heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

ReDS

Electronique Numérique

4ème tome

Systèmes séquentiels avancés

MSS complexes

Etienne Messerli
Sylvain Krieg
Mars 2013
Version 0.4b

Hes·so
Haute Ecole Spécialisée
de Suisse occidentale

Mise à jour de ce manuel

Je remercie tous les utilisateurs de ce manuel de m'indiquer les erreurs qu'il comporte. De même, si des informations semblent manquer ou sont incomplètes, elles peuvent m'être transmises, cela permettra une mise à jour régulière de ce manuel.

Etienne Messerli

Contact

Auteurs:	Etienne Messerli	Sylvain Krieg
e-mail :	etienne.messerli@heig-vd.ch	-
Tél:	+41 (0)24 / 55 76 302	-

Coordonnées à la HEIG-VD :

Institut REDS
HEIG-VD
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud
Route de Cheseaux 1
CH-1400 Yverdon-les-Bains
Tél : +41 (0)24 / 55 76 330
Internet : http://www.heig-vd.ch

Institut REDS
Internet : http://www.reds.ch

Autres personnes à contacter:

M. Gilles Curchod gilles.curchod@heig-vd.ch Tél direct +41 (0)24/55 76 259

Table des matières

Chapitre 1 Décomposition machine séquentielle complexe 1

Structures des unités de commande . . . 5
Structure de l’unité de traitement . . . 8
Combinaison des structures d’UC et d’UT . . . 10

Chapitre 2 Méthode de conception 13

Chapitre 3 Application de la méthodologie de conception à la multiplication 17

3-1. Spécification de la multiplication séquentielle ... 17
Schéma-bloc général . . . 18
Algorithme . . . 19
3-2. Algorithme : organigramme ou graphe ... 19
3-3. Partition unité de commande / unité de traitement ... 23
Identification des fonctionnalités de l'UT . . . 26
Unité de traitment . . . 27
Unité de commande . . . 28
3-4. Passage d’un organigramme à un graphe ... 28
Remarque importante . . . 31
3-5. Organigramme détaillé ... 32
3-6. UC câblée ... 35
Decription VHDL de l’UC câblée . . . 39
3-7. Exercices ... 39

Chapitre 4 Exemple :Distributeur automatique de billets 41

Introduction. . . . 41
4-1. Spécifications du distributeur ... 42
Spécifications préliminaires. . . . 42

Fonctionnement désiré. 42
Collecteur de monnaie. 42
Échangeur de monnaie. 44
Description des signaux 44
Mécanisme distributeur de billets. 45

4-2. Schéma-bloc général et organigramme grossier 46

4-3. Partition commande / traitement 48

4-4. Exercices 51

4-5. Organigramme détaillé 52

4-6. UC câblée 57

Description VHDL de l'UC câblée 59

4-7. Exercices 61

Chapitre 5 Unité de commande microprogrammée 63

5-1. Exercices 71

5-2. Minimisation de la mémoire de microprogramme 72

5-3. Codage des sorties 76

5-4. Exercices 78

5-5. Sous-programmes et interruptions 80

Chapitre 6 Unité de traitement universelle 85

Bibliographie 97

Manuel de la HEIG-VD 97
Médiagraphie 98

Lexique 99

Chapitre 1

Décomposition machine séquentielle complexe

La méthode de conception d'une MSS que nous avons étudiée au chapitre MSS simple, basée sur la description du comportement désiré à l'aide d'un graphe d'états, ne permet pas de s'attaquer à des MSS complexes. À partir de quelques dizaines d'états et trois ou quatre entrées, le graphe devient en effet fort malaisé à établir, la recherche d'un codage efficace devient un vrai casse-tête et le test devient un problème majeur.

Lorsqu'un problème devient trop volumineux pour être étudié et résolu globalement, la stratégie qui a fait ses preuves dans toutes sortes de domaines consiste à découper progressivement ce gros problème en problèmes partiels, plus petits et plus faciles à résoudre, jusqu'à ce que chaque problème partiel puisse être traité sans trop de difficulté avec les moyens à disposition. Nous chercherons donc à découper les machines séquentielles complexes, c'est-à-dire comportant beaucoup d'entrées, de sorties et d'états, en plusieurs machines plus simples.

Il existe des approches théoriques visant au découpage d'une MSS complexe. Dans la pratique, elles n'ont que peu d'utilité car il faut généralement commencer par décrire la machine complète de façon détaillée, ce que

nous voulons précisément éviter. De plus, elles ne favorisent pas l'identification des sous-machines déjà réalisées, ou pouvant être adaptées sans difficulté à partir de celles-ci, ce qui est précisément notre but. Comme exemple de sous-machines existantes, nous pouvons citer toutes celles composées d'une fonction standard, tel que compteur, registre à décalage, comparateurs, .. Ces sous-machines sont très facilement adaptables.

La décomposition d'une MSS, partie essentielle de la conception est au moins autant **un art qu'une science**. L'imagination et l'expérience du concepteur auront donc une influence prépondérante sur la qualité des solutions générées. Nous allons nous efforcer de développer l'une et l'autre par la pratique. Mais l'art est toujours tributaire de la technique, qui fera ici la différence entre le bricoleur et l'ingénieur. Nous allons donc acquérir les techniques nécessaires.

Parce que plus facile à concevoir et à réaliser, le découpage le plus utilisé est un découpage hiérarchique : une des sous-machines gère le fonctionnement des autres. La figure 1- 1 schématise une telle situation, avec un découpage en 3 parties d'une machine complexe : la sous-machine 0 commande les sous-machines 1 et 2 et peut être appelée machine principale (dans toute hiérarchie, celui qui commande se croit plus important que les autres).

