, comme c'était aussi le cas pour l'addition (voir questions précédente), les dépassements de capacité n'auront pas lieu au même moment si A et B sont codés en complément à deux ou non ... Si vous ne comprenez pas cette remarque, c'est le moment de poser une question ...

QUESTION 16 ► En utilisant la méthode présentée ci-dessus, calculez en binaire la soustraction $1101_2 - 0110_2$

QUESTION 17 ► Interprétez 1101_2, 0110_2 et votre résultat comme des entiers naturels, et vérifiez votre calcul en convertissant opérandes et résultats en décimal.

QUESTION 18 ► Interprétez maintenant les opérandes (et le résultat) comme des entiers relatifs codés en complément à deux, que pouvez vous dire ?

3.4 Extension de signe en complément à 2

Soit une valeur V représentée en complément à deux sur n bits, on peut représenter la même valeur sur $m > n$ bits en duplication $m - n$ fois le bit de poids fort (ie le bit de signe) de V à gauche. Cette opération s'appelle une « Extension de signe ».

QUESTION 19 ► Comment sont représentés $(34)_{10}$ et $(-42)_{10}$ en complément à 2 sur 8 bits (complément à 2^8) ?

QUESTION 20 ► Comment sont représentés $(34)_{10}$ et $(-42)_{10}$ en complément à 2 sur 12 bits (complément à 2^{12}) ?

3.5 Multiplication

QUESTION 21 ► Effectuer – en binaire – les opérations suivantes :

$11111111_2 \times 1_2$

$11111111_2 \times 10_2$

$11111111_2 \times 100_2$

$11010011_2 \times 1001_2$

$1111_2 \times 1111_2$

Comparez la taille du résultat par rapport à la taille des opérandes. Quelle règle générale peut-on en déduire ?

QUESTION 22 ► Donner la représentation en hexadécimal de quelques nombres utilisés dans les exercices précédents. Comment procédez-vous et pourquoi ? Quel est l'intérêt du codage hexadécimal ?

4 Annexe au chapitre 1

4.1 Puissances de 2

2^0 = 1	2^{16} = 65 536	2^{32} = 4 294 967 296	2^{48} = 281 474 976 710 656
2^1 = 2	2^{17} = 131 072	2^{33} = 8 589 934 592	2^{49} = 562 949 953 421 312
2^2 = 4	2^{18} = 262 144	2^{34} = 17 179 869 184	2^{50} = 1 125 899 906 842 624
2^3 = 8	2^{19} = 524 288	2^{35} = 34 359 738 368	2^{51} = 2 251 799 813 685 248
2^4 = 16	2^{20} = 1 048 576	2^{36} = 68 719 476 736	2^{52} = 4 503 599 627 370 496
2^5 = 32	2^{21} = 2 097 152	2^{37} = 137 438 953 472	2^{53} = 9 007 199 254 740 992
2^6 = 64	2^{22} = 4 194 304	2^{38} = 274 877 906 944	2^{54} = 18 014 398 509 481 984
2^7 = 128	2^{23} = 8 388 608	2^{39} = 549 755 813 888	2^{55} = 36 028 797 018 963 968
2^8 = 256	2^{24} = 16 777 216	2^{40} = 1 099 511 627 776	2^{56} = 72 057 594 037 927 936
2^9 = 512	2^{25} = 33 554 432	2^{41} = 2 199 023 255 552	2^{57} = 144 115 188 075 855 488
2^{10} = 1024	2^{26} = 67 108 864	2^{42} = 4 398 046 511 104	2^{58} = 288 230 376 151 711 744
2^{11} = 2048	2^{27} = 134 217 728	2^{43} = 8 796 093 022 208	2^{59} = 576 460 752 303 423 488
2^{12} = 4 096	2^{28} = 268 435 456	2^{44} = 17 592 186 044 416	2^{60} = 1 152 921 504 606 846 976
2^{13} = 8 192	2^{29} = 536 870 912	2^{45} = 35 184 372 088 832	2^{61} = 2 305 843 009 213 693 952
2^{14} = 16 384	2^{30} = 1 073 741 824	2^{46} = 70 368 744 177 664	2^{62} = 4 611 686 018 427 387 904
2^{15} = 32 768	2^{31} = 2 147 483 648	2^{47} = 140 737 488 355 328	2^{63} = 9 223 372 036 854 775 808
			2^{64} = 18 446 744 073 709 551 616

4.2 Conversion Décimal/Hexadécimal/Binaire

Dec	Hex	Bin
0	0	0
1	1	1
2	2	10
3	3	11
4	4	100

Dec	Hex	Bin
5	5	101
6	6	110
7	7	111
8	8	1000
9	9	1001

Dec	Hex	Bin
10	A	1010
11	B	1011
12	C	1100
13	D	1101
14	E	1110

Dec	Hex	Bin
15	F	1111
16	10	10000
17	11	10001
18	12	10010
19	13	10011

Chapitre 2

Calcul booléen

Dans ce TD, on s'intéresse tout d'abord au calcul booléen dans l'exercice 1, puis dans les exercices suivants à l'équivalence expression booléenne $\leftrightarrow$ table de vérité $\leftrightarrow$ circuit combinatoire. Le dernier exercice détaille notamment la construction d'un multiplexeur et d'un démultiplexeur.

1 Calcul booléen

QUESTION 1 ► En reprenant les propriétés remarquables de l'algèbre de Boole (cours #2, transparent 6), écrivez les règles deux règles de distributivité. Convainquez-vous de deux choses :

- La distributivité de l'algèbre de Boole est *différente* de la distributivité en arithmétique (pourquoi ?)
- Si vous oubliez (ou déplacez) les parenthèses au cours de cette opération, le résultat sera potentiellement très différent (et donc faux).

QUESTION 2 ► Rappeler la règle de De Morgan sous ses deux formes.

QUESTION 3 ► Prouver les équivalences suivantes :

$$(1)\quad a.b + \overline{a}.b = b \qquad (2)\quad a + a.b = a$$

$$(3)\quad a + \overline{a}.b = a + b \qquad (4)\quad a.b + \overline{a}.c = a.b + \overline{a}.c + b.c$$

QUESTION 4 ► Simplifier les expressions suivantes, notamment grâce au théorème de De Morgan :

$$\begin{aligned} s_1 &= \overline{\overline{(x+y)}(x+z)}(y+z) \\ s_2 &= \overline{\overline{x.y} + yz} + \overline{(x+z)\overline{(y+\overline{x}z)}} \\ s_3 &= \overline{y(\overline{x}+z) + \overline{x(\overline{y}+z)}} + \overline{(y+z)}(xy + \overline{x(\overline{y}+\overline{z})}\,) \end{aligned}$$

2 Expression algébrique

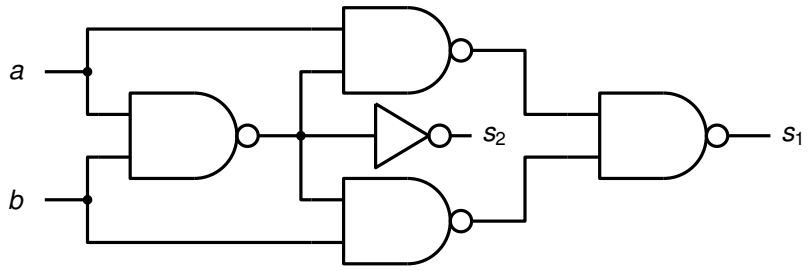
QUESTION 5 ► Donner une expression booléenne de la fonction $f(x, y, z)$ qui vaut 1 si et seulement si la majorité de ses trois arguments vaut 1.

QUESTION 6 ► Donner (sans chercher à la simplifier) une expression booléenne de la fonction $g(a, b, c)$ définie par la table de vérité suivante :

a	b	c	s
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

3 Circuits logiques

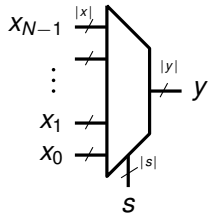
QUESTION 7 ► Donner une expression algébrique des sorties s_1 et s_2. Établir la table de vérité. La fonction calculée par ce circuit vous est-elle familière ?



