

---

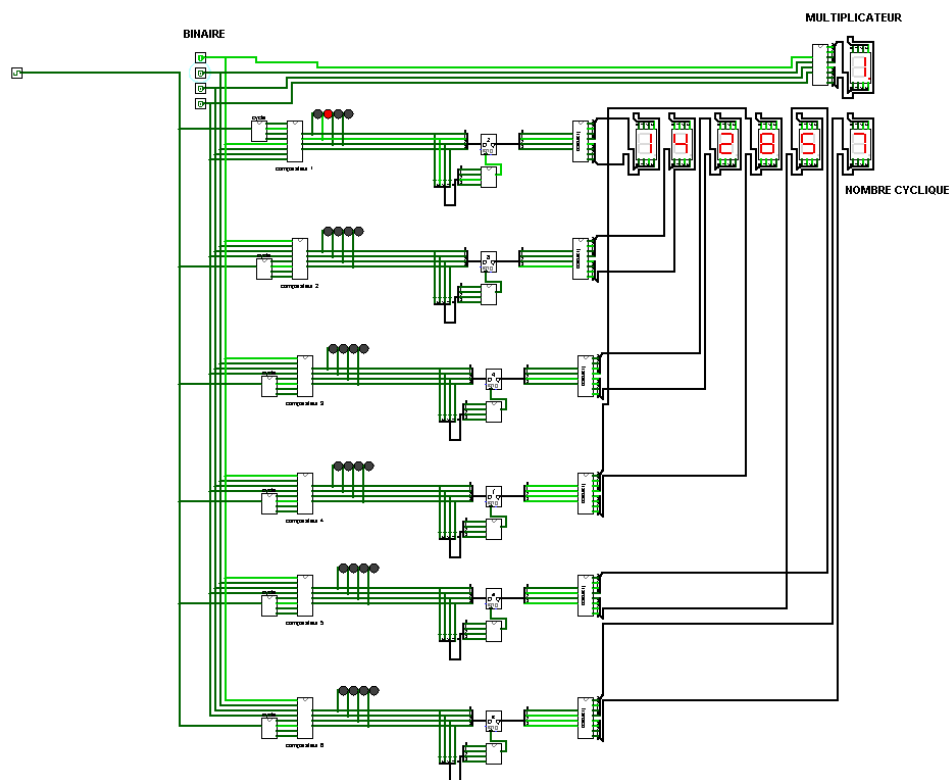
# Projet Electronique Mixte

Une approche magique : les nombres cycliques

---

Oggy

oggyfoxy@gmail.com



Année Académique 2023-2024  
21 Juin 2024

## Résumé

Ce projet vise à concevoir des transcodeurs capables de convertir les signaux binaires en affichages pour des écrans à 7 segments, mettant l'accent sur le séquençage du nombre cyclique "142857" et ses multiples. Après une série de simulations et de tests rigoureux dans Logisim, nous avons validé la précision de l'affichage de ces chiffres en logique combinatoire et séquentielle.

# Table des matières

<b>Liste des Figures</b>	<b>2</b>
<b>1 Introduction</b>	<b>3</b>
1.1 Contexte et Objectifs du Projet . . . . .	3
1.2 Portée et Démarche . . . . .	4
<b>2 Théorie et Principes de Base</b>	<b>6</b>
2.1 Les Afficheurs 7 Segments et leur Pilotage . . . . .	6
2.2 Le Code Binaire Pondéré . . . . .	7
<b>3 Simuler le tour sous Logisim en Logique Combinatoire</b>	<b>10</b>
<b>4 Simuler le tour sous Logisim en Logique Séquentielle</b>	<b>17</b>
<b>5 Conclusion</b>	<b>29</b>

# Table des figures

1.1	Le nombre cyclique 142 857 multiplié par les nombres de 1 à 6. . . . .	3
2.1	Afficheur 7 segments . . . . .	6
2.2	Code binaire pondéré de 0000 à 1111 . . . . .	7
2.3	Code binaire pondéré : cyclique . . . . .	8
2.4	Code binaire pondéré : multiplicateur . . . . .	8
2.5	Code binaire pondéré : le "0" . . . . .	8
3.1	Les circuits composants la partie combinatoire . . . . .	10
3.2	test_transco_hexa_aff . . . . .	11
3.3	Le code "0110" de mon code binaire pondéré affiche E. . . . .	12
3.4	Le circuit affiche_cyclique7seg . . . . .	13
3.5	affiche_cyclique_1 et sa table de transition . . . . .	14
3.6	Le circuit aff_res_mult . . . . .	16
3.7	aff_1_res_mult (table) . . . . .	16
4.1	Liste des circuits en logique séquentielle . . . . .	17
4.2	Les circuits cycle_complet_nb_x, cycle_nb_x et cycle_test . . . . .	18
4.3	Le circuit cycle_complet_nb_1 . . . . .	18
4.4	Architecture du flip-flop JK . . . . .	19
4.5	Table de vérité du cycle_nb_1 . . . . .	20
4.6	Les règles de la construction du tableau de transition JK . . . . .	20
4.7	Graphe des états cycliques . . . . .	21
4.8	Le cycle_test . . . . .	22
4.9	Les circuits cycle, finalcycle, multicycle et finalmulti . . . . .	23
4.10	En exemple le circuit multi_cycle . . . . .	23
4.11	Circuit main . . . . .	24
4.12	Les derniers circuits constituant le circuit final . . . . .	24
4.13	comparateur_1 . . . . .	25
4.14	Ligne du comparateur pour 0001 . . . . .	26
4.15	Le cycle et ses multiples . . . . .	26
4.16	Câblage du cycle et du comparateur_1 dans main . . . . .	26
4.17	Cycle lorsque le multiplicateur est 1. . . . .	27
4.18	Le register et le tickblocker dans main . . . . .	27
4.19	Le register dans Logisim . . . . .	28
4.20	Table de vérité du tickblocker . . . . .	28

# Chapitre 1

## Introduction

### 1.1 Contexte et Objectifs du Projet

Un nombre cyclique, ou nombre phénix, est un entier naturel dont les permutations circulaires des chiffres correspondent aux multiples du nombre. Le plus connu est 142 857 :

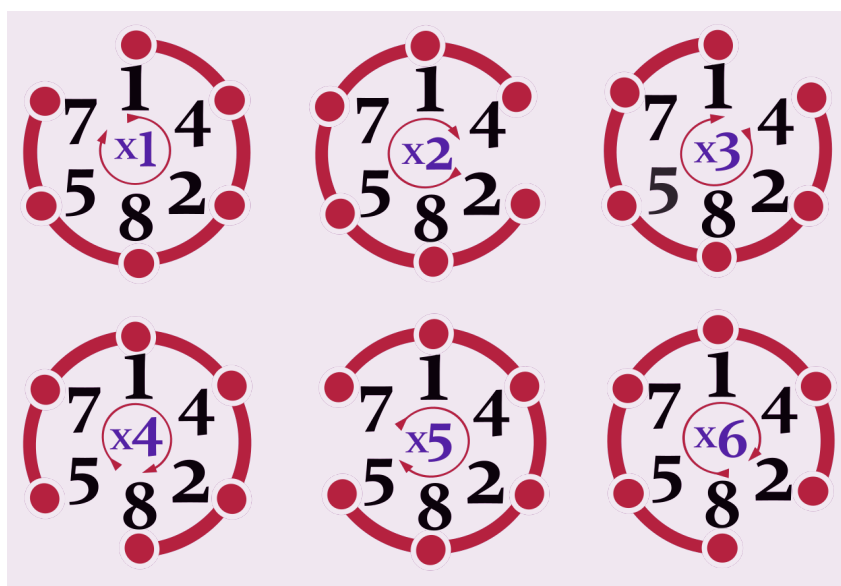


FIGURE 1.1 – Le nombre cyclique 142 857 multiplié par les nombres de 1 à 6.

Ce nombre "magique" est la période du développement décimal de la division  $\frac{1}{7}$  (Notons ici que nous travaillons en base 10). Sur la figure 1.1, on peut voir ses multiples successifs qui permettent de faire la permutation circulaire de 142 857 :

$$\begin{aligned}1 \times 142\,857 &= 142\,857 \\2 \times 142\,857 &= 285\,714 \\3 \times 142\,857 &= 428\,571 \\4 \times 142\,857 &= 571\,428 \\5 \times 142\,857 &= 714\,285 \\6 \times 142\,857 &= 857\,142\end{aligned}$$

Le but principal de ce projet est de simuler ce nombre cyclique sur Logisim, un logiciel nous permettant de simuler des circuits logiques électroniques. Nous allons ainsi réaliser ceci de deux manières distinctes : en logique combinatoire et en logique séquentielle. Le but final est pouvoir de visualiser ce nombre sur les afficheurs 7-segments grâce aux deux méthodes proposées.

