

SYSTEMES LOGIQUES TRAVAIL PRATIQUE TP03 - SOLUTIONS

3. TP03: CIRCUITS ARITHMÉTIQUES ET SÉQUENTIELS

Le travail pratique TP03 voit l'étude de circuits arithmétiques simples dans une première partie. La deuxième partie aborde la synthèse d'un circuit séquentiel.

3.1 PARTIE 1, ADDITIONNEUR-SOUSTRACTEUR

Le circuit présenté en Figure 3.1 est un additionneur-soustracteur 4-bit. Il comprend huit entrées et dix sorties. A et B sont des nombres codés en binaire 4-bit présentés en entrée. Ils sont constitués des bits respectifs A_3, A_2, A_1, A_0 et B_3, B_2, B_1, B_0 . Les valeurs de sorties sont encodées en binaire 4-bit tel que, $A+B=(P_3, P_2, P_1, P_0)$ et $A-B=(M_3, M_2, M_1, M_0)$. L'overflow O et l'underflow U déterminent un dépassement de capacité.

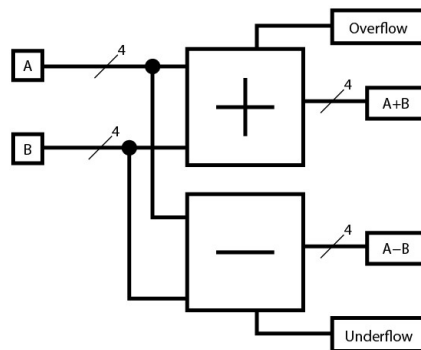


Figure 3.1: Schéma-bloc d'un additionneur-soustracteur

Le but de cet exercice est de concevoir les circuits logiques réalisant l'additionneur-soustracteur suivant une méthode systématique.

Les tâches proposées sont les suivantes.

- Dresser la table de vérité de l'additionneur, puis

Le nombre d'entrées du circuit est de 8, et ainsi la table de vérité comprend 2^8 lignes. De ce point de vue, le problème est peut être considéré complexe, et il convient d'appliquer une technique efficace à sa solution.

Les algorithmes de "Divide and conquer" sont adaptés à trouver des méthodes de solution à des algorithmes complexes. Il s'agit d'une méthode algorithmique appliquée à divers types de problèmes et qui peut permettre de trouver des solutions algorithmiques efficaces, par exemple des algorithmes de tri, ou des algorithmes de multiplication de grands nombres, etc. Il n'y a pas de recette qui permette de façon systématique d'appliquer la méthode et de converger vers une solution optimale. La stratégie de "divide-and-conquer" dicte la résolution d'un problème complexe par l'application des étapes suivantes. 1. Un problème complexe est partitionné en sous-problèmes qui sont eux-mêmes des instances de type identique au problème complexe de base. 2. Les sous-problèmes sont résolus récursivement. 3. Les résultats sont combinés de façon appropriée. La méthode est largement documentée dans la littérature traitant de l'algorithmique; elle est appliquée de façon pratique dans la suite.

Phase de division du problème

Les sous-problèmes de type identique sont les additions des 1-bit de rang successifs, formant quatre problèmes quasiment identiques (Prob.1 à Prob.4), et de type identique au problème général, Figure 3.2.

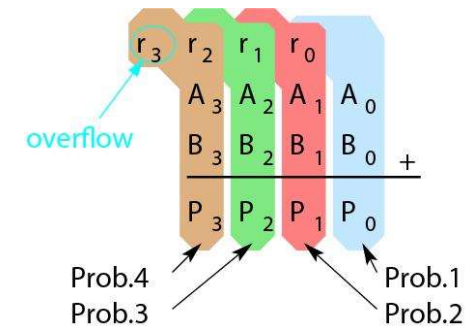


Figure 3.2: Problème général, et phase de division en quatre sous-problèmes de natures identiques.

Phase de conquête, première partie: solution des sous-problèmes

Les problèmes sont solutionnés. Notons que le problème 1 consiste à additionner deux bits sans prendre en compte d'entrée de retenue, car il s'agit des bits de rang 0. Les problèmes 2 à 4 sont identiques.