QUESTION 8 ► Dessiner un circuit logique pour chacune des fonctions f et g de l'exercice précédent en utilisant uniquement des portes logiques (et, ou, non, xor, nand, nor)

4 Multiplexeur et démultiplexeur

On rappelle ci-dessous le symbole générique d'un multiplexeur à N entrées.



Intuitivement parlant, ce composant «recopie» l'une de ses entrées, x_i, sur sa sortie y. On choisit la valeur de i en positionnant l'*entrée de sélection* (notée s sur le symbole). Ainsi, si cette entrée de sélection est codée sur $|s|$ bits, elle permet de sélectionner parmi $N = 2^{|s|}$ entrées différentes. Par ailleurs, un multiplexeur peut être conçu pour travailler avec des données composées de plusieurs bits ; dans ce cas-là, la largeur $|y|$ de la sortie sera la même que la largeur $|x|$ des entrées.

QUESTION 9 ► Écrire la table de vérité complète d'un multiplexeur à 2 entrées de 1 bit chacune (on parle aussi de «multiplexeur 2-vers-1 à 1 bit»).

QUESTION 10 ► Donner une expression booléenne de la fonction réalisée par ce multiplexeur, et dessiner le circuit logique correspondant.

QUESTION 11 ► On s'intéresse maintenant à un multiplexeur à 4 entrées de 1 bits (on parle aussi de «multiplexeur 4-vers-1 à 1 bits»). Proposer une implémentation de ce circuit à l'aide de quelques instances du multiplexeur 2 vers 1 construit dans la question précédente. Dessiner le circuit logique correspondant.

QUESTION 12 ► Dessiner un circuit logique équivalent à un démultiplexeur 1-vers-4 à 1 bit.

Chapitre 3

Circuits combinatoires

1 Introduction

Au cours de cette séance de travaux pratiques, vous allez utiliser un simulateur (“Digital”) pour “réaliser” vos premiers circuits numériques. De façon à éviter de perdre trop de temps à “tirer des fils” (ce qui est rapidement un brin rébarbatif), nous limiterons la taille des mots manipulés par nos circuits à 8 bits [1]. Avant d’aller plus loin, vérifiez que vous êtes bien conscient.e qu’il n’y a pas de différence conceptuelle entre un circuit 8 bits et un circuit 16, 32 ou 64 bits. Si vous n’en êtes pas sûr.e, n’hésitez pas à demander ... Notez enfin que la largeur du mot manipulé “physiquement” (en matériel) par une machine ne limite pas la taille des mots qu’il peut manipuler “en pratique” (il faudra juste du logiciel pour assembler les mots matériels). Mais c’est une autre histoire ...

Cette séance se compose de trois parties à peu près équivalentes (correspondant aux chapitres 3, 4 et 5 de ce document) : dans un premier temps, vous allez vous intéresser à des circuits combinatoires simples (nous prendrons comme exemple l’additionneur 8 bits et le comparateur 8 bits) et vous en profiterez pour vous familiariser avec Digital. Vous passerez ensuite aux circuits séquentiels simples (registres 8 bits). Enfin, dans un troisième temps, vous réaliserez un compteur en combinant les différents éléments que vous aurez réalisés.

2 Prise en main de Digital

Digital est un simulateur de circuits numériques que nous allons utiliser dans les TP AC pour développer des circuits numériques (des circuits combinatoires puis séquentiels simples dans un premier temps, après quoi nous passerons à des circuits plus « complexes » : une mémoire et un compteur). Digital est installé sur les plateformes Linux du département IF [2], ici :

```
/opt/Digital/
```

Pour lancer Digital, il suffit de lancer `Digital.sh` dans le répertoire ci-dessus.

QUESTION 1 ► Afin de vous permettre de prendre en main l’outil, suivez le tutoriel de la documentation de Digital (menu `Help`→`Documentation`, section *A1.2, First steps*). Dès que quelque chose vous énerve, allez chercher la sérénité dans la FAQ de ce même document. Notez que la configuration standard de Digital utilise une représentation simplifiée pour les portes logiques. Vous pouvez passer à la version « classique » utilisée en cours via le menu `Edit`→`Settings`→`Use IEEE 91-1984 shapes`.

1. On peut croire que ça nous fait remonter à un passé lointain mais ... si les processeurs 8 bits sont apparus il y a 50 ans (dans les années 70 ; les plus connus étant le 8008 d’Intel, le Zilog Z80 et le Motorola 6800), ils sont toujours très utilisés dans des applications embarquées (mais "évidemment" pas dans un PC ...). Notez cependant que, bien que manipulant des mots de 8 bits, ces processeurs ont en général un bus d’adresse plus large afin d’augmenter les capacités mémoire ...

2. Digital est disponible ici : `https://github.com/hneemann/Digital`. (la requête Google qui le trouve est “digital logisim”). Cliquez sur “Download latest Release”, puis “Digital.zip” ; décompressez le fichier zip. Finalement, lancez `Digital.exe` sous Windows ou tapez dans un terminal `sh Digital.sh` sous Linux.

3 Additionneur 1 bit

Créez quelque part un répertoire `TP-Digital`. Vous y placerez tous les circuits demandés dans les différentes séances de TP.

QUESTION 2 ► Votre premier circuit dans ce répertoire réalisera l'addition 1 bit. Ce circuit possède trois entrées `a`, `b` et `c_in` représentant respectivement les deux valeurs à additionner et la retenue entrante de l'addition. Il possède deux sorties : `s`, dénotant la valeur de la somme, et `c_out`, dénotant la valeur de la retenue de sortie.
On vous rappelle la définition de `s` et `c_out` :

```
s = (a xor b) xor c_in
c_out = (a and b) or (a and c_in) or (b and c_in)
```

Par défaut, les portes ont 2 entrées. Si vous avez besoin d'un `AND` ou d'un `OR` à trois entrées (ou plus) vous pouvez changer l'attribut correspondant d'un clic droit de la souris. Vous pouvez aussi changer l'orientation des portes, complémenter certaines entrées des portes (très utile pour implémenter multiplexeurs et démultiplexeurs), etc.

Sauvegardez ce circuit dans un fichier nommé `add1bit`. Attention : il est important d'utiliser des noms informatifs car vous allez ensuite pouvoir réutiliser vos circuits pour en construire d'autres, plus complexes.

Au delà de l'exécution pas-à-pas, dirigée par le changement des valeurs d'entrées du circuit, que vous avez utilisée dans le tutoriel ci-dessus, Digital vous propose aussi un menu `Analysis` qui calcule la table de vérité de votre circuit (`Digital` propose plein d'autres outils d'analyse et de test que vous êtes fortement encouragés à essayer).

Une fois votre circuit terminé et analysé, si vous n'êtes pas sûr de vous, vérifiez avec un enseignant que le comportement observé à travers les informations fournies dans la fenêtre d'analyse est bien celui attendu.

4 Additionneur 8 bits

Pour construire un additionneur 8 bits, vous n'allez surtout pas copier-coller 8 fois les portes que vous venez d'utiliser pour l'additionneur 1 bit. Plutôt, vous allez encapsuler votre additionneur 1 bit dans un composant que vous pourrez ensuite réutiliser 8 fois pour obtenir l'additionneur 8 bits (cf cours arithmétique). Notez que vous pouvez aussi procéder en deux temps, par exemple en réalisant d'abord un additionneur 4 bits puis en le réutilisant deux fois pour réaliser un additionneur 8 bits.

QUESTION 3 ► Lisez la partie 1.4 de la doc ("Hierarchical design"). Créez un nouveau fichier Digital par `File→New Embedded Circuit`. Sauvez le tout de suite sous le nom `add8bits` dans le même répertoire que votre `add1bit`. Ce dernier devrait apparaître dans la liste des composants disponibles, sous la rubrique `Components→Custom`.