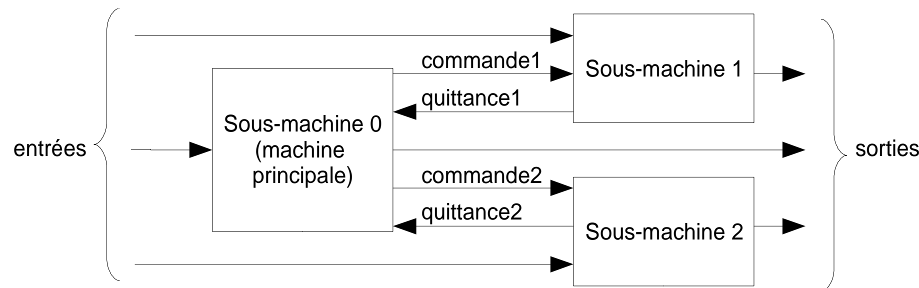


Figure 1- 1 : Découpage en sous-machines

Chaque flèche dans la figure 1- 1 doit-être vue comme pouvant représenter plusieurs signaux.

Le découpage hiérarchique ne s'arrête pas forcément à deux seuls niveaux hiérarchiques. Si la sous-machine 1 de l'exemple ci-dessus reste trop complexe pour être conçue d'un bloc, rien ne nous empêche de la décomposer à son tour en une sous-sous-machine principale et d'autres sous-sous-machines commandées par cette dernière. Et ainsi de suite. Mais deux niveaux hiérarchiques suffisent dans beaucoup de cas, et ils suffisent certainement pour étudier les notions de base qui font l'objet de ce cours. Dans ce qui suit, nous nous restreindrons donc à deux niveaux.

La différenciation du rôle joué par la machine principale d'une part et les autres sous-machines d'autre part, est un point essentiel de la méthode que nous allons développer. La machine principale est appelée unité de commande ou encore unité de séquencement, ou plus simplement séquenceur (System Controller, Control Unit ou Sequencer, en anglais). Les autres sous-machines sont regroupées sous le nom d'unité de traitement, ou unité d'exécution(Execution Unit ou Data Unit, en anglais). Il en résulte le schéma-bloc général de la figure 1- 2.

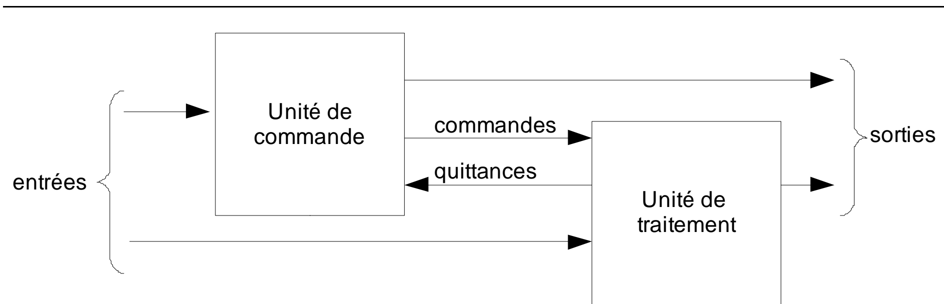


Figure 1- 2 : Séparation UC/UT

Un autre point essentiel de la méthode de synthèse des machines séquentielles complexes est qu'au lieu de voir le fonctionnement d'une MSS complexe comme une succession d'états décrite à l'aide d'un graphe, nous le verrons comme une suite d'opérations réalisées par l'unité de traitement et commandées, dans la séquence voulue, par l'unité de commande. L'avantage réside dans le fait que la suite d'opérations sera beaucoup plus petite que la suite d'états de la MSS. Prenons un exemple : envisageons une MSS qui doit, à un certain point de son fonctionnement, activer une sortie pendant n périodes d'horloge avant de continuer sa séquence. En suivant les méthodes du chapitre précédent, nous introduisons n états dans le graphe, qui serviraient à compter les périodes d'horloge, selon la figure 1- 3 (a). Mais en délégant la tâche de compter les n périodes d'horloge à un compteur faisant partie de l'unité de traitement, comme nous le montre la figure 1- 3 (c) , le graphe de l'unité de commande (qui est aussi une MSS) ne comportera qu'un seul état pour activer cette sortie. La figure 1- 3 (b) nous montre cette décomposition.