Problème 1

A_0	B_0	r_0	P_0
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

r_0	B_0	P_0	B_0
0	0	0	1
0	1	1	0

$$r_0 = A_0 \cdot B_0$$

$$P_0 = A_0 \oplus B_0$$

Figure 3.3: Sous-problème relatif aux bits de rang 0.

Problèmes 2, 3 et 4

r_{n-1}	A_n	B_n	r_n	P_n
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

r_n	B_n
0	0 1 0 1
1	0 1 1 1

$$r_n = A_n \cdot B_n + r_{n-1} \cdot A_n + r_{n-1} \cdot B_n$$

$$P_n = A_n \oplus B_n \oplus r_{n-1}$$

Figure 3.4: Sous-problème relatif aux bits de rangs supérieures à 0.

Phase de conquête, deuxième partie: combinaison appropriée des résultats

Les résultats de sous-problèmes sont combinés. Dans ce cas, les circuits obtenus comme sous-blocs sont électriquement connectés afin de réaliser la solution du problème général, l'additionneur 4-bit, Figure 3.5.

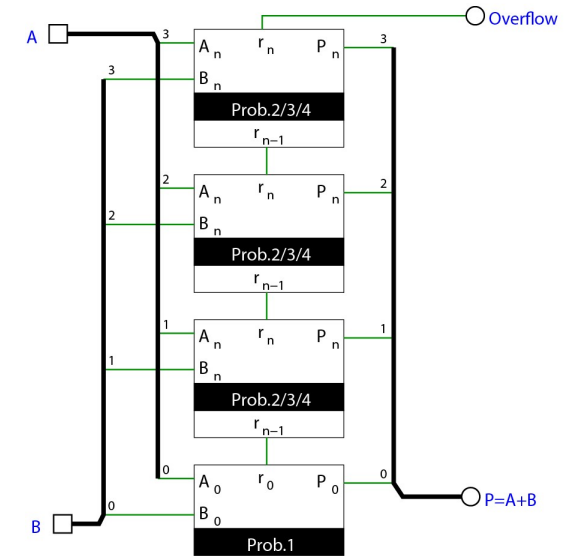
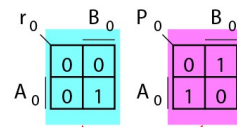


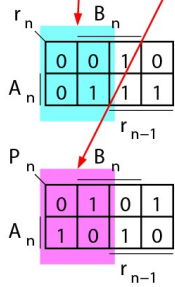
Figure 3.5: Additionneur réalisé de façon modulaire.

Une solution alternative et plus générale (optimisée) est possible et résulte de l'observation du fait que le problème 1 se réduit à la résolution d'un cas particulier des problèmes 2, 3 et 4. En effet, les tables de vérités pour r_e et P_0 du problème 1 sont des sous-ensembles des tables de vérités de r_n et P_n des problèmes 2, 3 et 4. Les tables de vérités sont identiques à la condition que r_{n-1} du problème 1 soit égal à 0. Electriquement, il est donc possible d'utiliser les mêmes circuits pour les quatre sous-blocs, à la condition que l'entrée r_{n-1} du bloc 0 soit connectée à logic-0, Figure 3.6.

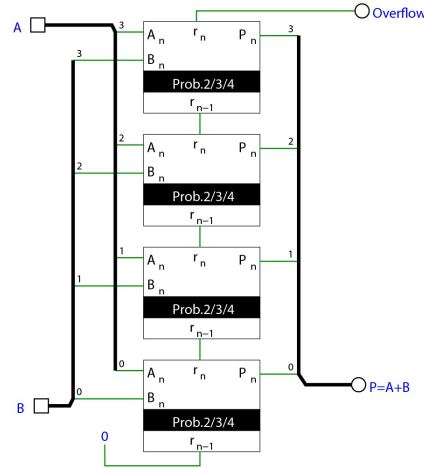
Prob.1:



Prob.2/3/4:



(a)



(b)

Figure 3.6: Circuit général n'utilisant qu'un type unique de module.

- développer et vérifier la fonctionnalité dans logisim-evolution.

Le fichier de solution est disponible sous le nom TP03S.circ sur le site Moodle. L'additionneur est nommé `exercice_2`.