Placez 8 copies du composant `add1bit` sur le circuit et connectez-les pour réaliser un circuit additionneur 8 bits. Essayez de les placer intelligemment en réfléchissant dès le départ aux connexions que vous allez devoir cabler. Notez que Digital vous permet de choisir plusieurs dispositions pour les entrées-sorties du circuit `add1bit` en fonction de la disposition des entrées-sorties dans le circuit original ... essayez d'en tirer partie (si vous ne comprenez rien à ce paragraphe, jetez un oeil au cours "arithmétique" et/ou demandez un à enseignant et/ou allez jeter un oeil au menu `Edit→Circuit Specific Settings→Advanced→Shape` ... On vous conseille le `Shape` intitulé `Layout` ...).

Placez les ports de votre additionneur, nommez-les `a`, `b` et `c_in` pour les entrées et `s` et `c_out` pour les sorties et changez leur attribut "Data bits" quand nécessaire pour créer des entrées-sorties de plusieurs bits (des "bus"). Attention : pour connecter des bus à un ou plusieurs fils « simples » (ou à des bus de largeur différente), vous allez devoir utiliser des composants `Splitter` (menu `Components→Wires`) ; prenez le temps de comprendre leur fonctionnement, vous en aurez besoin !

Testez votre additionneur 8 bits et sauvegardez-le.

5 Comparateur

Maintenant que vous avez tout compris, réaliser un circuit combinatoire testant l'égalité entre deux mots binaires de 8 bits. Attention : il y a un piège ! En effet, quelle que soit la taille des mots binaires dont on testera l'égalité, la sortie du test sera toujours binaire (un seul bit) ... Vous n'utiliserez donc pas la technique d'encapsulation présentée ci-dessus mais, au contraire, vous réaliserez directement le comparateur 8 bits ...

QUESTION 4 ► En posant la table de vérité de l'égalité (sur un bit), trouvez la porte logique correspondant à cette opération.

QUESTION 5 ► En utilisant huit portes de ce type (et quelques autres bricoles ...), implémentez votre comparateur 8 bits. Testez-le et sauvegardez-le, vous en aurez besoin plus tard ...

Chapitre 4

Circuits séquentiels simples

Jusqu'à présent on s'est contenté de décrire des circuits combinatoires : la valeur des sorties de vos circuits (additionneur, comparateur) à un instant donné t ne dépend pas du passé, mais uniquement de la valeur de ses entrées à cet instant précis (au temps de propagation près bien sûr !).

C'est très bien, mais on ne va pas aller très loin avec ça. Très vite, on va vouloir faire des calculs qui vont nécessiter plusieurs opérations *successives* : à chaque étape, une opération combinatoire produira un résultat temporaire qui sera une opérande d'une opération à l'étape suivante. Ces étapes sont les fameux instants successifs de l'exécution d'un programme et leur implémentation demande d'utiliser des *circuits séquentiels*. L'implémentation de circuits séquentiels complexes sera abordée plus en détails dans les chapitres suivants. Ici nous allons nous limiter à des circuits élémentaires, les registres, et à leur utilisation dans un contexte simple, un compteur.

Pour pouvoir construire des circuits séquentiels, il va nous falloir des petits circuits pour mémoriser des valeurs, *d'un instant sur l'autre*. Vous allez maintenant découvrir comment on construit de éléments de mémorisation (des *registres*) en partant d'éléments simples appelés *verrous* (en anglais *latch*), eux-mêmes construits à partir de portes logiques. Ce chapitre aborde la construction de ces éléments.

1 Verrous et registres de 1 bit

NB : Créez chacun des circuits demandés séparément dans Digital, notamment pour pouvoir facilement les réutiliser plus tard.

1.1 Latch et flip-flop

QUESTION 6 ► Construisez un verrou (latch) puis une bascule D de 1 bit (un registre ou flip-flop) dans Digital.
Dans le poly du cours (chapitre 4.2) ainsi que dans les transparents, vous trouverez une description de ces deux composants. Vous trouverez le multiplexeur dans la bibliothèque de composants de Digital (rubrique `Plexers`).

1.2 Flip-flop avec reset

QUESTION 7 ► Complétez votre flip-flop afin de permettre sa remise à zéro (reset).

Pour cela, créez un nouveau circuit qui utilise votre flip-flop précédent et possède une entrée supplémentaire *reset*. Il existe deux grandes catégories de reset : le reset synchrone, qui est pris en compte lors du prochain front d'horloge et le reset asynchrone qui est pris en compte immédiatement. Ici nous vous demandons d'implémenter un reset synchrone. Attention : l'objectif premier du reset n'est pas de mettre à zéro la sortie de votre circuit (même si c'est ce qui va arriver lors du prochain front montant [1]) mais bien le *contenu* de votre mémoire de 1 bit.

1. Si vous ne comprenez pas cette parenthèse, ça mérite de poser une question à vos enseignants !

1.3 Registre à commande de chargement

QUESTION 8 ► Ajoutez une commande de chargement à votre registre.

Pour cela, encapsulez votre flip-flop à reset, et étendez-le avec une entrée supplémentaire *enable* : la valeur du registre n'est modifiée pour prendre la valeur d'entrée que si *enable* est vrai. Sinon, la sortie conserve sa valeur actuelle.

Remarque : Vous venez de construire un registre complet à 1 bit, qui vous permet de conserver une information sur plusieurs cycles d'exécution et de changer cette information à la demande, grâce à la commande *enable*. Plus précisément, c'est un "D-Flip-Flop" à un bit.

2 Registre à 8 bits

On ne sait pour l'instant que mémoriser 1 bit. Pour aller plus loin, il va nous falloir de quoi mémoriser des paquets (des vecteurs) de bits. En pratique, la taille des registres est très liée à d'autres éléments de l'architecture concernée : taille des bus, taille de la mémoire. Pour ce TP, nous nous limiterons à des registres 8 bits.
Contrairement aux différents verrous et flip-flop que nous avons construits jusque-là, la complexité ne se situe pas dans le comportement temporel mais dans le nombre des boîtes et des connexions constituant le circuit.

QUESTION 9 ► Construisez un registre 8 bits en collant côte-à-côte 8 registres 1 bit. Veillez à la synchronisation de l'ensemble.

Chapitre 5

Circuits séquentiels plus complexes

Dans les sections précédentes, vous avez réalisé des circuits combinatoires relativement simples (l'additionneur et le comparateur) et des circuits séquentiels relativement simples (registre 8 bits avec commande de chargement et de Reset). En utilisant tous ces circuits, vous allez pouvoir construire des circuits séquentiels aux comportements de plus en plus complexes, jusqu'à aboutir (dans deux séances) à une "vraie" machine de von Neumann[1].

1 Le compteur 8 bits simple

QUESTION 10 ► Sur le papier, essayez de trouver une méthode pour réaliser un compteur 8 bits en utilisant une horloge, un additionneur et un registre[2]. Si vous êtes sûr de vous, passez à la question suivante ; sinon, demandez à un enseignant de valider votre conception ... Veillez en particulier à bien comprendre à quel moment votre compteur est incrémenté et pourquoi.

QUESTION 11 ► À partir du schéma précédent, implémentez votre compteur sous Digital en utilisant votre registre 8 bits, votre additionneur 8 bits et une horloge (menu `Components→IO→Clock Input`). Testez-le. Notez que vous avez plusieurs solutions pour afficher l'état de votre compteur. La solution la plus simple consiste à utiliser une `Probe` (menu `Components→IO→Probe`) mais ce n'est pas très rigolo... Une solution plus rigolote consiste à utiliser des afficheurs 7 segments hexadécimaux (menu `Components→IO→More→Seven-Segment-Hex Display`) et des splitters.

2 Le compteur 16 bits

QUESTION 12 ► En utilisant votre compteur 8 bits, comment pouvez-vous réalisez un compteur 16 bits ? Dessinez votre solution. Si vous êtes sûr de vous, implémentez-la et testez-la. Sinon, demandez à un enseignant ...