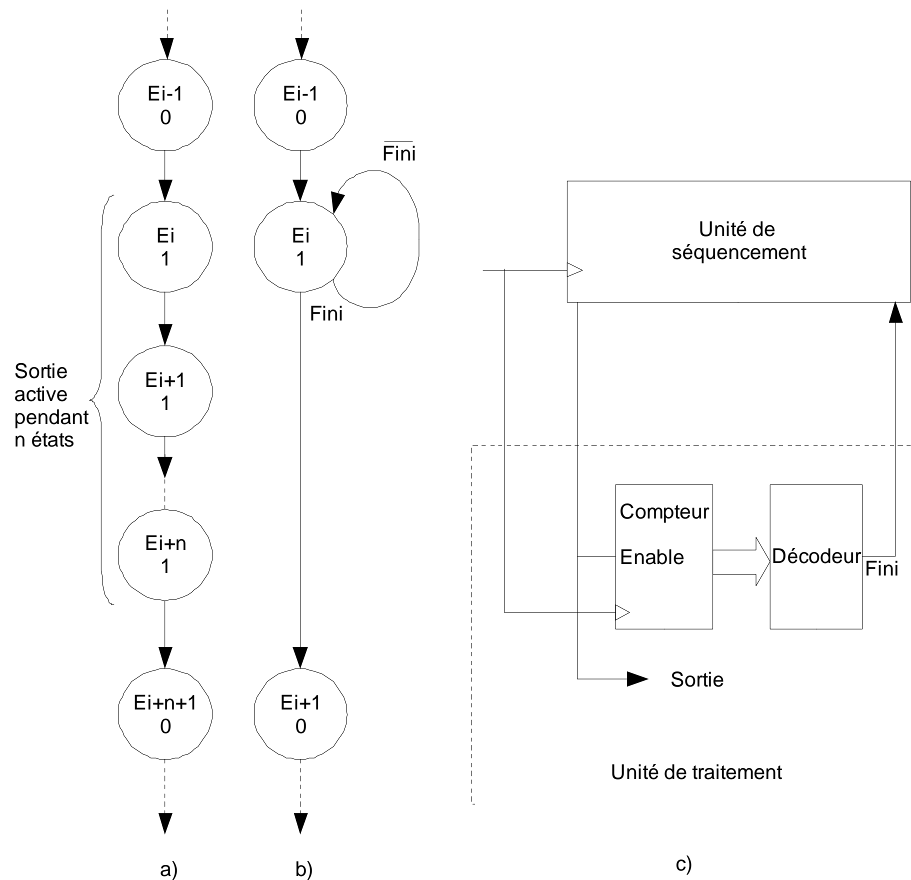


Figure 1- 3 : Traitement du comptage par l'UT

D'une façon générale, l'unité de commande d'une MSS complexe comportera beaucoup moins d'états que n'en aurait comporté la même machine conçue d'un seul bloc selon les méthodes du chapitre précédent (à supposer que nous soyons capables de concevoir cette machine). Par contre une machine conçue en séparant séquencement et traitement va comporter plus de portes et de flip-flops qu'une machine conçue d'un seul bloc. En chiffrant l'exemple ci-dessus nous en obtenons facilement la preuve : pour un nombre total d'états de la MSS conçue d'un seul bloc égal à 75, dont 50 pour générer l'impulsion de sortie, nous aurions besoin d'au moins 5 flip-flops pour la MSS à 26 états mais le compteur comportera 6 flip-flops, ce qui fera un total de 11.

Si la décomposition d'une MSS à tendance à augmenter le nombre de portes et de flip-flops, elle permet par contre de mieux tenir compte des composants existants et donne lieu facilement à des réalisations plus compactes dès qu'un certain seuil de complexité est dépassé. Ce seuil varie bien sûr avec l'apparition de nouveaux circuits et de nouvelles méthodes. Mais dans tous les cas, la décomposition d'une MSS complexe permet de diminuer très sensiblement le coût et la durée de son développement (conception + réalisation + tests), ainsi que les risques d'échec. Une bonne partie de ces gains viennent du fait que l'unité de traitement sera essentiellement

composée de fonctions standard, il est ainsi possible de réutiliser des éléments pour l'unité de traitement. Il faut entendre par "fonctions standard", des fonctions combinatoires ou séquentielles d'usage courant telles que le multiplexage, le décodage, la comparaison, le comptage ou le décalage. Pour ces fonctions il est possible de réutiliser des descriptions VHDL, ou d'utiliser des librairies dans les logiciels EDA où il existe des circuits intégrés spécifiques standard. Il n'est donc pas nécessaire de développer ces fonctions à chaque nouveau projet, mais il suffit de les adapter à nos besoins.

Structures des unités de commande

Bien qu'elle effectue elle-même parfois une partie du traitement de l'information, l'unité de commande a pour rôle principal de générer la séquence de commandes gérant le fonctionnement de l'unité de traitement. L'unité de commande, que nous abrégerons désormais UC, est donc une machine séquentielle appelée à générer une séquence de commandes. Ces commandes sont souvent appelées "instructions", et la séquence est aussi appelée "programme".

Les UC peuvent être séparées en deux types : les UC câblées et les UC microprogrammées. Cette distinction est justifiée par des différences importantes au niveau de la structure, la méthode de conception, les performances et les domaines d'application.

Une UC est dite "câblée" lorsque tout changement de son comportement nécessite un changement de son "câblage", soit des interconnexions entre les circuits qui la composent. Son schéma-bloc général est celui de la figure 1- 4.

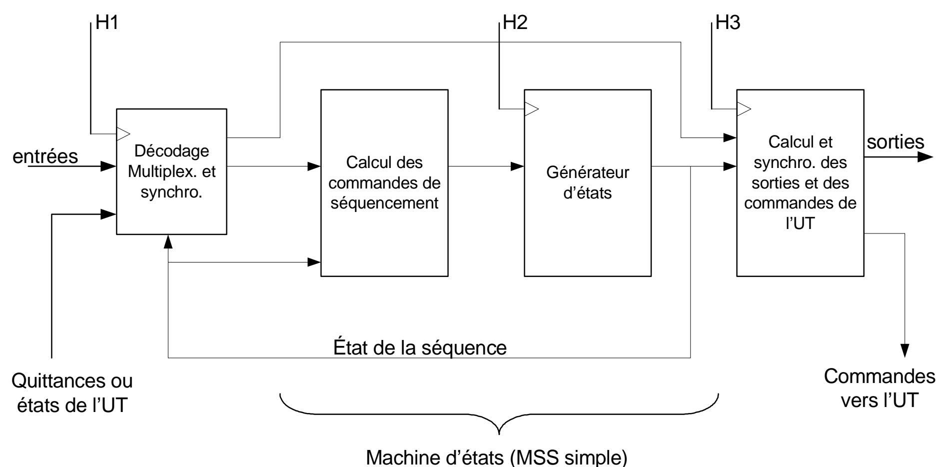


Figure 1- 4 : Schéma bloc général d'une UC câblée

Ce schéma-bloc fait apparaître un découpage et une terminologie plus adaptée aux machines complexes.

Les entrées de l'UC sont des entrées de la machine complexe (pas forcément toutes, certaines peuvent agir directement sur l'unité de traitement) et des quittances ou états provenant de l'unité de traitement, ci-après abrégée UT (voir figure 1- 2). Un premier bloc va procéder aux décodage, multiplexage et synchronisation éventuellement nécessaires. Le décodage permet de séparer les informations en signaux distincts, plus faciles à utiliser que des codes mutli-bits. Le multiplexage permet de réduire le nombre apparent des entrées dans le bloc de calcul des commandes de séquencement. En effet, dans les machines complexes, il est rare que l'évolution depuis un état donné dépende de plus qu'une ou deux des très nombreuses entrées. Il est ainsi avantageux de multiplexer les entrées en fonction de l'état de la séquence.