- Dresser la table de vérité du soustracteur, puis,

Le soustracteur 4-bit est développé suivant la méthode appliquée plus haut.

Phase de division du problème, Figure 3.7

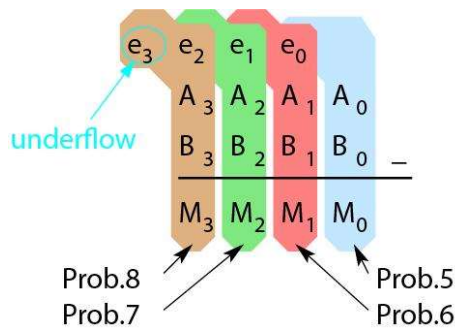


Figure 3.7: Problème général, et phase de division en quatre sous-problèmes de natures identiques.

Phase de conquête, première partie: solution des sous-problèmes

Problème 5, Figure 3.8

A_0	B_0	e_0	M_0
0	0	0	0
0	1	1	1
1	0	0	1
1	1	0	0

e_0	B_0	M_0	B_0
0	0	0	0
0	1	1	1
1	0	0	1
1	1	0	0

Figure 3.8: Sous-problème relatif aux bits de rang 0.

Problèmes 6, 7 et 8,

e_{n-1}	A_n	B_n	e_n	M_n
0	0	0	0	0
0	0	1	1	1
0	1	0	0	1
0	1	1	0	0
1	0	0	1	1
1	0	1	1	0
1	1	0	0	0
1	1	1	1	1

e_n	B_n
0	0
0	1
1	0
1	1

e_{n-1}	B_n
0	0
0	1
1	0
1	1

Figure 3.9: Soustracteur réalisé de façon modulaire.

Phase de conquête, deuxième partie: combinaison appropriée des résultats,

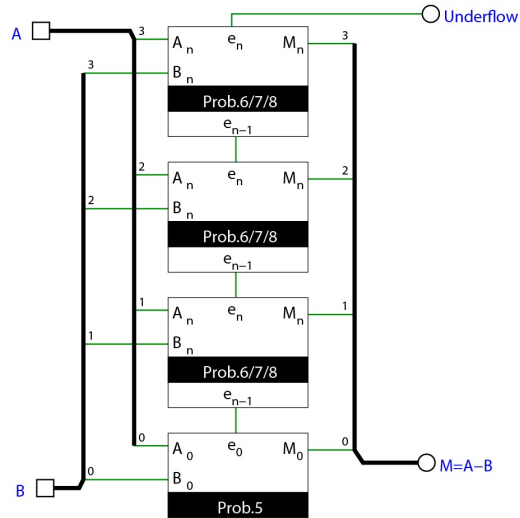
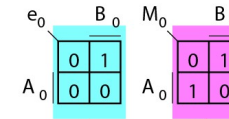


Figure 3.10: Soustracteur réalisé de façon modulaire.

Solution alternative optimisée, Figure 3.11.

- développer, insérer et vérifier la fonctionnalité du soustracteur et du circuit complet dans logisim-evolution.
Le soustracteur est nommé `exercice_4`.
- Finalement, porter le circuit sur le système-cible et tester fonctionnellement au moyen des DIP-switches en entrée et LEDs en sortie.

Prob.1:



Prob.2/3/4:

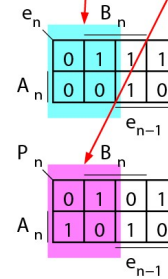


Figure 3.11: Circuit général n'utilisant qu'un type unique de module.

3.2 PARTIE 2, LOGIQUE SÉQUENTIELLE

Un feu de signalisation routière est composé de trois lampes. Dans le cadre de cet exercice, les trois lampes sont trois LEDs, de couleur identique.

L'état des trois feux change à chaque fois que un bouton de contrôle unique est activé. Les états consécutifs évoluent suivant que le bouton soit pressé ou relâché, et obéissent au schéma présenté en Figure 3.12.