3 Bonus : le compteur avec limite de temps

QUESTION 13 ► En repartant de votre compteur 8 bits (vous pouvez aussi partir de votre compteur 16 bits mais ça complique un peu plus la chose), comment pouvez-vous réaliser minuteur (c'est-à-dire un compteur qui s'arrête sur une valeur particulière spécifiée en entrée). Vous aurez bien évidemment besoin du comparateur réalisé en début de TP. Ce circuit n'est pas difficile à faire mais il comporte quelques subtilités. Faites donc vérifier votre circuit par un enseignant.

4 Bonus : le compteur-décompteur

QUESTION 14 ► En rajoutant une entrée à votre compteur (qui permettra de choisir le sens), réalisez un compteur-décompteur.

1. les guillemets sont là parce que, même si vous implémenterez effectivement une vraie machine de non Neumann, elle restera extrêmement simple, pour ne pas dire carrément frustre !
2. Notez qu'un compteur n'est pas censé s'arrêter lorsqu'il atteint sa valeur maximale (combien déjà ?) ; il continue à compter (et, du coup, quelle sera la valeur du compteur qui suivra la valeur maximale ?).

Chapitre 6

Premières briques pour un afficheur défilant

L'objectif de la série de travaux pratiques du cours d'Architecture des Circuits est de réaliser – sous Digital – une machine de von Neumann (en clair, un ordinateur) minimaliste, permettant d'afficher un message sur un afficheur défilant (tel que présenté en cours). La figure 6.1 présente une vue (statique) de ce que nous souhaitons obtenir (en pratique, le texte est censé défiler de droite à gauche ...).

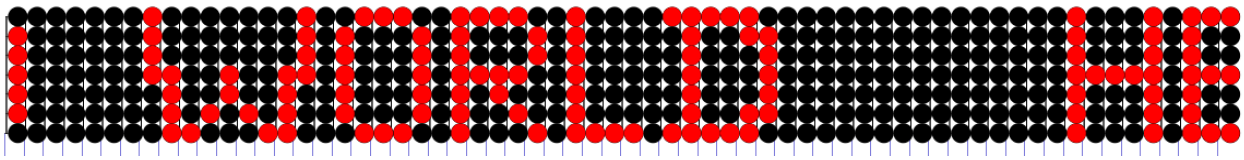


FIGURE 6.1 – L'afficheur défilant à 64 colonnes de 7 LEDs. Le texte défile de droite à gauche.

Notre afficheur est composé de 64 colonnes de 7 LEDs chacune. Son principe est le suivant : le circuit de contrôle envoie une information (sur 7 bits) à la colonne de droite. Cette information est ensuite transmise de colonne en colonne – de droite à gauche – ce qui permet d'obtenir l'aspect « défilant » de l'afficheur. Pour cela, chaque colonne dispose d'un registre de 7 bits et l'afficheur est synchronisé par un signal d'horloge : à chaque front montant de l'horloge, si la commande de chargement (signal EN pour ENable) est active, chaque colonne (chaque registre) enregistre la valeur de la colonne (du registre) située immédiatement à sa droite (sauf la colonne la plus à droite qui, elle, enregistre la valeur que lui envoie le dispositif de contrôle de l'afficheur).
Afin de bien maîtriser l'afficheur et son fonctionnement, nous allons commencer par le réaliser sous digital. Pour cela, ce chapitre vous donne des instructions très précises afin que vous ne perdiez pas trop de temps à cabler les 7×64 LEDs de l'afficheur (pour cela nous allons faire un usage abusif du copier-coller). Veillez donc à bien suivre ces instructions sous peine de passer l'intégralité des TPs d'AC sur ce chapitre !

1 Réalisation d'une colonne de l'afficheur

Nous allons commencer par réaliser une colonne de l'afficheur. Celle-ci sera organisés autour d'un composant SPLITTER à une entrée et sept sortie.

QUESTION 1 ► Pour implémenter une colonne de l'afficheur, placez un SPLITTER 8 vers 1*7 verticalement [1] et, dans ses options avancées, cochez la case « mirror » et spécifiez un écartement de 6 unités. Vous pouvez ensuite brancher directement 7 LEDs (en spécifiant une taille 4 et de la couleur qui vous amuse) sur les sept sorties de votre SPLITTER. Votre afficheur une colonne est terminé, reste à le contrôler. Pour cela vous allez le brancher sur votre registre 8 bits synchrone avec Reset et commande de chargement (Enable). Connectez ensuite les signaux de contrôle de votre registre sur trois tunnels (choisissez des

1. Il ne vous aura pas échappé que ce splitter ignore en sortie le 8ième bit d'entrée. Ce petit "truc" nous permettra de brancher notre afficheur sur un registre 8 bits synchrone – celui que vous avez réalisé lors des TPs précédents.

noms explicites, par exemple CLK, RST-AFF et EN-AFF[2]). Normalement vous avez désormais un machin ressemblant à la figure 6.2 ci-dessous[3] ...

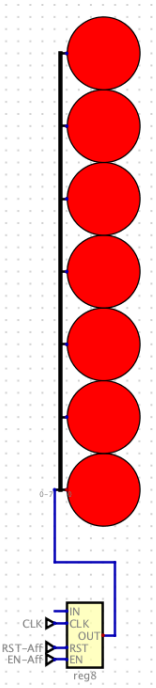


FIGURE 6.2 – L'afficheur une colonne

Votre première colonne est terminée, pensez à la sauvegarder ;) ...

2 Réalisation d'un afficheur 4 colonnes

QUESTION 2 ► En copiant-collant votre première colonne, réalisez un afficheur défilant de 4×7. Pour cela, vous devrez connecter correctement les quatre registres entre eux. Si vous ne voyez pas comment faire, relisez le début de ce chapitre ; si vous ne voyez toujours pas, appeler un prof ...
Une fois que vous aurez trouvé la solution, sachez que ce type de circuit s'appelle un *registre à décalage* ...
Vous pouvez tester votre circuit en lui ajoutant trois entrées de commande (que vous connecterez aux trois tunnels CLK, RST-AFF et EN-AFF) ainsi qu'une entrée binaire 8 bits sur la colonne de droite. Vous devriez obtenir un machin ressemblant à la figure 6.3 ci-dessous (avec un peu de filasse en plus mais on ne va pas vous donner la solution tout de même !).

3 Réalisation d'un afficheur 64 colonnes

QUESTION 3 ► En utilisant le copier-coller (et en connectant correctement les morceaux entre eux !), réalisez successivement des afficheurs 8, 16, 32 et 64 colonnes.

QUESTION 4 ► Une fois l'afficheur 64 colonnes terminé, connectez-le à une horloge (réglée sur 10 Hz) et à une entrée 8 bits pour le tester. Pour cela, il vous suffit de démarrer la simulation et de modifier manuellement les valeurs de l'entrée 8 bits (n'oubliez pas que le bit de poids fort est ignoré). À chaque changement de valeur un motif se déplace rapidement vers la gauche. Profitez-en pour réviser un peu le binaire et l'hexadécimal en comparant les motifs obtenus sur l'afficheur et les valeurs de l'entrée.

2. L'idée est de bien différentier, dans vos futurs circuits, les commandes Reset et Enable de votre afficheur de celles d'autres circuits. Notez que le problème ne se pose pas pour Clock puisque dans un circuit synchrone, il n'y a qu'une seule horloge !
3. L'implantation des broches sur le registre peut évidemment être différente suivant la façon dont vous aviez organisé votre circuit.

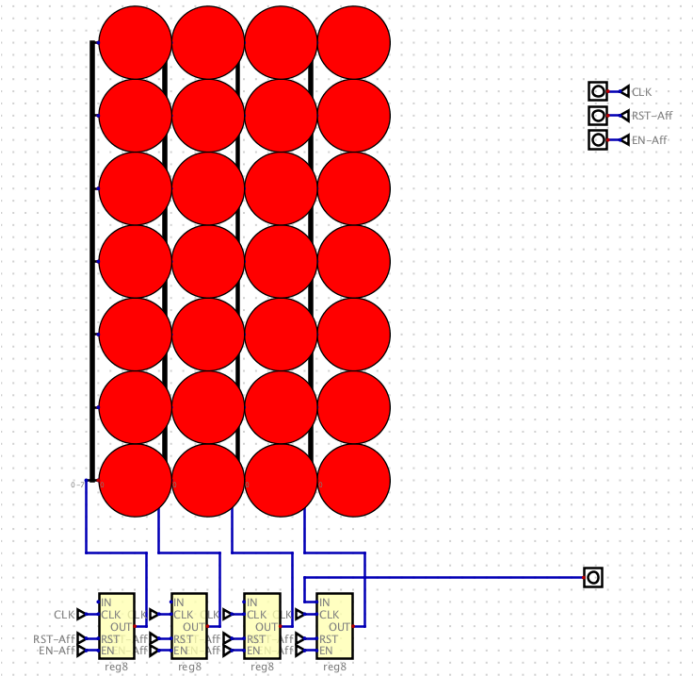


FIGURE 6.3 – L'afficheur quatre colonnes. Attention : la connectique des registres n'est pas complète, il faut réfléchir un peu ...