Le bloc de calcul des commandes de séquencement n'est rien d'autre que ce que nous appelions le décodeur d'état futur. Le nouveau nom vient du fait que ce bloc ne calcul le code de l'état futur que dans le cas où il agit directement sur des flip-flops d'état de type D. En fait, comme le schéma-bloc l'indique, l'état de la séquence est généré par un bloc nommé "générateur d'états", qui peut en effet être simplement un ensemble de flip-flops, mais aussi un compteur, un registre à décalage ou une autre machine séquentielle. Ce que nous appelions un décodeur d'état futur génère donc des commandes qui vont provoquer la génération d'un nouvel état, plutôt que l'état futur lui-même.

Le dernier bloc calcule (et éventuellement synchronise) les commandes de l'UT et des sorties de la machine (pas forcément toutes les sorties, certaines d'entre elles pouvant être générées par l'UT).

Avant l'apparition des PLDs, les UC câblées étaient réalisées à l'aide de circuits spécialisés standard (multiplexeurs, décodeurs, portes, flip-flops, compteurs et registres, essentiellement). Tout changement du câblage impliquait de couper des pistes sur le circuit imprimé et d'ajouter des fils soudés aux pattes des ICs, ou de refaire carrément la conception d'un circuit imprimé. Si un circuit intégré sécifique (ASIC = Application Spécific Integrated Circuit) avait été réalisé, tout changement de comportement (ou la correction d'une erreur) demandait une reconception impliquant des coûts et des délais importants.

Depuis l'apparition des PLDs, le câblage est en bonne partie programmable, ce qui augmente fortement la souplesse des UC câblées. De plus, les logiciels EDA facilitent la conception d'UC câblées de plus en plus complexes. L'utilisation des PLD permet l'intégration d'UC câblée fonc-

tionnant à fréquence élevée. Leur modification est relativement facile et automatisée par l'utilisation des logiciels EDA

Une UC est dite "microprogrammée" lorsque son comportement est essentiellement déterminé par un programme contenu dans une mémoire. Tant que l'on n'augmente pas le nombre d'entrées ou le nombre de sorties, et que le nombre d'instructions composant le programme ne dépasse pas la capacité de la mémoire et du générateur d'états prévus, le comportement peut être modifié par la simple reprogrammation de la mémoire (si elle est effaçable, sinon il faut programmer un autre IC) sans toucher au câblage.

Le schéma-bloc d'une UC microprogrammée apparaît à la figure 1- 5. Ce schéma-bloc ne prétend pas couvrir toutes les structures possibles d'UC microprogrammées, mais il fait apparaître les caractéristiques distinctives d'une UC microprogrammée simple.

Le bloc de décodage-multiplexage-synchronisation des entrées et des quittances de l'UT est similaire à celui d'une UC câblée. Toutefois les commandes de sélection des conditions de séquencement ne proviennent généralement pas directement de l'état de la séquence, mais plutôt de la mémoire du microprogramme.

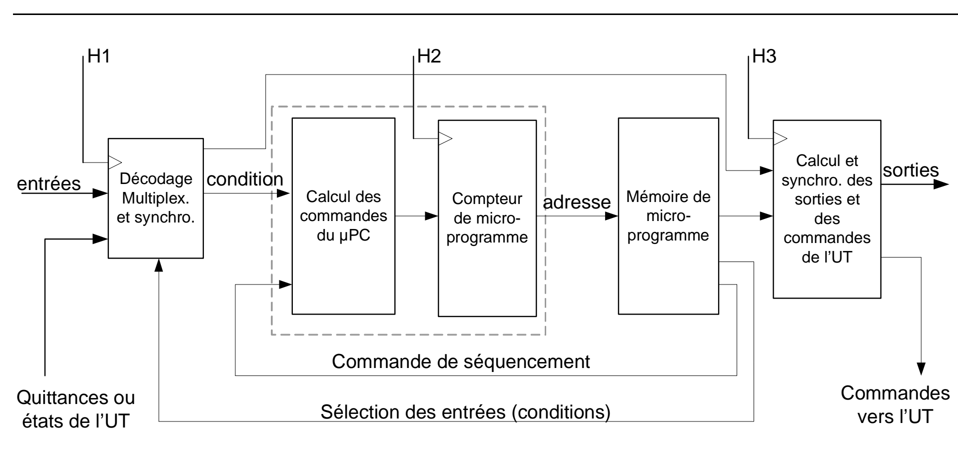


Figure 1- 5 : Schéma bloc d'une UC microprogrammée

Les deux blocs regroupés dans un traitillé constituent ce que l'on appelle un micro-séquenceur. Son rôle est de générer l'adresse de la prochaine microinstruction, en fonction de la commande de séquencement et de l'entrée de condition (il n'y a normalement qu'une entrée de condition, connectée à la sortie du multiplexeur de conditions). Le bloc de calcul des commandes du uPC décode les commandes de séquencement provenant de la microinstruction en cours et en tire les signaux de commande du uPC, tenant comp-

te de l'entrée de condition. Le compteur de microprogramme est ainsi appelé parce que ce bloc, qui mémorise l'état de la séquence, a généralement un fonctionnement correspondant à celui d'un compteur.

Chaque état généré par le microséquenceur est utilisé comme une adresse pour aller chercher dans la mémoire de microprogramme la prochaine microinstruction à exécuter. Les microinstructions se décomposent en commandes de séquencement, généralement codées et comportant quelques bits utilisés pour sélectionner une condition (à travers le MUX des conditions), et en commandes destinées à gérer l'unité de traitement et à générer des sorties. Le bloc de calcul et sychronisation des sorties et des commandes de l'UT est similaire à celui d'une UC câblée.