Nous distinguerons ainsi le multiplicateur et les nombres cycliques. Des explications en détail seront transmises en deuxième partie.

## 1.2 Portée et Démarche

Pour ce projet, nous simulons le cycle en deux parties :

### Logique combinatoire

La logique combinatoire fait référence aux circuits dans lesquels l'état des sorties dépend uniquement de la combinaison actuelle des états d'entrée. Autrement dit, il n'y a pas de mémoire des états d'entrée précédents ou de notion de séquence de temps dans le comportement du circuit. Par exemple, portes logiques comme AND, OR, NOT, NAND, NOR et XOR sont des exemples de composants de base utilisés dans les circuits combinatoires. Pour notre projet, la création de ces portes logiques sont décrites par des tables de vérité qui calculent les valeurs de sortie pour toutes les combinaisons possibles des valeurs d'entrée. Cela permet ainsi que les circuits puissent répondre instantanément aux changements d'entrées.

#### Exemples de logique combinatoire :

- Un circuit qui calcule la somme de deux bits (demi-additionneur).
- Un multiplexeur qui sélectionne une ligne d'entrée pour la transmettre à la sortie.
- Un circuit qui compare deux nombres pour déterminer s'ils sont égaux.

### Logique séquentielle

La logique séquentielle, en revanche, intègre la notion de temps et de séquence. Les circuits séquentiels ont de la mémoire, ce qui signifie que l'état actuel des sorties dépend non seulement de l'état actuel des entrées mais aussi de l'historique des entrées. En somme les sorties dépendent des entrées passées du circuit. Nous pouvons mettre ça en œuvre à l'aide de composants qui permettent de stocker l'information tels que les bascules (flip-flops) comme la bascule JK et les registres, qui peuvent maintenir un état jusqu'à ce qu'ils reçoivent un signal pour changer (par exemple, un signal d'horloge). La logique séquentielle sera essentielle pour nous permettre d'afficher le nombre cyclique de manière asynchrone et aussi grâce à des outils comme des compteurs, des registres à décalage, et des machines d'état.

#### Exemples de logique séquentielle :

- Un compteur qui incrémente sa valeur à chaque coup de clock.
- Un registre à décalage qui déplace les bits d'entrée à travers une série de flip-flops à chaque cycle d'horloge

- Une machine d'état fini qui change d'état en fonction de son état actuel et des entrées.

La principale différence entre les deux est que les circuits combinatoires n'ont pas de mémoire et que leur sortie est directement le produit des entrées actuelles, tandis que les circuits séquentiels ont de la mémoire et que leur sortie est une fonction des entrées actuelles et des états passés.

# Chapitre 2

## Théorie et Principes de Base

### 2.1 Les Afficheurs 7 Segments et leur Pilotage

L'afficheur à sept segments est un dispositif utilisé pour afficher des chiffres ou des lettres. Il est composé, comme son nom l'indique, de sept segments disposés de manière à former un motif qui peut représenter les chiffres de 0 à 9 et parfois quelques lettres comme A, B, C, etc.

Chaque segment est désigné par une lettre ou un chiffre, généralement de "0" à "6". Ces segments sont allumés ou éteints sélectivement pour former différents caractères.

Voici un schéma simplifié d'un afficheur à sept segments :

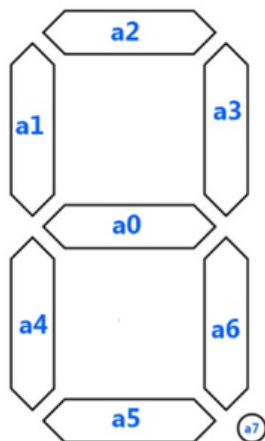


FIGURE 2.1 – Afficheur 7 segments

Pour notre projet, nous utiliserons plutôt un afficheur 8 segments puisqu'on se servira également du point situé en bas à gauche (noté a7 sur le schéma) pour différencier nos chiffres multiplicateur et cyclique.



## 2.2 Le Code Binaire Pondéré

Pour ce projet nous avons tous un code binaire pondéré différent. Pour ma part, je possédais le code binaire pondéré **1124** et je possédais le choix numéro 2.

### Choix 2

(a) : Le chiffre composant le nombre cyclique aura le plus de bits nul tandis que le multiplicateur en aura le moins.

(b) Le chiffre composant le nombre cyclique aura son premier bit non nul le plus proche du bit de poids faible.

En suivant ces règles, j'ai pu créer mon propre code binaire pondéré.

b3	b2	b1	b0	chiffre
0	0	0	0	0
0	0	0	1	4
0	0	1	0	2
0	0	1	1	6
0	1	0	0	1
0	1	0	1	5
0	1	1	0	3
0	1	1	1	7
1	0	0	0	1
1	0	0	1	5
1	0	1	0	3
1	0	1	1	7
1	1	0	0	2
1	1	0	1	6
1	1	1	0	4
1	1	1	1	8

FIGURE 2.2 – Code binaire pondéré de 0000 à 1111

Nous distinguerons ici les chiffres correspondant au cycle et ceux qui correspondront au multiplicateur.

0	0	0	1	4
0	0	1	0	2
0	0	1	1	6
0	1	0	0	1
0	1	0	1	5
0	1	1	0	3
0	1	1	1	7
1	1	1	1	8

FIGURE 2.3 – Code binaire pondéré : cyclique

1	0	0	0	1
1	0	0	1	5
1	0	1	0	3
1	0	1	1	7
1	1	0	0	2
1	1	0	1	6
1	1	1	0	4

FIGURE 2.4 – Code binaire pondéré : multiplicateur

0	0	0	0	0
---	---	---	---	---

FIGURE 2.5 – Code binaire pondéré : le "0"

(En figure 2.5, on a le 0 qui affiche rien car on le considère comme la valeur éteinte)

Nota bene : le "3" et le "6" du nombre cyclique ainsi que le "7" du multiple ne s'afficheront bien évidemment jamais car ils n'existent pas dans le cycle. Un "E" s'affiche donc, ils ont été créés pour faciliter le tableau.

Le code binaire pondéré est un système de codification où chaque bit d'un nombre a une valeur de poids attribuée, basée sur sa position. Typiquement, dans un système binaire classique, les poids sont des puissances de deux. Par exemple, dans un système de quatre bits les poids de la gauche vers la droite sont 8, 4, 2, et 1. La représentation pondérée nous permet de faciliter la conversion entre les nombres binaires et décimaux, et est la base sur laquelle Logisim (ou même les ordinateurs en général) fait ses calculs.

En appliquant ce concept à notre projet nous permet de jouer avec des valeurs binaires pondérées pour représenter des nombres de manière que les circuits logiques puissent les

interpréter et les manipuler efficacement. Ils jouent un rôle crucial dans l'interprétation des entrées et des sorties pour nos transcodeurs et le reste de notre circuit.

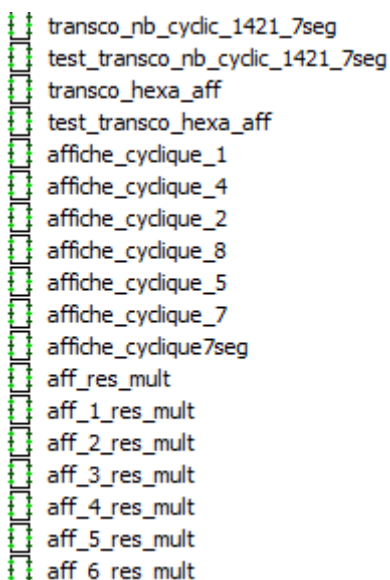
Nous implémenterons ces codes binaires dans les tables de vérités tout au long du projet.

## Chapitre 3

# Simuler le tour sous Logisim en Logique Combinatoire

Voyons ensemble maintenant les circuits qui constitueront la partie combinatoire du projet.

Ici nous avons plusieurs circuits :



```
transco_nb_cyclic_1421_7seg
test_transco_nb_cyclic_1421_7seg
transco_hexa_aff
test_transco_hexa_aff
affiche_cyclique_1
affiche_cyclique_4
affiche_cyclique_2
affiche_cyclique_8
affiche_cyclique_5
affiche_cyclique_7
affiche_cyclique7seg
aff_res_mult
aff_1_res_mult
aff_2_res_mult
aff_3_res_mult
aff_4_res_mult
aff_5_res_mult
aff_6_res_mult
```

FIGURE 3.1 – Les circuits composants la partie combinatoire

Regardons ensemble les deux premiers circuits : les circuits **transco\_hexa\_aff** et **test\_transco\_hexa\_aff**.

Le circuit **transco\_hexa\_aff** contient la table de vérité en binaire qui permet d'afficher les chiffres de "0 à 9" et les lettres de "A à F" sur l'afficheur 7-segments.