Chapitre 7

Implémentation d'automates — Afficher des motifs réguliers

1 introduction

Pour le moment nous avons commandé notre afficheur "à la main" mais vous aurez constaté que ce n'est pas très facile. Dans cette série d'exercices, nous allons implémenter une série de *machines à nombre fini d'états* (en anglais *finite state machine* ou FSM), de plus en plus complexes, de façon à envoyer à notre afficheur des signaux cycliques à intervalles réguliers. Cela nous permettra d'afficher des motifs réguliers comme illustré sur les figures 7.1, 7.2 et 7.3.

2 Affichage d'un motif simple

2.1 !!!

Dans un premier temps, nous allons réaliser un circuit très simple affichant des motifs comportant une seule colonne et séparés par un seul "espace" (figure 7.1).

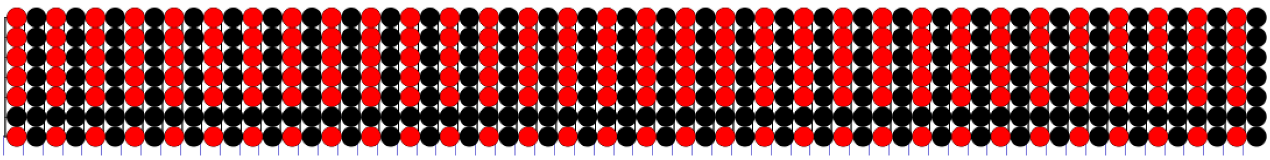


FIGURE 7.1 – L'afficheur défilant avec le motif simple

Nous allons concevoir un automate (plus précisément une machine de Moore) permettant d'envoyer à l'afficheur la séquence de signaux permettant d'afficher le motif souhaité.

QUESTION 1 ► Nous savons que notre afficheur fera défiler un motif composé de deux éléments (un espace et un point d'exclamation) et qu'il ne fera jamais rien d'autre. Cela nous permet de déduire plusieurs informations cruciales sur notre automate de contrôle de l'afficheur :

- Il aura deux états (on pourrait en mettre plus mais ce serait absolument inutile)
- Son alphabet d'entrée (I) est vide (notre automate ayant toujours le même comportement, il ne reçoit aucune commande)
- Son alphabet de sortie (O) comporte neuf bits (un pour chaque signal envoyé à l'afficheur)
- Il y a deux possibilités équivalente pour l'état initial q_0 puisqu'il n'est pas spécifié si l'affichage doit commencer par un espace ou par un point d'exclamation

Vérifiez que vous comprenez bien les explications précédentes ; dans le cas contraire, demandez impérativement à un enseignant de vous expliquer.

QUESTION 2 ► Dessinez votre automate en faisant bien figurer tous ces éléments. Si vous avez le moindre doute, faite-le valider par un enseignant.

QUESTION 3 ► Á partir des informations précédentes, on va construire notre automate. Combien de bits sont nécessaires pour coder son état (en utilisant un codage logarithmique) ? Pourquoi ? Combien d'équations booléennes comporteront respectivement les fonctions de transition (T) et de sortie (F) de votre automate. Pourquoi ? Posez les tables de vérité correspondant à ces deux fonctions et écrivez les équations booléennes correspondantes (petit indice : elles sont très très simples ... tellement simples que ça pourrait vous surprendre, voire vous dérouter).

QUESTION 4 ► En utilisant le schéma d'implémentation d'une FSM présenté en cours, implémentez votre automate et connectez-le à votre afficheur[1]. Pour votre implémentation, vous aurez besoin d'un registre d'état. C'est un registre de type D-Flip-Flop comportant impérativement une commande de reset[2] (si vous ne voyez pas pourquoi, demandez à un enseignant, c'est important !). Vous pouvez utilisez celui que vous avez déjà réalisé mais nous vous conseillons plutôt de vous simplifier la vie en utilisant celui registre fourni par Digital (menu `Components`→`Flip-Flops`→`D-Flip-flop, asynchronous`). Vous veillerez à connecter proprement les entrées `Clr` et `Set` (veillez à bien comprendre à quoi elles servent[3] !). Pour l'initialisation de votre circuit (c'est-à-dire pour choisir q_0), nous vous conseillons d'utiliser le composant `Reset` de Digital (menu `Components`→`Misc.`→`Reset`). Si vous ne comprenez pas comment il marche, n'hésitez pas à demander[4].

QUESTION 5 ► Testez, c'est cool !

2.2 !

Note préliminaire : si vous pensez être en retard, répondez aux deux premières questions ci-dessous et faites impérativement valider vos réponses par un enseignant. Vous pourrez ensuite faire l'impasse sur l'implémentation de votre solution et passer directement à la section 3.1.

Dans l'exercice précédent, nous avons affiché un motif simple (" !") séparé par un espace. Dans cette configuration, vous aurez noté que l'afficheur ne donne pas réellement d'impression de défilement mais plus une impression de clignotement. Pour donner une réelle impression de défilement, nous allons afficher le même motif mais cette fois, nous ferons en sorte que les " !" soient séparés par trois espaces (figure 7.2).

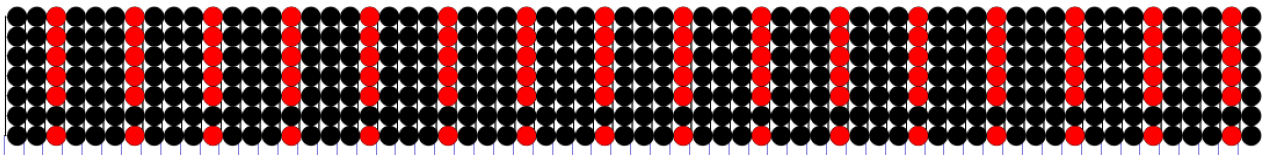


FIGURE 7.2 – Affichage de motifs simples séparés par trois espaces.

Dans un premier temps, vous allez modifiez votre automate pour qu'il réponde à cette nouvelle spécification.

QUESTION 6 ► Combien d'états comportera votre automate ; quel sera son alphabet d'entrée, quel sera son alphabet de sortie ? Si vous avez le moindre doute sur vos réponses, faites-les valider par un enseignant.

QUESTION 7 ► Dessinez votre automate.

QUESTION 8 ► En suivant la même procédure que pour la section précédente, écrivez les équations booléennes de T et F, implémentez votre circuit, testez ... c'est cool !

1. Vous avez deux solutions : soit vous implémentez votre automate dans un circuit autonome que vous encapsulerez et connecterez ensuite à votre afficheur, soit vous implémentez directement votre automate dans le même circuit que l'afficheur. La première solution est de loin la plus propre et la plus logique en termes de gestion de projet. Malheureusement, elle rend le déboggage beaucoup plus compliqué sous Digital. À contre-cœur nous vous conseillons donc d'adopter la deuxième solution ...

2. Notez que la commande de chargement n'est pas nécessaire ici. Là encore, si vous ne voyez pas pourquoi, demandez à un enseignant ...

3. On rappelle qu'en positionnant la souris sur l'entrée ou la sortie d'un circuit, Digital vous affiche une bulle d'aide vous indiquant la fonction de cette entrée ou de cette sortie.