Structure de l'unité de traitement

Dans une machine complexe décomposée en une unité de commande et une unité de traitement, l'uité de traitement effectue l'essentiel des opérations standard de traitement de l'information. Ces opérations ou fonctions standard peuvent être réalisées à l'aide de circuits spécialisés, que ce soit des circuits spécialisés standard (tels que des compteurs ou multiplexeurs) ou surtout des réseaux logique programmables (PLDs), ou à l'aide de circuits universels effectuant toutes les opérations élémentaires dans lesquelles nous pourrons décomposer n'importe quel traitement. Nous avons ainsi deux types de structures : les UT dites "spécialisées" ou "câblées" et les UT dites "universelles" ou "programmables".

Comme une UC câblée, une UT spécialisée est spécifiquement conçue pour une certaine machine complexe. Tout changement de fonctionnement demande une reconception du circuit et des modifications dans le câblage. Plus le traitement désiré est complexe, plus il comporte de fonctions, et plus volumineuse sera la circuiterie de l'UT.

Depuis l'apparition des PLDs et des outils de CAO associés, des UT performantes et complexes peuvent être réalisées dans une structure spécialisée, tout en conservant une grande souplesse d'adaptation et des dimensions très compactes. L'utilisation d'un langage de description de haut niveau tel le VHDL, permet une très forte réutilisation de fonction ou opérations. Le schéma-bloc de la figure 1- 6 illustre la structure d'une UT spécialisée. Les fonctions standard ou quasi-standard sont commandées individuellement par l'UC, mais peuvent également interagir entre elles. Certaines seront de nature purement combinatoire, d'autres de nature séquentielle.

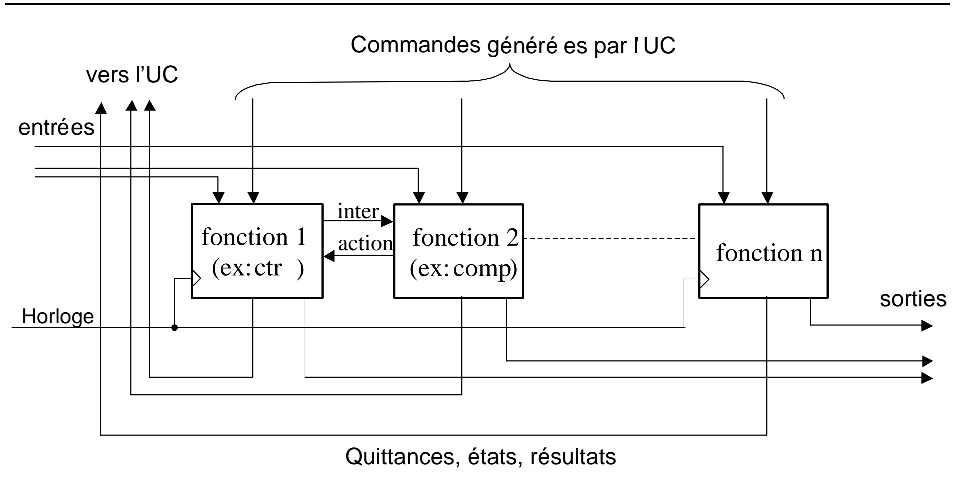


Figure 1- 6 : Schéma bloc de la structure d'une UT spécialisée

Une UT universelle repose sur le fait qu'une fonction combinatoire quelconque peut être décomposée en une suite de fonctions élémentaires (ET, OU et inversion logique, plus l'addition arithmétique), et qu'une fonction séquentielle quelconque peut être obtenue à l'aide d'un registre et d'une fonction combinatoire.

Les diverses fonctions élémentaires combinatoires, arithmétiques ou logiques, sont regroupées dans une unité appelée "unité arithmétique et logique" ou ALU (pour "Arithmetic Logic Unit", en anglais). Cette ALU comporte deux entrées d'opérandes (les informations d'entrée sur lesquelles elle doit effectuer une certaine opération, comme par exemple une addition), une entrée permettant de choisir l'opération à effectuer et une sortie pour le résultat. Chacune de ces entrées et sortie comporte normalement plusieurs bits.

La figure 1- 7 nous montre un schéma-bloc très simplifié d'une UT universelle. Puisque tous les traitements de l'information se font à travers l'ALU, en séquence, c'est l'UC qui est chargé d'aiguiller les résultats de l'ALU vers les sorties concernées de la machine complexe.

Figure 1- 7 : Schéma bloc simplifié d'une UT universelle

Des informations d'entrée et des résultats intermédiaires sont mémorisés dans les registres. En plus du résultat d'une opération arithmétique ou logique, l'ALU fournit également des décodages normalisés qui sont : le report résultant d'une opération arithmétique, le signe du résultat, le fait que le résultat soit nul, et le fait que le résultat requière plus de bits que n'en comporte la sortie de l'ALU (dépassement de capacité). Ces décodages normalisés, appelés bits d'état de l'ALU, sont connectés au multiplexeur de conditions de l'UT.

Comme son nom l'indique, une même unité de traitement universelle peut être utilisée pour effectuer n'importe quel traitement de l'information aussi complexe soit-il. Toutefois, plus le traitement est complexe et plus il prendra de temps, du fait de sa décomposition en une séquence d'opérations élémentaires. Pour modifier le comportement d'une machine complexe utilisant une UT universelle, il suffit de modifier l'unité de commande. Si celle-ci est micropogrammable, une simple modification du microprogramme suffira s'il n'est pas nécessaire d'augmenter la taille de la mémoire de microprogramme, le nombre d'entrées ou le nombre de sorties.

Combinaison des structures d'UC et d'UT

Avec les deux structures de base définies pour les UC d'une part et les UT d'autre part, nous obtenons quatre combinaisons UC+UT possibles:

1)UC câblée et UT spécialisée

-avec circuits discrets (exemple TTL): pratiquement plus utilisé.

-avec PLD, très utilisé depuis l'arrivée de PLD "Low cost".