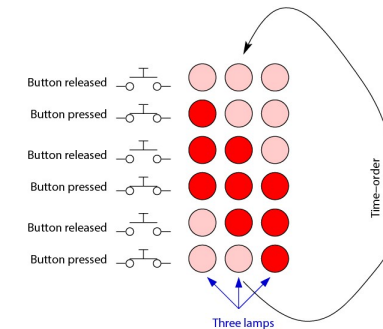


Figure 3.12: Schéma de contrôle de feux de signalisation

Le circuit de contrôle nécessite ainsi de la mémoire afin d'établir l'état actuel et futur. Il s'agit donc d'un circuit séquentiel. Celui-ci est réalisé au moyen de bascules SR.

Les tâches proposées sont les suivantes.

- Développer le système implémentant la fonctionnalité présentée plus haut dans logisim-evolution au moyen de portes logiques, de bascules asynchrones SR, de LEDs et de boutons uniquement.

Chacune des LEDs est contrôlée par la sortie d'une bascule SR. La fonction de la bascule SR est la suivante.

Reset	Set	Fonction
0	0	Etat de mémorisation
1	0	La LED est éteinte
0	1	La LED est allumée

La solution au problème sous forme de développement du circuit de contrôle vient de l'étude des états et conditions de transition pour chaque LED.

Considérant la LED L1, les états de mémorisation sont présentés en Figure 3.13.

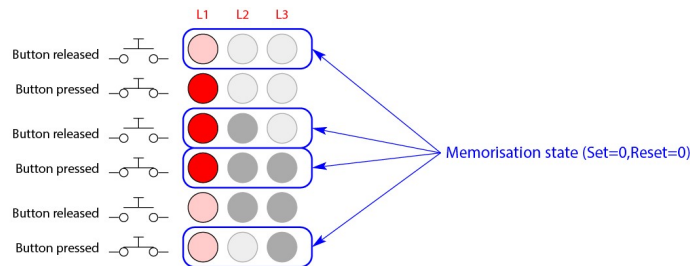


Figure 3.13: Etude des états de mémorisation relatifs à L1.

Lorsque ni le Set ni le Reset ne sont activés, alors le système mémorise l'état courant et ne progresse pas. Le système est donc séquentiel asynchrone.

Considérant la LED L1, la condition de Set est présentée en Figure 3.14.

L'interprétation textuelle de la condition de Set est la suivante: si la LED L1 est éteinte, et que L2 et L3 sont éteintes, et que le bouton B1 est pressé, alors un signal Set est transmis à la bascule SR. Algébriquement: $Set(L1) = \bar{L1} \wedge \bar{L2} \wedge \bar{L3} \wedge B1$. Cette dernière expression est réalisée au moyen d'une porte logique AND à quatre entrées dont la sortie est connectée à l'entrée Set de la bascule SR qui contrôle la LED L1.

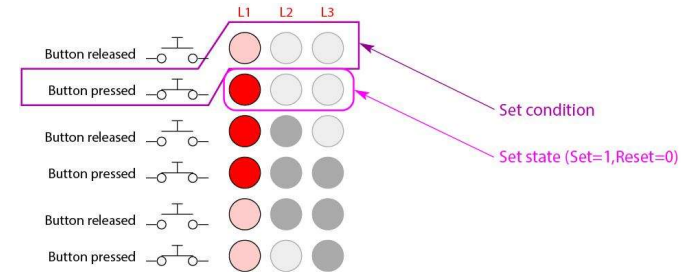


Figure 3.14: Etude des états des conditions de Set de L1.

Considérant la LED L1, la condition de Reset est présentée en Figure 3.15.

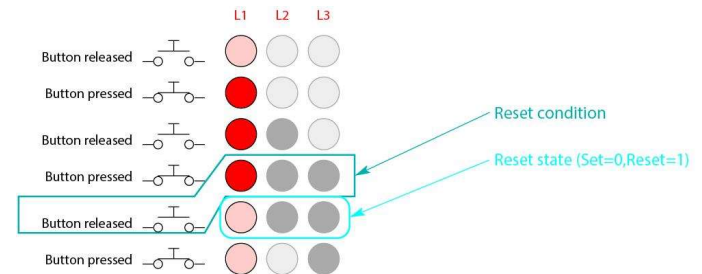


Figure 3.15: Etude des états des conditions de Set de L1.