4. On rappelle aussi qu'en positionnant la souris sur le circuit, Digital vous affiche une bulle d'aide vous indiquant son fonctionnement. Attention : dans le cas du circuit `Reset`, vous remarquerez que sa sortie est inversée. Vous pouvez changer ça dans la configuration du `Reset`.

3 Affichage commandé

3.1 < < < < < < < < < < < < < < < < < <

Nous allons maintenant afficher un motif plus complexe (figure 7.3[5]). En outre, comme l'illustre la figure 7.3, nous allons rajouter la possibilité de commander l'afficheur. Pour cela, nous rajouterons deux entrées à notre circuit. L'entrée `Show` et l'entrée `Clear`. Lorsque `Show` est active, le motif défile sur l'afficheur ; lorsque `Show` est inactive, l'afficheur reçoit des lignes vides (mais le défilement continue). Lorsque `Clear` est active, l'afficheur s'éteint immédiatement (alors que, lorsque `Show` passe de l'état actif à l'état inactif, l'afficheur ne s'éteint pas immédiatement : le motif affiché continue de défiler jusqu'à "sortir" à gauche).

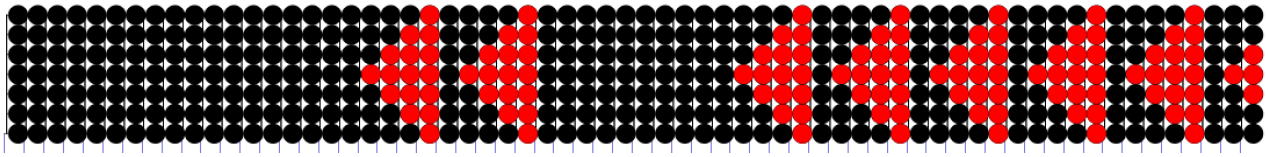


FIGURE 7.3 – Affichage d'un motif plus complexe avec commande d'affichage.

QUESTION 9 ► Quel seront les alphabets d'entrée et de sortie de votre automate ? Si vous avez le moindre doute sur vos réponses, faites-les valider par un enseignant.

QUESTION 10 ► Dessinez votre automate ; combien d'états a-t-il ? Si vous n'êtes pas sûr de votre automate, faites-le impérativement valider par un enseignant avant de passer à la suite ...

QUESTION 11 ► En suivant la même procédure que pour les sections précédentes, écrivez les équations booléennes de T et F, implémentez votre circuit, testez ... c'est cool !

3.2 Bonus

3.2.1 Bonus 1 : afficher des motifs complets

Lors de la question précédente, vous aurez constaté que, lorsque vous stoppez l'affichage (en passant `Show` de 1 à 0), vous pouvez obtenir des motifs incomplets.

QUESTION 12 ► Modifiez votre automate et votre circuit pour garantir que tous les motifs affichés le soient toujours complètement.

3.2.2 Bonus 2 : < < < < < < < < < < < < < < < <

Attention : ce bonus est assez complexe, veillez à ne vous engager dans ce travail que si vous êtes en avance ...

On veut pouvoir modifier à la demande l'intervalle entre les motifs. Celui-ci pourra comporter de un à quatre "espaces", ce nombre étant spécifié par une entrée binaire à deux bits.

QUESTION 13 ► Vous connaissez la chanson : combien d'états ? Alphabets d'entrée, de sortie ? Dessinez, équationnez[6], implémentationnez[7], testez ... c'est cool !

5. Vous pouvez afficher un motif différent si vous le souhaitez ; la seule contrainte que nous vous imposons est qu'il soit du même ordre de complexité que celui que nous vous proposons.
6. Un "n" ou deux "n" ?
7. J'arrête là avec les footnotes ...

Chapitre 8

Automate Parcourant La Mémoire

1 Introduction

Dans le chapitre précédent, nous avons pu contrôler notre afficheur déroulant pour afficher des motifs réguliers. Pour cela nous avons utilisé une machine à nombre fini d'états, ce qui nous a permis d'envoyer à l'afficheur des signaux réguliers. Cette solution est cependant très limitée et ce pour au moins deux raisons. D'une part, le nombre d'états de l'automate (et donc la complexité du circuit de contrôle) dépend de la taille du motif que l'on souhaite afficher. D'autre part, toute modification du motif entraîne une refonte totale du circuit de contrôle. Si on veut pouvoir afficher des motifs compliqués (par exemple des textes – dont le fameux "HELLO WORLD" présenté en exemple au chapitre 6), il va falloir adopter une autre solution. Pour cela nous allons stocker le motif correspondant au texte dans une mémoire et lire séquentiellement cette mémoire pour envoyer à l'afficheur les signaux à afficher.

2 Utiliser une mémoire

Avant toute chose, il est important que vous vous familiarisiez avec le concept de mémoire et avec le circuit Digital correspondant. Digital vous propose plusieurs types de mémoire (menu `Components`→`Memory`) dont plusieurs types de RAM et de ROM ...

QUESTION 1 ► Faites un petit tour sur Internet pour bien comprendre la différence entre une RAM et une ROM ...
Dans la suite du TP, nous utiliserons une ROM afin de garantir que le contenu de la mémoire soit bien conservé d'une simulation à l'autre (si vous ne comprenez pas cette phrase, retournez à la question précédente).

QUESTION 2 ► Créez un circuit de test contenant une ROM (`Components`→`Memory`→`ROM`→ `ROM, separated Ports`). Configurez votre ROM pour qu'elle contienne des mots de 8 bits adressés par des mots de 8 bits [1]. Quelle sera, en octets, la capacité de cette mémoire ?

QUESTION 3 ► Connectez votre ROM à une série d'entrées-sorties simples. Manipulez-là jusqu'à être certain de bien comprendre comment elle fonctionne. Notez que vous pouvez visualiser le contenu de votre ROM (clic droit, menu `Data`→`Edit`). Vous pouvez aussi modifier le contenu de votre ROM (via la même interface) mot par mot. Vérifiez que vous comprenez bien comment le contenu de la mémoire est affiché/modifié (ce que vous observez ici est ce qu'on appelle en architecture un "Dump" mémoire – ce qui est du joli franglais soit dit en passant ;). Par exemple, vérifiez que vous savez où se trouve l'octet d'adresse `0xB6` et que vous savez initialiser votre mémoire en écrivant `0xCD` à cette adresse. Fermez le menu `Edit` et vérifiez, grace aux entrées-sorties de votre mémoire, que la bonne valeur est bien au bon endroit.

3 Parcourir une mémoire

Parcourir une mémoire indique qu'on va lire son contenu octet par octet. Pour cela, vous aurez besoin d'un registre stockant l'adresse courante et d'un circuit permettant de modifier le contenu de ce registre. Ca vous rappelle quelque-chose ?

1. Notre afficheur n'affichant que des mots de 7 bits, le bit de poids fort des mots mémoire sera systématiquement mis à zéro. Si vous ne comprenez pas cette note, demandez à un enseignant ...

QUESTION 4 ► En utilisant la ROM de la question précédente, le compteur 8 bits du TP1 et votre afficheur simple (celui du chapitre 6), réalisez un circuit parcourant la mémoire et envoyant séquentiellement son contenu à l'afficheur. Notez que nous ne prévoyons pas de pouvoir écrire dans cette mémoire. Du coup les entrées de la mémoire seront toutes connectées à des constantes (sauf l'entrée `A` bien sûr !).

QUESTION 5 ► Il ne reste plus qu'à entrer les bonnes valeurs dans votre mémoire pour afficher votre `HELLO WORLD`. Pour cela vous pouvez utiliser l'interface d'`Edit` comme précédemment mais c'est un peu fastidieux, surtout pour corriger les erreurs ! Pour cette étape, nous vous conseillons plutôt de charger le contenu de votre ROM à partir d'un fichier (notez que cette procédure s'avérera quasiment indispensable pour le TP3). Bien entendu rien ne vous interdit d'afficher autre chose qu'`HELLO WORLD` mais prenez quand même un texte suffisamment long pour que l'exercice ait un sens ...