2)UC microprogrammable et UT spécialisées

3)UC câblée et UT universelles

4)UC microprogrammable et UT universelles

La combinaison UC câblée – UT spécialisée est celle qui permet d'atteindre les plus grandes performances. Son coût, que ce soit le coût de développement ou le coût de fabrication, est quasiment proportionnel à la complexité du traitement de l'information désiré. Cette combinaison sera donc utilisée pour des machines séquentielles de faible à moyenne complexité, ou pour des machines à hautes performances. L'apparition de PLD "Low cost" avec des performances très élevées rend cette combinaison très intéressante. Il est possible d'obtenir des fréquences de fonctionnement très élevées. Dans le cas de systèmes de faible complexité, les coûts peuvent être très compétitifs.

Le coût de développement d'une UC microprogrammée est inférieur à celui d'une UC câblée. Par contre, son coût de production augmente de façon incrémentale avec la complexité. La figure 1- 8 compare les courbes de coût d'une UC câblée et d'une UC microprogrammée.

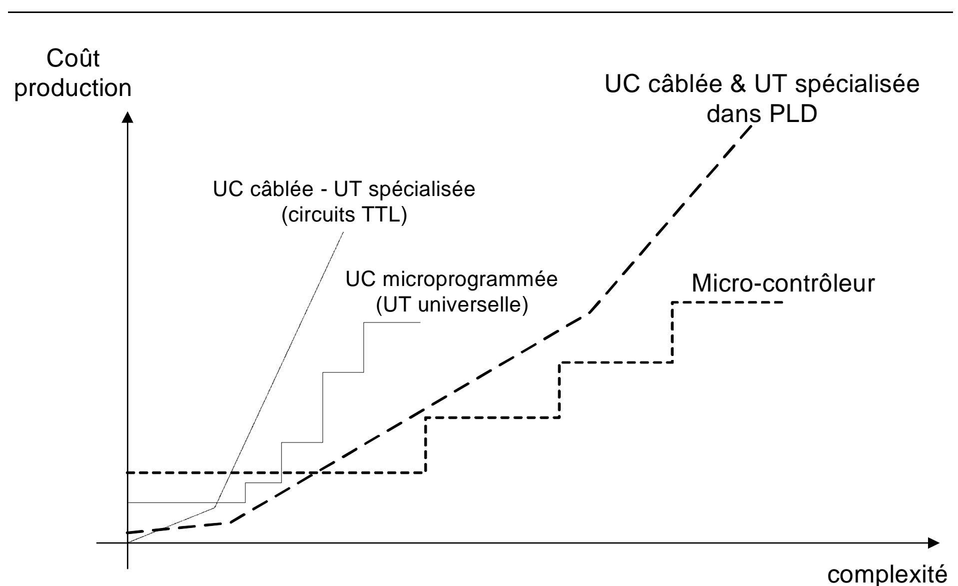


Figure 1- 8 : Évolution du coût de production en fonction de la complexité

Une UC microprogrammée est un système modulaire, ce qui explique la courbe de coût en escalier. La courbe de coût d'une UC câblée comporte aussi des escaliers, mais de plus faible amplitude (non représenté). Dans le début de la courbe, l'augmentation de complexité résulte dans l'utilisation d'un circuit programmable plus complexe. La pente augmente dès qu'il est

nécessaire d'intégrer l'UC câblée dans un PLD de plus grande capacité ou dans plusieurs PLD.

Le point de croisement des deux courbes évolue avec l'apparition de nouveaux circuits et l'évolution du marché. Le point est aussi différent selon la fréquence de fonctionnement.

Le fonctionnement d'une UC microprogrammable sera plus lent que celui d'une UC câblée. La combinaison UC microprogrammable – UT spécialisée convient donc pour les machines d'un niveau de performance et de complexité moyen.

Le coût de développement d'une UT universelle est inférieur à celui d'une UT spécialisée. Par contre, les coûts de production ont des courbes comparables à celles de la figure 1- 8: celle d'une UT spécialisée est similaire à celle d'une UC microprogrammée.

Les performances atteintes à l'aide d'une UT universelle sont faibles, du fait de la décomposition de tout travail en une séquence d'opérations élémentaires. Ainsi, la combinaison d'une UC câblée et d'une UT universelle n'a guère d'intérêt.

En combinant une UC microprogrammée et une UT universelle, nous obtenons un maximum de souplesse et un coût favorable pour les hautes complexités. Les performances seront par contre les plus faibles des quatre combinaisons. C'est pourtant la structure utilisée dans la plupart des microprocesseurs. La combinaison UC microprogrammée avec une UT universelle correspond à la structure d’un processeur. Celui-ci comprend une unité de séquencement qui traite les instructions du programme. Le processeur est capable de faire des opérations qui sont réalisées dans l’ALU(Arithmetic Logic Unit) interne à celuil-ci. Il s’agit d’une UT universelle.

La disponibilité de PLDs de grande capacité permet l’intégration d’un processeur complet. Il existe actuellement des descriptions, en langage HDL (tel le VHDL), fournie par les vendeurs de PLD. L’UC pourra donc être remplacée par un processeur avec un programme écrit en langage assembleur, voir même de haut niveau (exemple "C"). Il s’agit d’une nouvelle possibilité de réaliser des MSS complexes.

Chapitre 2

Méthode de conception

Le fonctionnement d'une machine séquentielle conçue en séparant UC et UT résulte d'un algorithme implémenté par l'UC. De telles machines sont donc appelées "machines algorithmiques" (ou ASM, pour "Algorithmic State Machine" en anglais). L'établissement d'un algorithme de fonctionnement va être l'étape la plus importante dans la conception d'une machine algorithmique.

La méthode de conception est de type descendant ("top-down"): on commence par une approche globale, pour aborder progressivement des détails de plus en plus fins. Les étapes principales de la méthode sont listés ci-dessous, par ordre chronologique.