Nous trouverons cet afficheur sur le circuit de test : **test\_transco\_hexa\_aff**.

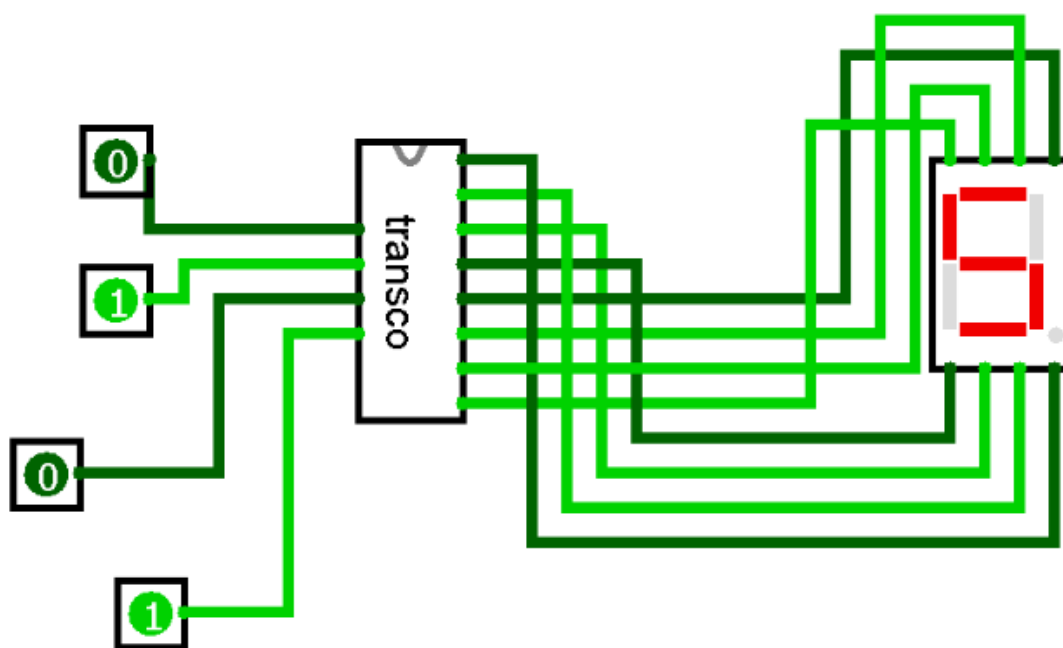


FIGURE 3.2 – test\_transco\_hexa\_aff

Exemple : le code binaire pondéré **0101** affiche **5** sur l'afficheur 7-segments.

L'élément clé ici est le transcodeur qui va nous permettre d'afficher la sortie que l'on souhaite suivant notre code binaire pondéré tout au long du projet.

Un transcodeur dans ce contexte est un dispositif de circuit qui convertit les codes binaires d'entrée en un ensemble de signaux utilisés pour contrôler l'affichage d'un afficheur à 7 segments. Il sert d'intermédiaire pour traduire les nombres binaires en représentation visuelle correspondante sur l'afficheur.

Ce circuit nous a permis de comprendre comment le programme de base fonctionne et comment utiliser les tables de vérités et les afficheurs 7-segments.

Maintenant nous allons voir les deux circuits suivants :  
les circuits **transco\_nb\_cyclic\_1421\_7seg** et **test\_transco\_nb\_cyclic\_1421\_7seg**.

De manière analogue aux deux derniers circuits, ces circuits vont nous permettre d'afficher tous les chiffres du nombre cyclique 142857 et les multiplicateurs de 1 à 6.

Pour les chiffres qui appartiennent à aucun de ces deux groupes, la lettre "E" s'affiche pour marquer qu'il y a une erreur.

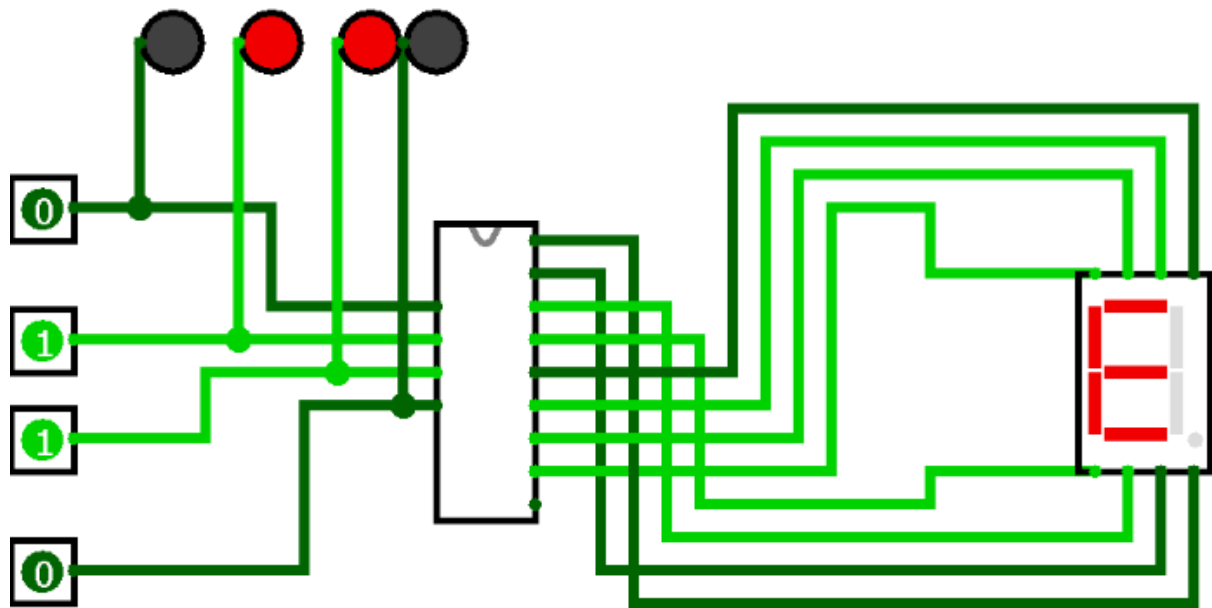


FIGURE 3.3 – Le code "0110" de mon code binaire pondéré affiche E.

C'est ce transcodeur que l'on va utiliser tout au long du projet pour afficher nos nombres cycliques.

Les 6 circuits **affiche\_cyclique\_x** sont des éléments constructeurs pour l'utilisation du circuit **affiche\_cyclique\_7seg**.