$Reset(L1) = L1 \wedge L2 \wedge L3 \wedge \bar{B1}$. Afin de permettre un reset général qui remette le système dans son état initial, un signal supplémentaire est ajouté et connecté à une entrée B2 (parfois nommé POR, Power-on reset, de façon discutable). $Reset(L1) = (L1 \wedge L2 \wedge L3 \wedge \bar{B1}) \vee B2$. Cette dernière expression est réalisée au moyen d'une porte logique AND à quatre entrées dont la sortie est connectée à l'entrée d'une porte logique OR à deux entrées dont l'autre entrée est connectée au Reset général B2 et dont la sortie est connectée au Reset de la bascule SR qui contrôle la LED L1.

Ainsi, le contrôle séquentiel de la LED L1 est terminé. Un raisonnement similaire est conduit afin de développer les circuits de contrôle des LEDs L2 et L3.

La solution porte le nom `exercice_5`.

- Porter et vérifier les circuits sur le système-cible. (Si votre résultat ne fonctionnait pas, par exemple en assignant les trois LEDs sur la droite, essayez d'assigner les trois LEDs sur la gauche; quel est la raison de ce problème ?).

Deux types de comportement non-idéaux peuvent être à l’origine d’un résultat qui ne correspond pas aux simulations sur logisim-evolution.

- Les boutons-poussoirs sont utilisés dans cette application en mode pressé, et relâché. Les boutons-poussoirs sont des dispositifs électriques simples et bon marchés qui ont l’inconvénient de ne pas réaliser l’opération idéale d’un switch (interrupteur). Des transitoires de tension nommés rebonds existent, qui sont interprétés par les circuits avals comme des flancs légitimes. Dans un cas extrême, le contrôleur de LEDs peut être amené dans un état qui n’est pas prévu et duquel il ne peut plus s’échapper. Pour cette raison, un reset général est indispensable. Des systèmes ou circuits anti-rebonds doivent être utilisés.
- Les SR-Latches sont évidemment très sensibles à des délais non-intentionnels dans leur fonctionnement car ils sont constitués d’une architecture rebouclée (les sorties sont rebouclées sur les entrées). Le circuit de contrôle des LEDs étudié comprend des SR-Latches qui sont placés dans une architecture rebouclée (les sorties sont connectées sur les entrées). Ainsi, un délai non-intentionnel pourrait générer une erreur locale, qui à son tour, par le biais des connections rebouclées sur les entrées génère une erreur globale. Finalement, un comportement semi-chaotique semble observé.

Les délais non-intentionnels ont deux origines. Les délais entre l’entrée du bouton-poussoir jusqu’aux (six) entrées des portes logiques formant les (trois) SR-Latches ne sont pas connus et peuvent être différents. D’autre part, le circuit généré par Quartus et qui est téléchargé sur la FPGA n’est pas connu ni contrôlé, et il est possible que des délais excessifs soient dus au fait que certains circuits soient placés à grande distance. Rappelons que logisim-evolution ne prend pas les délais en compte et réalise une simulation logique, alors que le circuit implémenté dans la FPGA résulte d’une réalisation électrique qui est soumise aux délais.

Un circuit d’entrée synchrone est réalisé dans le circuit `exercice_5_b` qui permet de s’affranchir des délais externes, et de garantir que toutes les portes logiques du contrôleur de LEDs reçoivent le signal du bouton-poussoir de façon synchrone. Le système générale est resynchronisé, et une horloge système raisonnable doit être choisie, par exemple 256 Hz.

La solution a été guidée vers le développement d’une logique de contrôle asynchrone faisant usage de SR-Latches. Remarquons que la solution réalise une machine d’états finis de Medvedev. Les points-mémoire qui mémorisent l’état courant sont dans ce cas réalisés par des SR-latches. Le développement de la logique de transition est présenté par une raisonnement sur le scénario qui permet d’exprimer les formules algébriques des six signaux de contrôle nécessaires: L1-set, L1-reset, L2-set, L2-reset, L3-set, L3-reset. Les étapes du développement classique consistant à développer les six tables de vérité ainsi que l’optimisation par six diagrammes de Karnaugh ont été ainsi omises car le problème est suffisamment simple.

D’autres solutions au problème sont possibles et valides dans un contexte général qui sont présentées dans la suite.