1) Établir les spécifications ou, si elles sont déjà données, les étudier d'un oeil critique pour s'assurer qu'elles sont complètes, qu'elles correspondent bien aux besoins à couvrir, et qu'elles n'introduisent pas des contraintes superflues.

2) Définir les relations temporelles désirées liant les entrées et les sorties de notre machine, ceci à l'aide de chronogrammes. Prendre aussi note de toute les contraintes spéciales de fréquence, débit d'information ou temps de réponse.

3) Décrire le système dans lequel s'insère notre machine sous la forme d'un schéma-bloc grossier faisant apparaître les échanges d'information, et le comportement de notre machine sous la forme d'une algorithme général. Celui-ci décrira le comportement à l'aide de phrase, texte...

4) Détailler progressivement l'algorithme de fonctionnement, de façon à faire apparaître des fonctions standard ou d'autres sous-machines. Durant cette phase, l'organigramme sera affiné, évolué.

5) Choisir la structure de l'unité de traitement la mieux appropriée, étant donnée les fonctions standard apparaissant dans l'algorithme, le niveau global de complexité, les performances à atteindre (définies en particulier au point 4), le temps de développement et le coût visé. Établir un schéma-bloc donnant les fonctions principales de cette unité de traitement.

6) Choisir la structure la mieux appropriée pour l'unité de commande, en tenant compte de la taille et de la complexité de l'algorithme à implémenter, de la vitesse de fonctionnement, du temps de développement et des impératifs économiques. Établir un schéma-bloc de cette unité de commande, faisant clairement apparaître toutes ses entrées et toutes ses sorties.

7) Choisir les principaux composants (circuits intégrés) à utiliser, aussi bien pour l'UC que pour l'UT, et vérifier que les objectifs visés sont atteignables.

8) Finir la conception de l'UT aboutissant à des schémas et des fichiers de description VHDL pour les PLDs. Tester l'UT.

9) Finir la conception de l'UC, aboutissant à des schémas, des fichiers de description VHDL pour les PLDs et des fichiers de programmation pour les mémoires. Tester l'UC, puis la machine complète.

L'étape 1 relève d'avantage de la conduite de projets que de l'électronique numérique et ne sera donc pas étudiée dans le cadre de ce cours. Les autres étapes seront traitées à l'aide d'exemples dans ce qui suit. Les choix

de structure dans les étapes 5 et 6 demandent de bonnes connaissances des circuits disponibles et seront donc laissés à l'ingénieur expérimenté.

Le test, en particulier le test de fabrication, est un gros problème en soi. Bien qu'il ait une influence importante sur la conception d'une machine, nous le traiterons séparément au chapitre IX.

Jusqu'au point 7 y-compris, la méthode ci-dessus est aussi valable pour le développement de machines basées sur un microprocesseur. En fait, si nous avons choisi une UT universelle et une UC microprogrammée, nous utiliserons plutôt un système à microprocesseur. Dans ce cas, les dernières étapes sont l'établissement du schéma puis l'écriture d'un programme.

Chapitre 3

Application de la méthodologie de conception à la multiplication

Pour présenter la démarche de conception d'une machine séquentielle complexe, nous allons utiliser la multiplication binaire. Pour cet exemple, nous allons utiliser 2 nombres entiers non-signés de 4 bits. Le résultat sera donné sur 8 bits. Nous allons nous appuyer sur l'étude faite, dans le chapitre précédent, de la multiplication non-signée.

3-1 Spécification de la multiplication séquentielle

Nous allons utiliser une multiplication avec une décomposition séquentielle. Cette exemple permettra de mettre en évidence les tâches gérées par l'UC et les opérations réalisées par l'UT.

Voici la spécification du fonctionnement de la multiplication séquentielle, soit:

-Lors d'une transition de '0' à '1' du signal START, le calcul de la multiplication est démarré. À cet instant, les valeurs du Multiplicateur et du

Multiplicande sont disponibles en entrée. Le calcul utilisera le principe de la décomposition temporelle.

-Lorsque la multiplication est terminée, le signal Done est activé. Le résultat est disponible tant que Done reste actif.

Schéma-bloc général

Les étapes 1 et 2 de la méthode de conception ont été traité dans le chapitre qui précède.

Pour notre exemple, nous pouvons donc passer à l'étape 3 et établir un schéma-bloc général. Celui-ci est représenté à la figure 3- 1. Le système de commande que nous devons développer apparaît comme un seul bloc, le but étant de montrer comment il est relié aux autres parties de l'appareil. Tout découpage plus détaillé serait prématuré, et ne pourrait résulter que d'à-prioris qu'il faut absolument éviter pour être en mesure par la suite de faire librement les meilleurs choix.

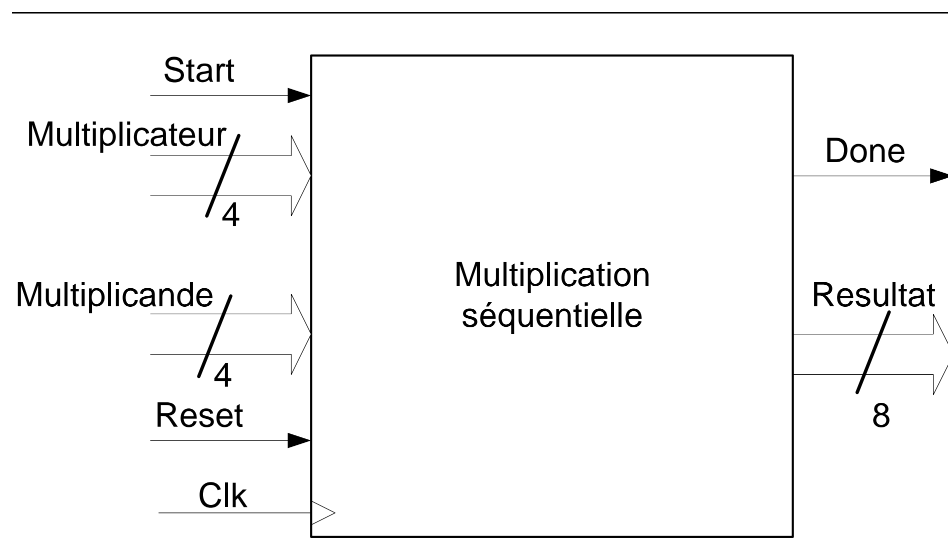


Figure 3- 1 : Schéma-bloc général de la multiplication séquentielle

Le chronogramme de la figure 3- 2 nous donne le comportement des signaux Start et Done du système.