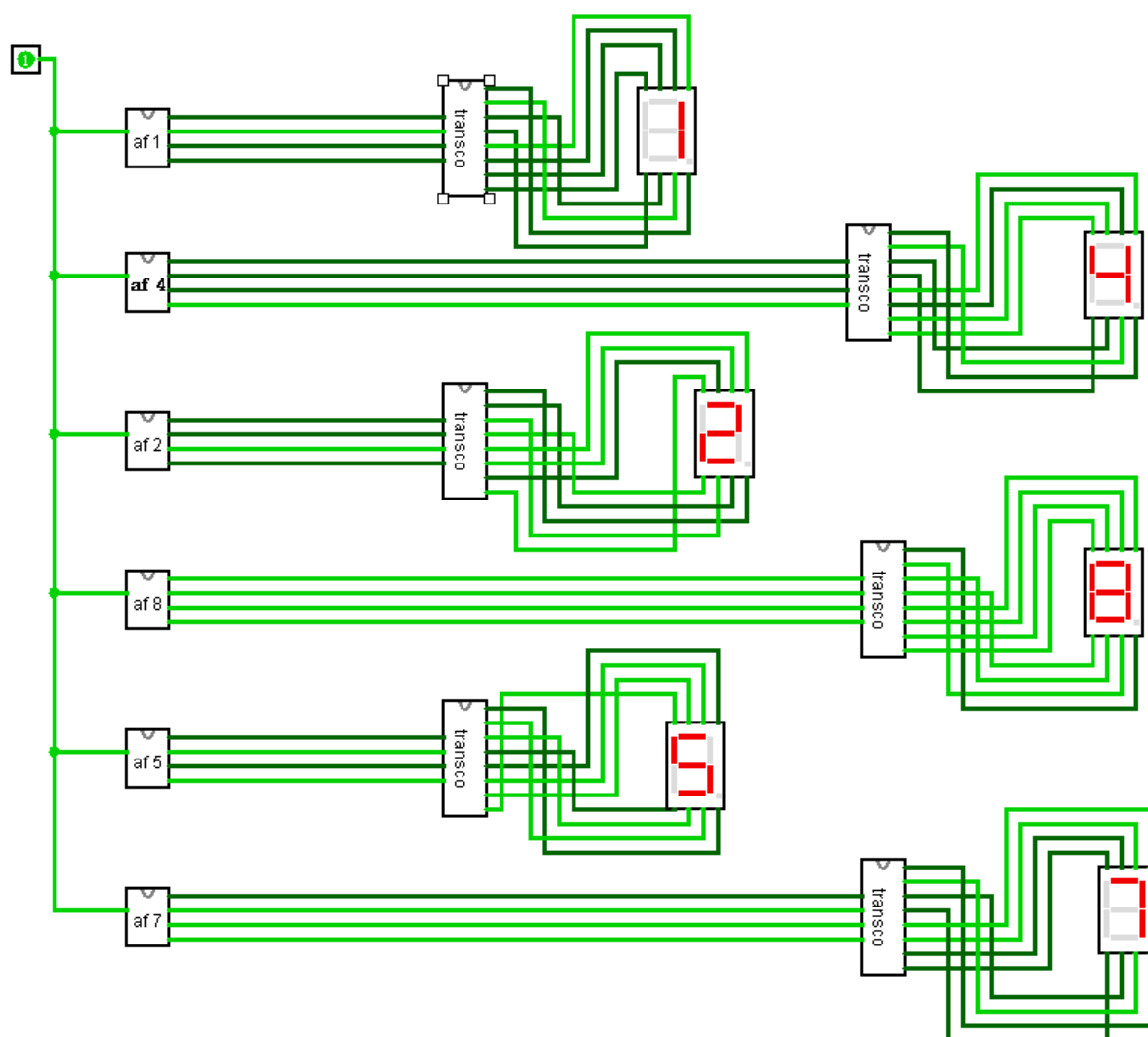


FIGURE 3.4 – Le circuit affiche\_cyclique7seg

En appuyant sur l'entrée, (le petit carré avec un 0 en haut), vous allez voir pour chaque afficheur le cycle de départ 142857.

Dans chacune des petites boîtes se trouve une table de vérité qui permet d'afficher spécifiquement les chiffres du cycle initial 142857.

Par exemple dans affiche 1 :

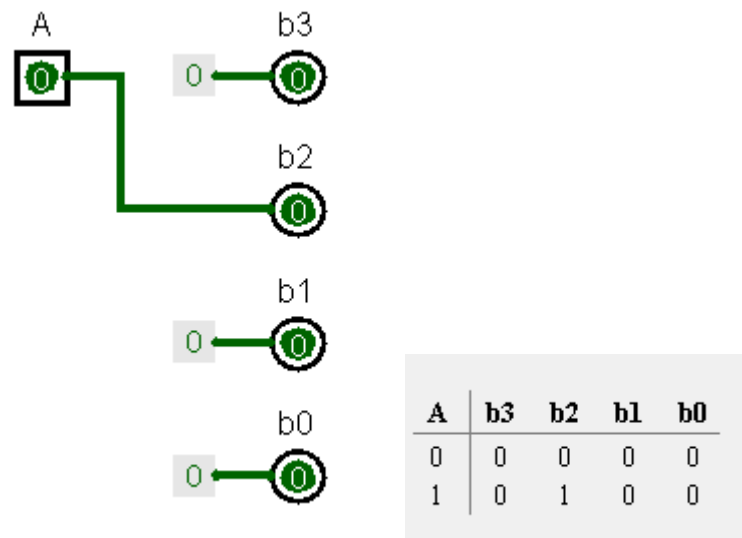


FIGURE 3.5 – affiche\_cyclique\_1 et sa table de transition

Cela affiche bien 1 car cela correspond bien à 1 pour mon code binaire pondéré : **0100**

Ce sont simplement des circuits de tests pour comprendre la logique combinatoire du projet, et pouvoir ainsi construire le projet petit à petit.

Dans le circuit précédent, nous pouvions seulement voir le cycle de départ 142857. Les 6 circuits suivants nous serviront pour pouvoir afficher les 6 cycles selon leur multiples.



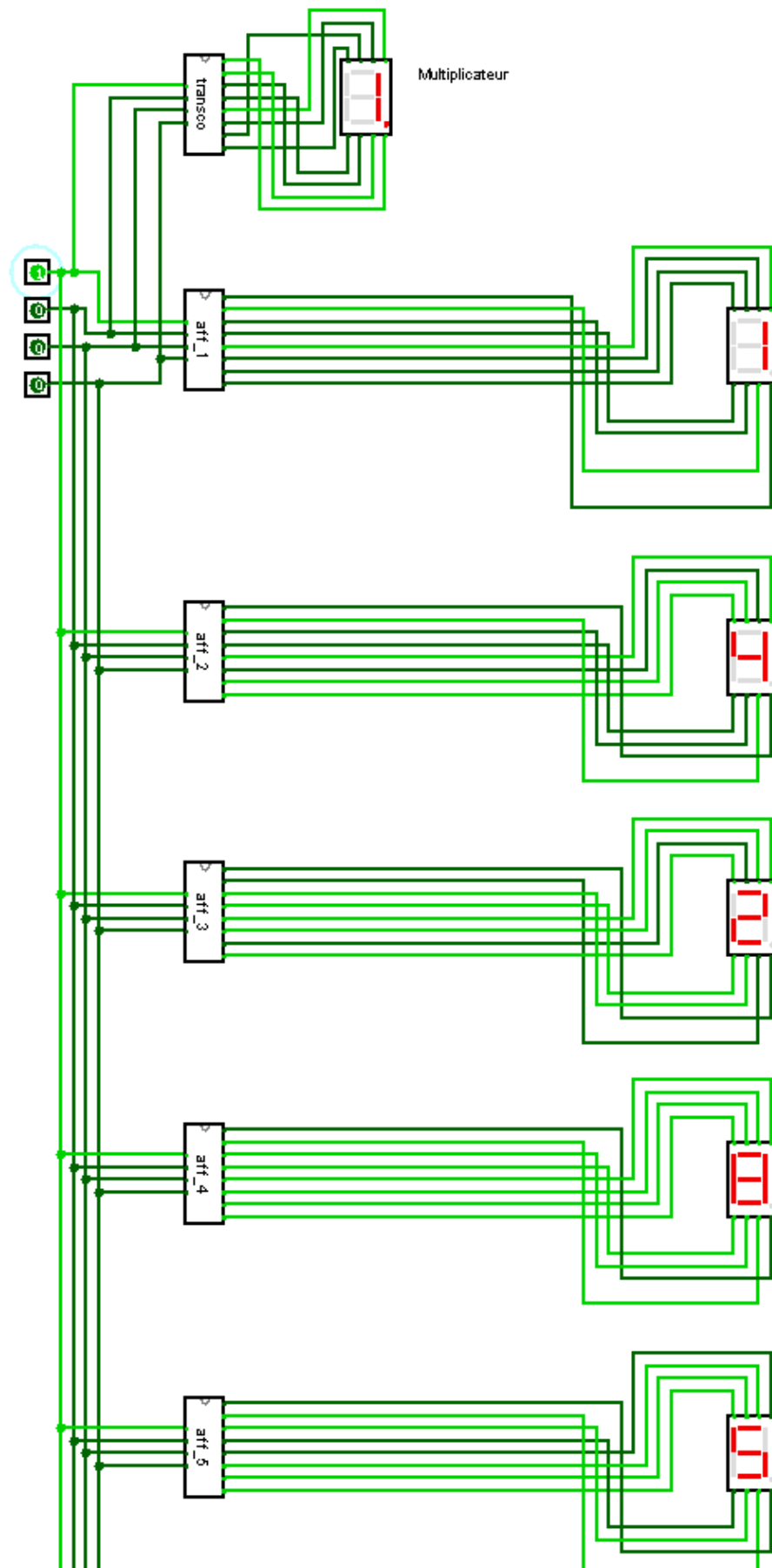


FIGURE 3.6 – Le circuit `aff_res_mult`

Dans ce circuit là, nous avons simulé le circuit précédent mais en entrant toutes les valeurs du cycle dans la table de vérité pour chacun des afficheurs. Le but ici est d'avoir cette séquence cyclique pour chaque afficheur et d'afficher dans l'ordre les valeurs correspondantes du cycle.

Nous voyons aussi en haut un transcodeur avec un afficheur libellé "Multiplicateur", qui permet de nous situer visuellement où nous sommes dans le cycle.