Initialiser une ROM depuis un fichier de données dans `Digital`

La procédure se déroule en trois étapes :

1. La première étape consiste à indiquer à `Digital` que votre ROM contiendra un programme et que ce programme devra être chargé à chaque lancement du circuit (Clic droit sur la ROM, onglet `Advanced` et cochez la case `Program memory`).
2. Ensuite, entrez quelques valeurs dans votre ROM, puis enregistrez-là sous le nom `prog.hex`.
3. Ouvrez `prog.hex` à l'aide de l'éditeur de texte de votre choixet retrouvez-y vos petits. Ne touchez pas à la première ligne de votre fichier contiendra l'entête suivante : `v2.0 raw`. La suite du fichier contiendra ligne par ligne les mots mémoire en hexadécimal (un mot par ligne, il n'est pas nécessaire de spécifier l'intégralité de la mémoire). Changez quelques valeurs, sauvegardez, puis dans Digital chargez le fichier dans votre ROM.
4. Il ne reste plus qu'à relier dans `Digital` votre circuit avec le fichier. Pour cela, dans le menu `Advanced`, cochez la case `Preload memory at startup.` et indiquez le nom du fichier qui devra être chargé dans la ROM au démarrage de la simulation.

QUESTION 6 ► Testez votre circuit. Aux bugs binaires prêt, normalement il marche ... Mais pourtant il a un problème (que nous résoudrons au prochain chapitre ...). Lequel ?

Chapitre 9

Une machine de Von Neumann Simple

1 Introduction

Au cours du chapitre précédent, vous avez pu afficher un “motif complexe” (un texte en l'occurrence) sur votre afficheur défilant en parcourant une mémoire de 256 mots à l'aide d'un compteur 8 bits. Mais cette solution présente un défaut : quelle que soit la longueur du texte/motif affiché, votre contrôleur boucle tous les 256 pas de temps (c'est-à-dire lorsque votre compteur retourne à zéro lors du dépassement de capacité de l'addition [1]). Pour corriger ce point, nous allons contrôler notre afficheur au moyen d'une machine de von Neumann. Celle-ci sera cependant très simple puisqu'elle ne comportera que quatre instructions.

2 Une machine de von Neumann à quatre instructions ...

La machine de von Neumann que nous allons concevoir et implémenter comportera donc trois instructions (`Show`, `Jump`, `Clear` et `Nop`), toutes les quatre codées sur 8 bits. Ces 8 bits seront séparés en code opératoire (“opcode”) et opérande. Cependant, comme souvent dans les machines réelles, cette distinction sera un peu complexe puisque l'opcode fera un ou deux bits selon l'instruction (l'opérande faisant 6 ou 7 bits – voire zéro – toujours en fonction de l'instruction). En effet, si le bit de poids fort est zéro, alors l'instruction sera `Show` et les sept bits restant forment l'opérande. En revanche, si le bit de poids fort est à un, alors c'est le bit 6 qui détermine si l'instruction est `Jump` (bit 6 à zéro) ou `Clear` (bits 6 à un). Dans le premier cas l'opérande compte donc 6 bits (bits 0 à 5) tandis que dans le deuxième cas il n'y a pas d'opérande (ce qui laisse de la place pour coder de nouvelles instructions si on veut ...). Enfin, l'instruction `Nop` sera effectuée en utilisant un cas particulier du `Jump` (son opcode est donc le même que celui du `Jump`). Les quatre instructions ainsi que leurs codages sont détaillées ci-dessous :

`Show` L'instruction `Show` est la plus simple des trois : elle envoie à l'afficheur les sept bits à afficher puis passe à l'instruction suivante (celle située à l'adresse suivante dans la mémoire). L'opcode de l'instruction `Show` est 0 (zéro), situé sur le bit de poids fort (B7) ; l'opérande est placé en bits B0 à B6.

`Jump` L'instruction `Jump` permet de “sauter” vers une instruction quelconque (alors que, dans une machine de von Neumann, par défaut les instructions se suivent séquentiellement dans la mémoire). Pour cela, l'instruction `Jump` utilise un codage d'adresse par décalage (“offset”) dont la valeur est l'opérande de l'instruction (codé sur 6 bits). L'adresse de l'instruction suivant le `Jump` sera donc l'adresse du `Jump` plus l'offset (codé en complément à deux, ce qui permet de sauter en arrière (offset négatif) ou en avant (offset positif). L'opcode de l'instruction est 10, situés sur les bits de poids fort (B7 et B6 respectivement). L'opérande est placé en bits B0 à B5. Attention : l'exécution d'un `Jump` ne provoque pas de décalage à gauche sur l'afficheur.

`Clear` C'est l'instruction la plus simple : son opcode est 11 (situé sur les deux bits de poids fort). L'instruction `Clear` efface immédiatement le contenu de l'afficheur défilant. Cette instruction ne comporte pas d'opérande.

`Nop` L'instruction `Nop` (ou No-Op pour *No Operation*) est une instruction qui consiste à ne rien faire. C'est (aussi étonnant que cela puisse paraître) une instruction très courante sur les processeurs (en particulier les processeurs RISC). Elle permet de forcer le processeur à attendre pour assurer la

1. Vérifiez que vous comprenez parfaitement ce point ; sinon, demandez à un enseignant.

synchronisation entre différentes tâches. Cette instruction est souvent une “pseudo-instruction”, c’est-à-dire qu’elle est implémentée par le biais d’une autre instruction en lui donnant des opérandes qui en font une instruction nulle. Ce sera le cas dans notre machine : `Nop` sera assurée par un `Jump +1`, ce qui force la machine à prendre le temps d’exécuter une instruction mais qui passe malgré tout à l’instruction suivante dans la mémoire. Dans notre cas, l’instruction `Nop` pourra être utile pour ralentir le défilement de l’afficheur.

Le tableau suivant résume la fonction et le codage des quatre instructions de notre machine :

instruction	fonction	opcode	longueur de l’opérande
`Show`	Affiche une colonne	`0`	7 bits
`Jump`	Saut relatif	`10`	6 bits
`Clear`	Efface l’afficheur	`11`	pas d’opérande
`Nop`	Instruction nulle (No-Op)	`10`	6 bits (opérande fixe : `000001`)

QUESTION 1 ► Vérifiez que vous avez bien compris le fonctionnement des quatre instructions (tout particulièrement de l’instruction `Jump` et de la pseudo-instruction `Nop`) avant de passer à la suite. Si vous n’êtes pas sûrs de vous, n’hésitez pas à demander !

A l’aide de ces quatre instructions, vous allez pouvoir corriger le défaut du circuit précédent. En effet, il vous suffira de rajouter dans la mémoire une ou plusieurs instructions `Jump` pour réaliser une boucle infinie permettant d’afficher en boucle (justement !) un message de la longueur de votre choix. Pour cela votre “programme” prendra typiquement la forme suivante :

adresse	opcode	opérande	hexadécimal	instruction
0	0	1111101	0x7D	`Show 1111101`
1	10	000001	0x81	`Nop`
2	0	0000000	0x00	`Show 0000000`
3	10	000001	0x81	`Nop`
4	0	0000000	0x00	`Show 0000000`
5	10	000001	0x81	`Nop`
6	0	0000000	0x00	`Show 0000000`
7	10	111001	0xB9	`Jump -7`

QUESTION 2 ► Selon vous, quel motif sera affiché par ce programme ?

2.1 Conception

Nous allons implémenter notre machine de von Neumann sous la forme d’une “Algorithmic State Machine” (ASM). Notre circuit comprendra donc quatre éléments :

Un afficheur défilant C’est l’afficheur que vous utilisez depuis de début de ce TP

Une mémoire La même qu’au chapitre précédent. Cependant, alors qu’au chapitre précédent, la mémoire contenait directement les mots à envoyer à l’afficheur, elle contiendra ici les instructions qui seront interprétées par notre machine de von Neumann. C’est cette dernière qui communiquera avec l’afficheur.

Une Unité de Contrôle (UC) en charge de contrôler les échanges entre tous les éléments

Un DataPath qui réalisera les calculs (certes extrêmement simples ici) correspondant aux instructions reçues par la machine.