3.2.1 SOLUTIONS ASYNCHRONES SUPPLÉMENTAIRES

logisim-evolution ne dispose pas de SR-Latches dans ses librairies car un SR-Flip-Flop est disponible. Une SR-FF est un système dont les deux entrées S et R sont synchrones et donc évaluées au flanc montant de l’horloge. Cependant, un SR-FF dispose également de deux entrées S et R asynchrones qui ont un effet immédiat, c’est-à-dire non synchronisé sur le flanc montant de l’horloge. Contrôler un SR-FF uniquement par ses entrées S et R asynchrones revient au comportement d’un SR-Latch.

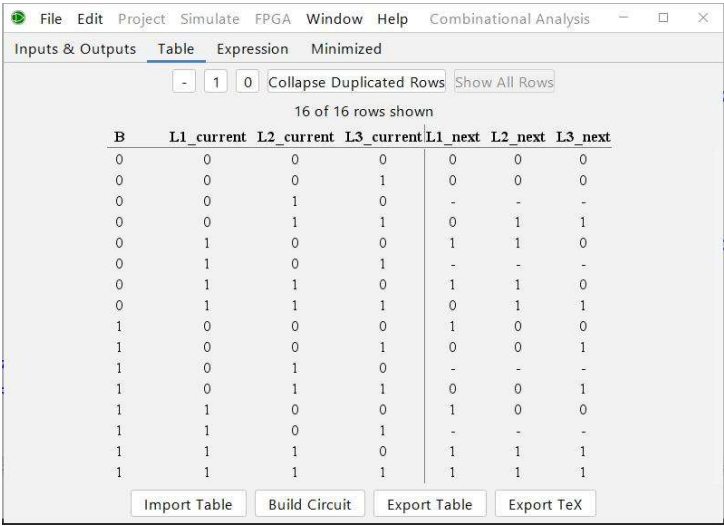
Le circuit `exercice_5_c` présente cette utilisation. L’entrée l’horloge est mise à la terre. Les entrées synchrones S et R sont laissées flottantes car elles ne sont jamais évaluées; elles peuvent aussi être mises à la terre.

C’est un cas particulier dans lequel une entrée laissée flottante ne cause pas de problème car elle ne peut physiquement pas influencer le circuit interne.

Le circuit `exercice_5_d` présente une autre utilisation du SR-FF dans laquelle les entrées synchrones S et R sont utilisées. L’horloge doit être très rapide pour le système situé en aval, par exemple l’humain qui observe les lampes, afin que ce dernier puisse considérer le fonctionnement comme pseudo-asynchrone et donc identique au comportement d’un SR-Latch.

3.2.2 SOLUTION SYNCHRONE

Une machine d’états finis classique peut être utilisée. Dans ce cas, des DFFs mémorisent l’état courant des lampes. La logique de transition est créée par développement des tables de transitions sur la base du scénario. La table de transition complète est présentée en Figure 3.16.



B	L1_current	L2_current	L3_current	L1_next	L2_next	L3_next
0	0	0	0	0	0	0
0	0	0	1	0	0	0
0	0	1	0	-	-	-
0	0	1	1	0	1	1
0	1	0	0	1	1	0
0	1	0	1	-	-	-
0	1	1	0	1	1	0
0	1	1	1	0	1	1
1	0	0	0	1	0	0
1	0	0	1	0	0	1
1	0	1	0	-	-	-
1	0	1	1	0	0	1
1	1	0	0	1	0	0
1	1	0	1	-	-	-
1	1	1	0	1	1	1
1	1	1	1	1	1	1

Figure 3.16: Table de transition de la FSM dans l’interface *Combinational Analysis* de logisim-evolution.

Nous observons que la table encode les six transitions. Également, lorsqu’une transition a été effectuée il faut que le système reste dans son état et la table encode ceci comme une transition vers l’état courant. Finalement, les états fantômes ne sont pas traités (ne sont pas origines d’une transition) et il s’agit donc d’une application non-critique. L’encodage des états est trivial dans ce cas; le code représentant un état est identique au signal contrôlant les lampes; par exemple l’état 000 décrit l’état dans lequel les trois lampes sont éteintes.

Le circuit `exercice_5_e` présente la FSM de Medvedev.