En entrée nous avons plus simplement qu'à entrer notre code binaire constituant l'un de nos 6 multiplicateurs. (ici : 1000 correspond à mon 1. multiplicateur)

A	B	C	D	a7	a6	a5	a4	a3	a2	a1	a0
0	0	0	0	0	0	1	1	0	1	1	1
0	0	0	1	0	0	1	1	0	1	1	1
0	0	1	0	0	0	1	1	0	1	1	1
0	0	1	1	0	0	1	1	0	1	1	1
0	1	0	0	0	0	1	1	0	1	1	1
0	1	0	1	0	0	1	1	0	1	1	1
0	1	1	0	0	0	1	1	0	1	1	1
0	1	1	1	0	0	1	1	0	1	1	1
1	0	0	0	0	1	0	0	1	0	0	0
1	0	0	1	0	1	0	0	1	1	0	0
1	0	1	0	0	1	0	0	1	0	1	1
1	0	1	1	0	0	1	1	0	1	1	1
1	1	0	0	0	0	1	1	1	1	0	1
1	1	0	1	0	1	1	1	1	1	1	1
1	1	1	0	0	1	1	0	0	1	1	1
1	1	1	1	0	0	1	1	0	1	1	1

FIGURE 3.7 – `aff_1_res_mult` (table)

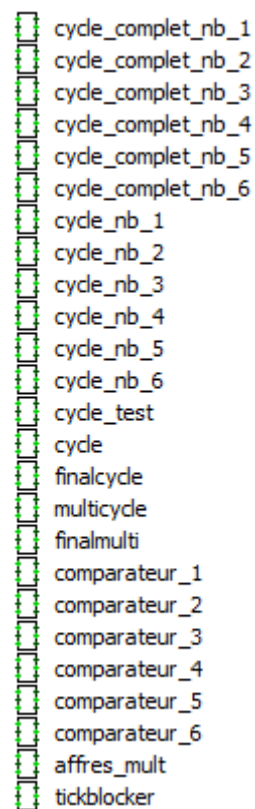
Prenons pour exemple la table de vérité pour `aff_1_res_mult`. Nous voyons ici une séquence qui se répète pour un grand nombre de codes binaires : 001100111. Cette séquence permet d'afficher une erreur (E sur le transcodeur) pour nous indiquer que la valeur n'appartient pas au cycle.

Ainsi, nous avons notre cycle complet pour la partie combinatoire.

Le but maintenant serait de voir ce cycle en logique séquentielle, qui est intimement lié à un résultat automatisé.

# Chapitre 4

## Simuler le tour sous Logisim en Logique Séquentielle



cycle\_complet\_nb\_1  
cycle\_complet\_nb\_2  
cycle\_complet\_nb\_3  
cycle\_complet\_nb\_4  
cycle\_complet\_nb\_5  
cycle\_complet\_nb\_6  
cycle\_nb\_1  
cycle\_nb\_2  
cycle\_nb\_3  
cycle\_nb\_4  
cycle\_nb\_5  
cycle\_nb\_6  
cycle\_test  
cycle  
finalcycle  
multicycle  
finalmulti  
compareur\_1  
compareur\_2  
compareur\_3  
compareur\_4  
compareur\_5  
compareur\_6  
affres\_mult  
tickblocker

FIGURE 4.1 – Liste des circuits en logique séquentielle

Voici l'ensemble des circuits qui constituent la partie séquentielle du projet, ainsi que le circuit final main que nous verrons juste après.

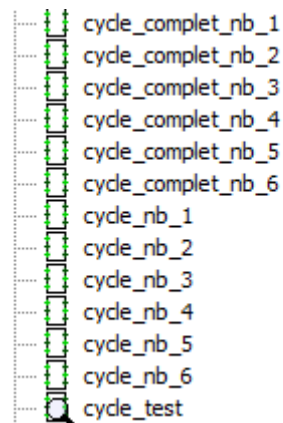


FIGURE 4.2 – Les circuits cycle\_complet\_nb\_x, cycle\_nb\_x et cycle\_test

Commençons par voir l'ensemble de ces circuits (les circuits cycle\_complet\_nb\_x, cycle\_nb\_x et cycle\_test) et comprendre comment chacun vont permettre de tester le comportement séquentiel de notre cycle, afin d'implémenter une logique similaire au circuit final.

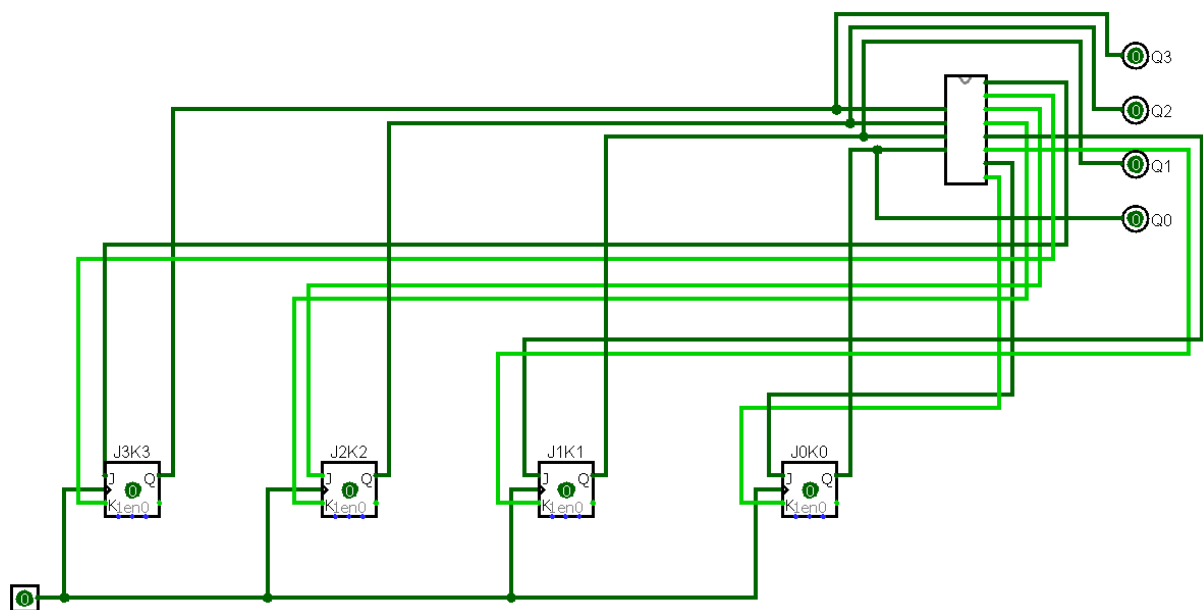


FIGURE 4.3 – Le circuit cycle\_complet\_nb\_1

Ce circuit introduit un nouvel élément : les flip-flops JK.

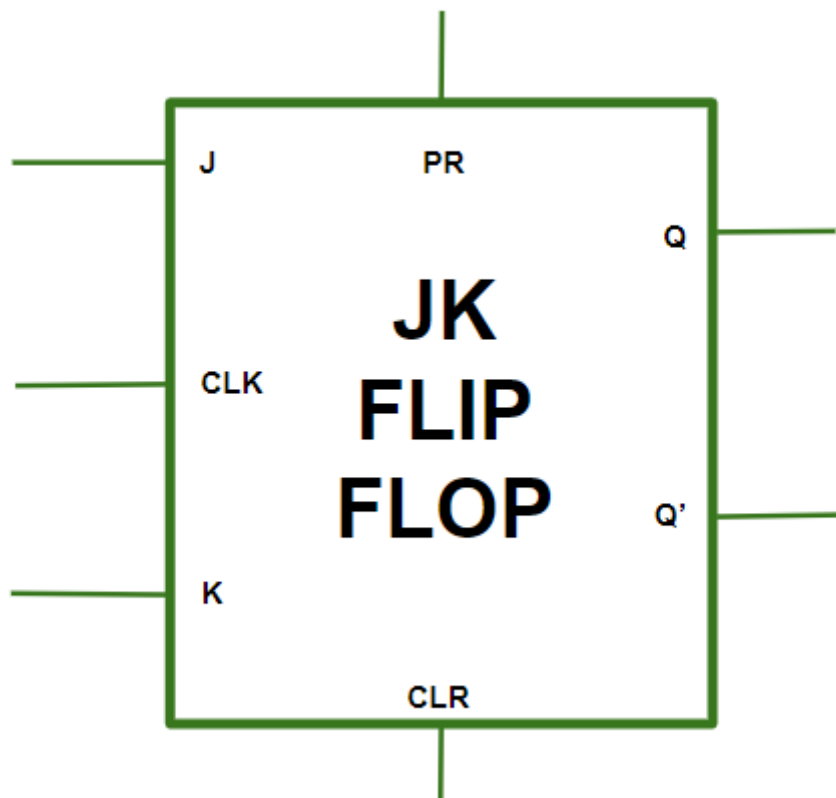


FIGURE 4.4 – Architecture du flip-flop JK

Expliquons brièvement son comportement :

- J : C'est l'entrée qui, lorsqu'elle est activée, configure la sortie Q à 1 au prochain signal d'horloge.
- K : C'est l'entrée qui, lorsqu'elle est activée, configure la sortie Q à 0 au prochain signal d'horloge.
- CLK (Horloge) : Elle synchronise le flip-flop, déclenchant les changements d'état sur ses sorties.
- PR (Preset) : Utilisée pour forcer la sortie Q à 1 instantanément, généralement active quand elle est à 0.
- CLR (Clear) : Utilisée pour forcer la sortie Q à 0 instantanément, aussi généralement active à 0.
- Q : La sortie principale du flip-flop.
- Q' : L'inverse de Q, donc si Q est à 1 Q' sera à 0, et vice-versa.