2.1.1 Conception du DataPath

QUESTION 3 ► Dans une machine de von Neumann, le DataPath doit toujours contenir au moins deux registres. Lesquels ? Vérifiez que vous comprenez parfaitement leurs rôles respectifs.

QUESTION 4 ► Connaissant le jeu d’instruction de votre machine, quelles sont les opérations que devra pouvoir réaliser votre DataPath sur ces deux registres ? Dessinez (sur papier pour le moment) le schéma de votre DataPath.

QUESTION 5 ► Votre DataPath sera contrôlé par votre unité de contrôle par le biais de signaux de commande (“Commands”). Listez ces signaux.

QUESTION 6 ► Votre DataPath produira deux types de signaux : des “Reports” et des “Data-Out”. Listez les signaux correspondants. Si vous n’arrivez pas à répondre à la question, rappelez-vous que les “Reports” sont à destination de l’UC tandis que les “Data-Out” sont à destination du “monde extérieur”. Quels sont, ici, les éléments composant ce “monde extérieur” ?

QUESTION 7 ► Votre DataPath recevra des informations en provenance du “monde extérieur". Lesquelles ?

2.1.2 Conception de l’UC

QUESTION 8 ► En plus des signaux de commande, votre unité de contrôle produira aussi des “Acknowledges” à destination du “monde extérieur”. Quels sont-ils ?

QUESTION 9 ► Votre UC sera conçue sous la forme d’une machine à états finis (FSM). À partir des réponses aux questions précédentes, donnez l’alphabet d’entrée et l’alphabet de sortie de cette FSM.

QUESTION 10 ► Si vous allez faire un petit tour sur la page du cours AC pour réviser le principe du “cycle de von Neumann”, vous constaterez que l’implémentation du cycle repose sur un automate comportant trois phases (un état `Fetch`, un état `Decode` et un ensemble plus ou moins complexe d’états `Execute`). Avant de passer à la question suivante, vérifiez que vous avez bien compris à quoi servent chacun de ces états.

QUESTION 11 ► Connaissant le jeu d’instructions de votre machine et le principe du cycle de von Neumann, combien d’états aura votre automate ? Lesquels ? Dessinez l’automate correspondant à votre unité de contrôle en veillant à bien faire figurer les signaux d’entrée et de sortie de votre automate pour chaque transition et état respectivement[2].

2.2 Implémentation

QUESTION 12 ► Implémentez votre DataPath (pour simplifier le déboggage éventuel, nous vous conseillons d’implémenter directement votre DataPath et votre UC sur le circuit de l’afficheur). Nous vous conseillons d’utiliser les registres 8 bits réalisés lors du premier TP.

QUESTION 13 ► Implémentez votre UC. Comme pour les automates réalisés dans les TPs précédents, nous vous conseillons d’implémenter le registre d’état au moyen du composant `D-Flip-Flop, asynchronous`.

QUESTION 14 ► Ajoutez une ROM et initialisez-là, par exemple avec le petit programme présenté en exemple ci-dessus. Dans un premier temps vous pouvez aussi réutiliser la ROM que vous avez utilisée dans le chapitre précédent puisque les bits de poids forts sont à zéro (ce qui correspond donc à des instructions `Show`). Connectez le tout, lancez ... corrigez ... lancez ... corrigez[3].

2.3 Programmation

QUESTION 15 ► Même si elle est très frustre, votre machine de von Neumann est bien une machine programmable. Pour vérifier cette affirmation, codez cinq programmes réalisant les affichages suivants :

- Un programme affichant en boucle `HELLO WORLD` (vous pouvez évidemment repartir de la mémoire utilisée au chapitre précédent),
- Un programme affichant `HELLO WORLD` et éteignant ensuite immédiatement l’afficheur (sans laisser le message défiler) avant de recommencer,
- Un programme affichant une et une seule fois `HELLO WORLD`.
- Un programme affichant `HELLO WORLD` une fois puis affichant en boucle le mot `WORLD`,

2. Si vous ne comprenez pas pourquoi “respectivement”, il est urgent de réviser le cours et/ou de demander à un enseignant...

3. Pour débogger votre machine, il peut être utile de rajouter des afficheurs sur quelques bus stratégiques de façon à suivre le déroulement du cycle de von Neumann et du programme (typiquement l’état courant, l’état suivant et le bus d’adresse). Pour cela vous pouvez utiliser des sondes (`Probe`) ou, mieux, des afficheurs hexadécimaux (menu `Components`→`IO`→`more`→`Seven-Segment-Hex Display`).

— Si, dans votre premier programme, vous avez utilisé des sauts-relais pour revenir au début du programme, vous aurez remarqué que cela provoque des ralentissements intempestifs dans le déroulement du programme. Codez un programme qui résout ce problème en utilisant l'instruction `Nop`.

Quelques conseils pour réaliser ces programmes :

Utilisation de l'instruction `Jump` : La taille des sauts est limitée puisqu'ils sont codés en complément à deux sur 6 bits. En conséquence, si on veut afficher un message de plus de 32 colonnes on ne peut plus revenir au début avec un seul saut (si vous ne comprenez pas d'où vient ce 32, il est temps de réviser un peu le complément à deux...). Deux solutions : soit faire des sauts "vers l'avant" (offset positif), dépasser la limite de la mémoire et revenir au début(là encore, si vous ne comprenez pas ce point, il est temps de revoir le codage des entiers), soit effectuer plusieurs sauts successifs "vers l'arrière" (des "sauts-relais" à offset négatif) pour remonter dans la mémoire. Dans ce cas, il ne vous échappera pas que, lors de l'affichage, la machine va nécessairement passer sur le saut-relais ce qui risque d'interrompre la séquence d'affichage. On évite ce problème en insérant, en amont du saut-relais, un saut vers l'avant qui "passe par dessus" le saut relais[4].

Idle Idle n'est pas une instruction mais plutôt un état du programme (attention : ce n'est pas un état du processeur ou de l'automate !). Vous aurez compris qu'il est impossible d'arrêter une machine de von Neumann puisqu'à la fin d'une instruction elle passera toujours à l'instruction suivante. La seule façon de faire en sorte qu'elle ne fasse "plus rien" est de faire tourner le programme "dans le vide" en le forçant à ne rien faire (*to run idle* : tourner à vide). A vous de trouver comment ...

Utilisation de l'instruction `Nop` : Si vous insérez des sauts dans un programme, ceux-ci vont provoquer des arrêts de l'afficheur le temps de faire le saut. Ce n'est pas très grave mais c'est disgracieux. La solution consiste à retarder systématiquement toutes les instruction `Show` en les faisant suivre d'une ou plusieurs instructions `Nop` (sauf, bien sûr, celles qui sont déjà suivies d'un `Jump` "utile").

Si ces tous vos programmes fonctionnent, alors Bravo ! ! ! Vous êtes arrivés à la fin des TPs d'AC ...

3 Bonus

Si vous en voulez plus, un dernier bonus ...

QUESTION 16 ► Modifiez votre machine pour ajouter une cinquième instruction : `Set`. Cette instruction, sans opérande, permettra d'allumer simultanément toutes les leds de l'afficheur (et donc de faire clignoter l'afficheur – en alternant les instructions `Set` et `Clear` – ou de faire défiler du texte en noir sur fond "blanc" au lieu de "blanc" sur fond noir). Un indice pour vous guider, il vous faudra commencer par modifier votre registre 8 bits ...

4. Un petit exemple pour bien se faire comprendre : le programme `Show(A) Show(B) Show(C) Jump(-3) Show(D) Show(E) Show(F) Jump(-4)` affiche la séquence `ABCABCABC...` et ignore les quatre dernières instructions en raison du saut-relais (le `Jump(-3)`). Pour éviter ce problème il suffit d'écrire `Show(A) Show(B) Show(C) Jump(+2) Jump(-4) Show(D) Show(E) Show(F) Jump(-4)` qui, lui, affiche bien la séquence `ABCDEFABCDEFAB...` Si vous n'avez rien compris à ce qui précède, n'hésitez pas à demander à un enseignant...