Dans notre projet, le flip-flop JK est utilisé pour créer des séquences de nombres ou pour stocker et modifier des états dans un processus séquentiel.

Voici l'intérieur de la table de vérité (table transition) du cycle\_nb\_1.

Q3	Q2	Q1	Q0	J3	K3	J2	K2	J1	K1	J0	K0
0	0	0	0	0	1	1	1	0	1	0	1
0	0	0	1	0	1	0	0	1	0	1	1
0	0	1	0	1	1	1	1	0	0	1	1
0	0	1	1	0	0	1	1	0	1	1	1
0	1	0	0	0	0	1	1	0	0	1	1
0	1	0	1	0	1	0	0	1	1	1	0
0	1	1	0	0	0	1	1	0	1	1	1
0	1	1	1	0	1	0	0	1	1	1	1
1	0	0	0	0	0	1	1	0	1	1	1
1	0	0	1	0	0	1	1	0	1	1	1
1	0	1	0	0	0	1	1	0	1	1	1
1	0	1	1	0	0	1	1	0	1	1	1
1	1	0	0	0	0	1	1	0	1	1	1
1	1	0	1	0	0	1	1	0	1	1	1
1	1	1	0	0	0	1	1	0	1	1	1
1	1	1	1	0	1	0	0	1	1	1	0

FIGURE 4.5 – Table de vérité du cycle\_nb\_1

Pour construire ce tableau, nous faisons appel à une table de transition.

Q <sub>n</sub>	Q <sub>n+1</sub>	J	K
0 → 0	0	0	X
1 → 0	0	X	1
0 → 1	1	1	X
1 → 1	1	X	0

FIGURE 4.6 – Les règles de la construction du tableau de transition JK

L'idée ici est très importante car elle va nous permettre d'afficher les valeurs du cycle de manière séquentielle.

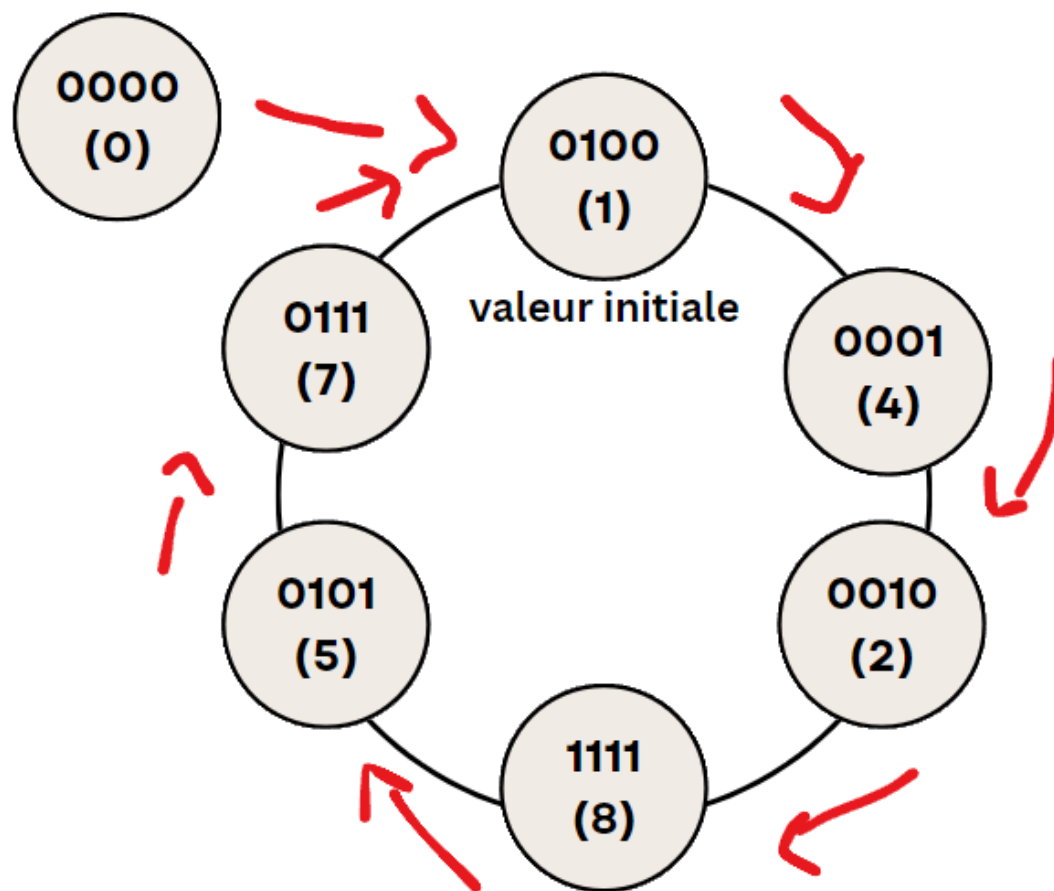


FIGURE 4.7 – Graphe des états cycliques

L'idée est que l'on associe pour chaque valeur du cycle, sa valeur suivante, puis nous construisons la table de transition JK selon la somme binaire de ses valeurs.

Prenons pour exemple 0100 -> 0001 (1 -> 4).

Nous avons ici 4 sommes binaires, construisant J3K3 J2K2 J1K1 J0K0.

Il suffit de commencer par le premier indice des deux nombres : 0 0 -> 0x.

Donc J3K3 == 0x.

Ainsi, pour 0100 -> 0001, nous avons 0x x1 0x 1x.

Il faut maintenant faire ça pour toutes les valeurs du cycle. Pour les états non utilisés, on laisse simplement une rangée de "x" pour laisser Logisim mettre les états qu'il souhaite (les valeurs sont arbitraires).

En suivant cette logique pour le reste de ces circuits, nous parvenons à créer notre premier cycle en logique séquentielle : **cycle\_test**.

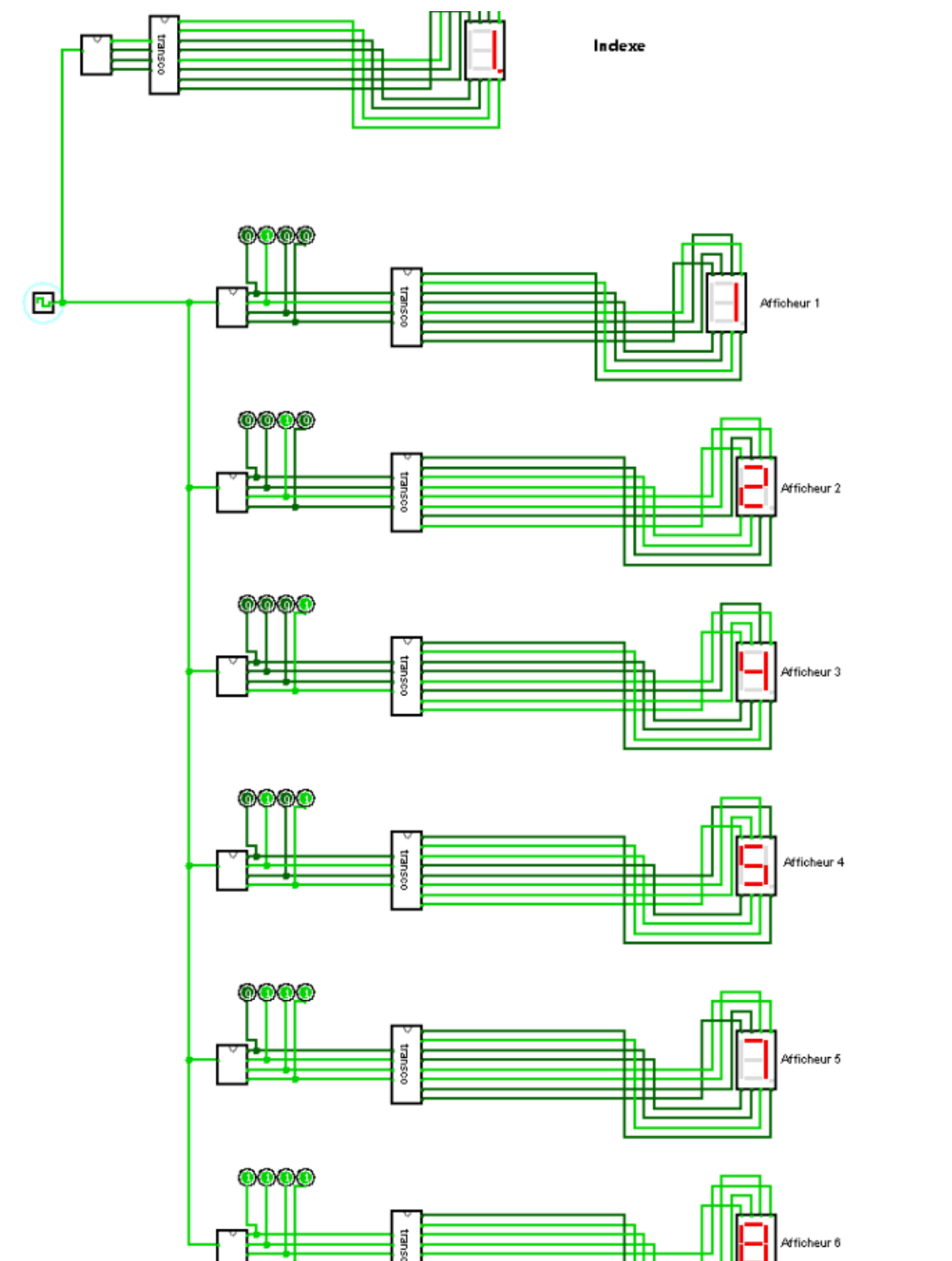


FIGURE 4.8 – Le cycle\_test

Sur ce cycle, nous pouvons voir quelques différences par rapport à notre cycle final en logique combinatoire :



- Pour chaque afficheur on a une petite boîte contenant les **cycle\_complet\_nb\_x**
- En entrée on a une horloge à la place de 4 entrées binaires.

Sur ce cycle de test, nous avons effectivement chaque valeur du cycle sur chaque afficheur, qui tourne de manière séquentielle. Nous avons le multiplicateur en haut pour nous situer à quel niveau du cycle on se trouve et on peut simplement assister à ce joli pivotement du cycle.

Nous voyons enfin que chacun des afficheurs affiche le nombre cyclique 142857 chacun leur tour.

Maintenant qu'on a fait des tests et compris la logique de l'affichage séquentiel, nous pouvons mettre en oeuvre une manière plus élégante de faire les choses : un seul cycle pour notre circuit final.

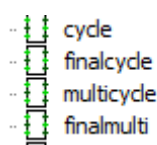


FIGURE 4.9 – Les circuits cycle, finalcycle, multicycle et finalmulti

Ici nous avons simplement un seul circuit avec les flip-flops JK pour constituer le cycle et un autre pour constituer le multiplicateur.

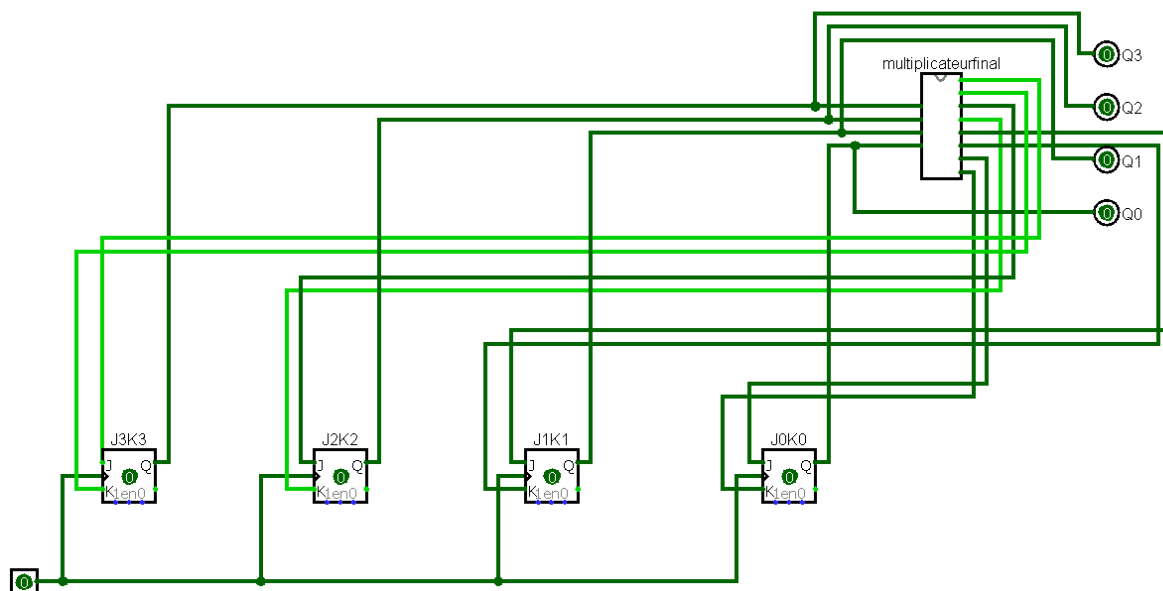


FIGURE 4.10 – En exemple le circuit multi\_cycle

C'était le but initial de ma démarche qui était de faire un seul cycle pour tous les états du nombre cyclique, par rapport aux circuits précédents qui étaient de faire 6 cycles

différents pour chacun des états.

Voyons maintenant le circuit final main et comprendre comment tout cela se construit.

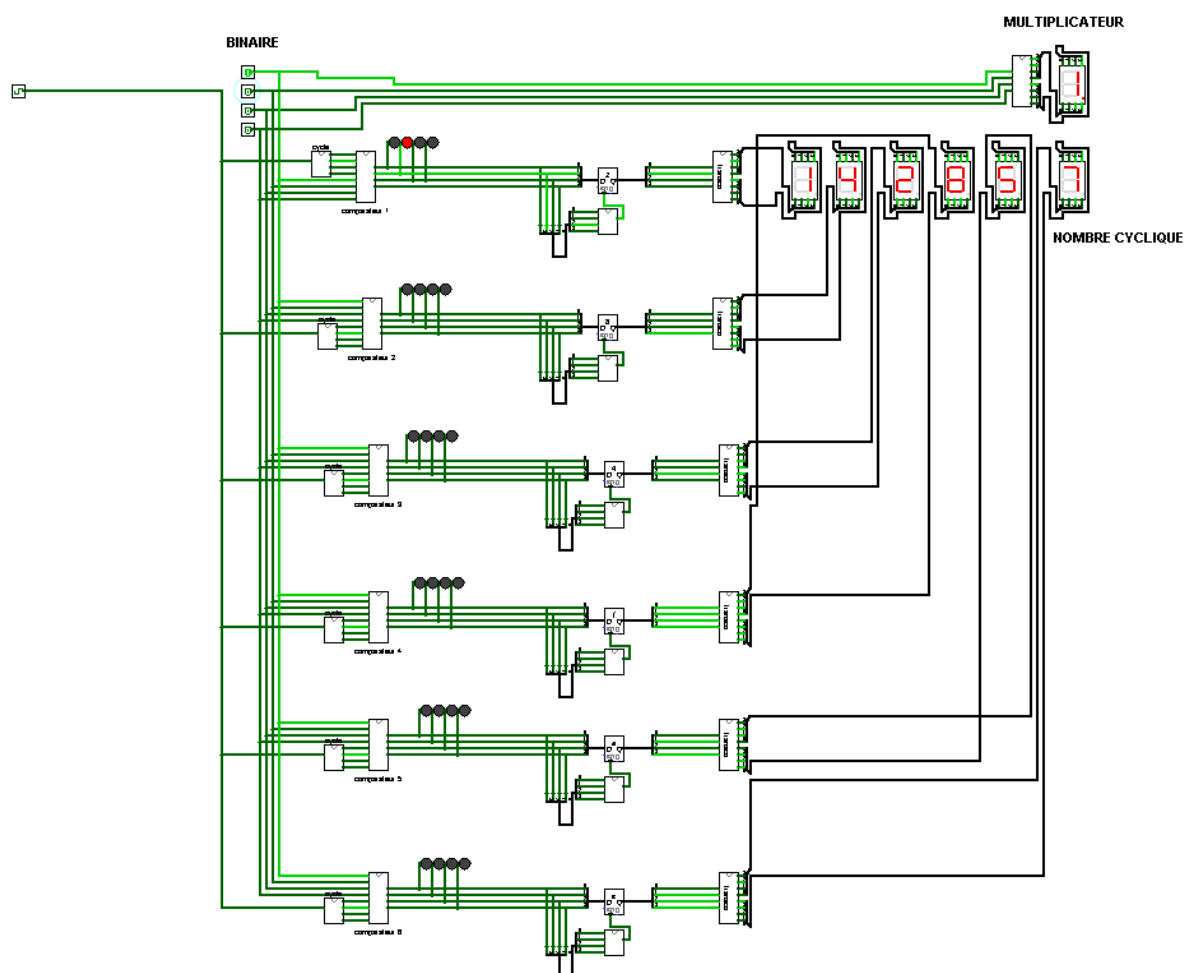


FIGURE 4.11 – Circuit main

Décomposons étape par étape ce circuit.

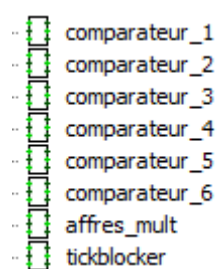


FIGURE 4.12 – Les derniers circuits constituant le circuit final

Voici la liste finale des circuits constituant notre circuit final en logique séquentielle. Commençons par voir ensemble les circuits **comparateurs\_x..**

Q3m	Q2m	Q1m	Q0m	Q3c	Q2c	Q1c	Q0c	S3	S2	S1	S0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1	0	0	0	0
0	0	0	0	0	0	1	0	0	0	0	0
0	0	0	0	0	0	1	1	0	0	0	0
0	0	0	0	0	1	0	0	0	0	0	0
0	0	0	0	0	1	0	1	0	0	0	0
0	0	0	0	0	1	1	0	0	0	0	0
0	0	0	0	0	1	1	1	0	0	0	0
0	0	0	0	1	0	0	0	0	0	0	0
0	0	0	0	1	0	0	1	0	0	0	0
0	0	0	0	1	0	1	0	0	0	0	0
0	0	0	0	1	0	1	1	0	0	0	0
0	0	0	0	1	1	0	0	0	0	0	0
0	0	0	0	1	1	0	1	0	0	0	0
0	0	0	0	1	1	1	0	0	0	0	0
0	0	0	0	1	1	1	1	0	0	0	0
0	0	0	1	0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1	0	0	0	0
0	0	0	1	0	0	1	0	0	0	0	0
0	0	0	1	0	1	0	0	0	0	0	0
0	0	0	1	0	1	0	1	0	0	0	0
0	0	0	1	0	1	1	0	0	0	0	0
0	0	0	1	1	0	0	0	0	0	0	0
0	0	0	1	1	0	0	1	0	0	0	0
0	0	0	1	1	0	1	1	0	0	0	0
0	0	0	1	1	1	0	0	0	0	0	0
0	0	0	1	1	1	0	1	0	0	0	0
0	0	0	1	1	1	1	0	0	0	0	0
0	0	0	1	1	1	1	1	0	0	0	0
0	0	1	0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	1	0	0	0	0

FIGURE 4.13 – comparateur\_1

Les comparateurs sont créés à partir d'une table de vérité qui a 8 entrées et 4 sorties. Le but des comparateurs est de comparer les différentes entrées que l'on pourrait avoir et de s'arrêter à la sortie que l'on veut.

Regardons ensemble cette ligne :

0	0	0	1	1	0	1	0	0	0	0	1
---	---	---	---	---	---	---	---	---	---	---	---

FIGURE 4.14 – Ligne du comparateur pour 0001

- Les 4 premières entrées : Q3m, Q2m, Q1m et Q0m correspondent aux 4 sorties.
- Les 4 dernières entrées : Q3c, Q2c, Q1c et Q0c vont nous permettre de choisir les données du comparateur.

On peut lire cette ligne de la manière suivante : lorsque la valeur du cycle est 4 et que le multiplicateur est 3., afficher 4.

Nous construisons les comparateurs de manière verticale.

x1	142857
x2	285714
x3	428571
x4	571428
x5	714285
x6	857142

FIGURE 4.15 – Le cycle et ses multiples

Les valeurs pour le premier comparateur, donc le premier multiplicateur serait d'avoir 124578. (ex : lorsque j'ai 2 en entrée, je veux que la sortie soit 2 et 2 uniquement).

En suivant cette logique de manière analogue, nous avons pu créer 6 comparateurs allant de 124578 à 741852.

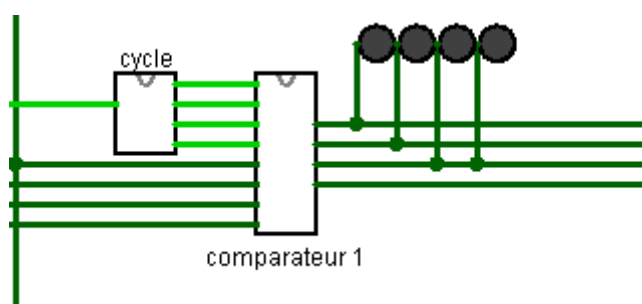


FIGURE 4.16 – Câblage du cycle et du comparateur\_1 dans main

Concrètement, le cycle envoie une information à chaque coup de clock au comparateur, et lorsque c'est l'information que le comparateur veut, il la laisse passer dans la suite du circuit.

Encore une fois pour le premier multiplicateur 1 et le premier comparateur, l'information qu'on veut que le comparateur fasse passer est bien le 1.

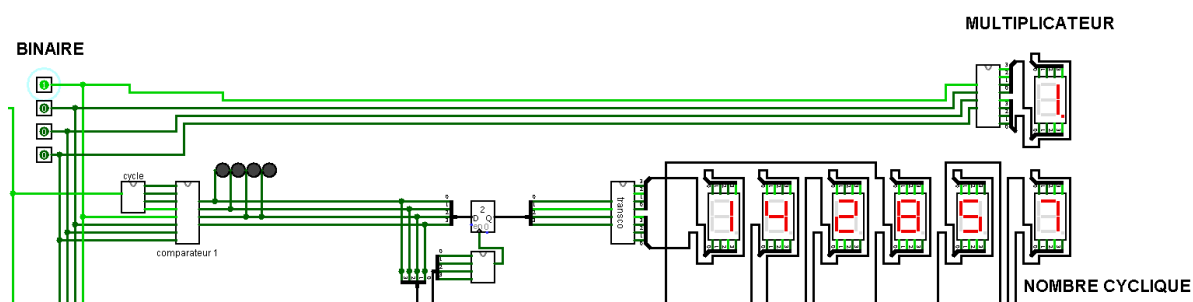


FIGURE 4.17 – Cycle lorsque le multiplicateur est 1.

Voyons ensuite cette partie du cycle

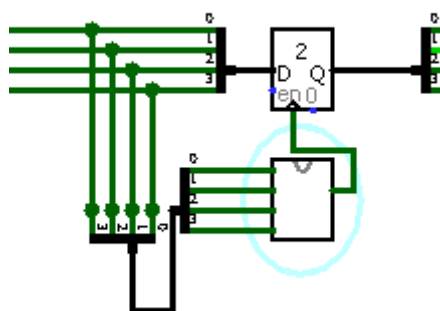


FIGURE 4.18 – Le register et le tickblocker dans main

Nous avons ici deux éléments, le register et le tickblocker.

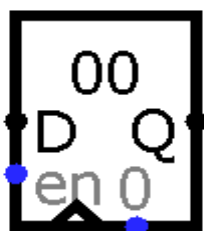


FIGURE 4.19 – Le register dans Logisim

Le register dans le circuit final sert à stocker la valeur binaire à un certain moment du cycle de fonctionnement du compteur. Dans le contexte de notre nombre cyclique 142857, chaque register est utilisé pour retenir la valeur actuelle du compteur jusqu'à ce que le cycle progresse.

Dans l'exemple ci-dessus, le but est de retenir la valeur 1 jusqu'à la fin. (jusqu'à ce qu'on change de multiplicateur).

En couplant celui-ci avec le tickblocker, nous pouvons synchroniser correctement les transitions du cycle.

Q3	Q2	Q1	Q0	S
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	1
0	1	0	0	1
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	1
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

FIGURE 4.20 – Table de vérité du tickblocker

Dans cette table, nous laissons passer tous les ticks (les coups d'horloge) sauf quand le circuit est éteint (0).

Avec le register il permet d'aligner les mises à jour en bloquant temporairement les ticks jusqu'à ce que tous les composants soient prêts à changer d'état simultanément.

Ces deux éléments sont finalement branchés à nos transcodeurs initiaux puis connectés à nos afficheurs 7-segments comme le reste de nos circuits.

Finalement, en lançant la simulation on peut assister à cette merveilleuse séquence de notre nombre cyclique devant nous. Il suffit juste d'activer les ticks et de changer le multiplicateur à notre guise.

# Chapitre 5

## Conclusion

Pour conclure, ce projet a été très enrichissant malgré quelques difficultés (de groupe, de matériel et de problèmes personnels.) En retravaillant ce rapport, je me suis rendu compte de pas mal d'erreurs que j'ai pu faire, que ce soit au niveau du style et du contenu.

Dans le fond du projet, j'aurais pu peut-être utiliser des techniques ou des outils plus élégants (i.e : grâce au comparateur à 3 bits ou à l'interrupteur pour un démultiplexeur (DMUX)). Dans l'ensemble, le fruit de ce travail m'a permis de réaliser quelque chose de tangible pour les deux parties du projet, et ainsi l'imitation du tour de carte présenté par notre professeur à été un succès !