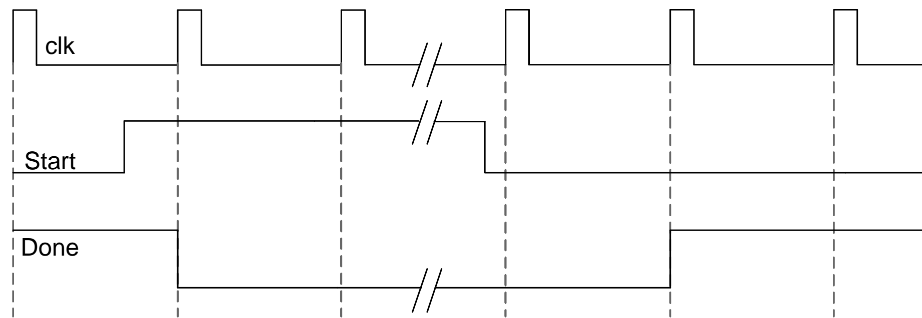


Figure 3- 2 : Chronogramme du fonctionnement

Algorithme

La première phase de la conception consiste à établir un algorithme répondant au cahier des charges. Cet algorithme a été étudié lors du chapitre sur la numération et l'arithmétique. Voici celui-ci:

```
Resultat_Haut := 0;

for I=1 to Nombre_Bits
   if LSB(Multiplicateur) = 1 then
      Resultat_Haut := Resultat_Haut + Multiplicande;
      (mémorisation du report dans un flip-flop)
   else
      Resultat_Haut := Resultat_Haut+0;
   end if

   Décalage à droite(Report, Resultat_Haut, Multiplica-
                                                  teur);
end for;

Resultat_Bas := Multiplicateur;
Resultat := Resultat_Haut & Resultat_Bas;
```

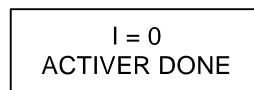
3-2 Algorithme : organigramme ou graphe

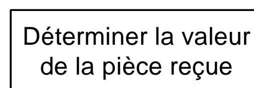
Lors de la conception des MSS simples, nous avons utilisé un graphe d'états pour décrire le comportement d'une machine séquentielle. En l'adaptant aux exigences liées au grand nombre d'entrées, de sorties et d'états, nous pouvons également l'utiliser pour décrire le comportement d'unités de commande câblées, dans les machines séquentielles complexes. Toutefois, un graphe d'états n'est guère pratique pour décrire un algorithme de fonctionnement général.

Dans ce qui suit, nous allons utiliser une autre forme de représentation graphique, appelée "organigramme" ("flow-chart" ou "flow diagram", en anglais), qui offre l'avantage d'être utilisable, non seulement pour décrire l'algorithme de fonctionnement général, mais aussi pour décrire une réalisation à l'aide d'un microprocesseur ou d'une unité de commande microprogrammée. Contrairement à un graphe d'états, un organigramme met l'accent sur la séquence des actions qu'il faut commander, plutôt que sur une séquence d'états. Une action pourra par la suite être une suite d'états ou nécessiter du matériel dans l'UT.

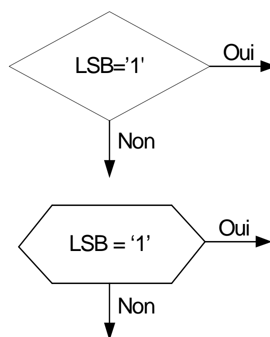
Un organigramme utilise les symboles graphiques suivants:

Une flèche, généralement pointant de haut en bas, illustre le passage d'une action à la suivante.





Une boîte rectangulaire désigne un ensemble d'actions ou opérations s'effectuant simultanément; par exemple : Initialisation de la variable I à zéro et activation du signal Done.



Une boîte en forme de losange ou d'hexagone désigne une condition binaire qui modifie le déroulement des opérations, selon que cette condition est remplie ou non

Nous allons débuter par représenter l'algorithme sous forme d'un organigramme. À partir de celui-ci, il est possible de réaliser la partition des tâches entre l'unité de traitement et l'unité de commande. Le premier organigramme sera nommé organigramme grossier de la multiplication séquentielle. Il ne comprend, dans les boîtes, que des **textes/actions ou fonctions générales**.

Nous allons maintenant présenter la construction de l'organigramme pour notre exemple.

Lors du démarrage de tout système séquentiel, une remise à zéro est indispensable. Dans l'organigramme grossier, l'action d'un Reset sera visualisée par une bulle arrondie avec RAZ.

Le premier élément de l'organigramme est le test du flanc montant du signal start. Dès que le flanc a été détecté il faut initialiser les différentes valeurs. Il faut affecter aux variables Multiplicateur et Multiplicande les valeurs fournies en entrée. Resultat_haut est initialisé à "0000". La variable de boucle i est initialisée à 1 et le signal de sortie Done devient inactif.

Ensuite, le teste de l'état du bit LSB de Multiplicateur détermine s'il faut effectuer l'addition de Resultat_haut avec le Multiplicande ou maintenir la valeur de Resultat_haut. Cette dernière opération est équivalente à additionner 0. L'opération suivante décale Resultat_haut et Multiplicateur sur la droite.

Il faut incrémenter le compteur de boucle afin de comptabiliser le traitement d'un bit du multiplicateur. Il faut ensuite tester si la variable de boucle i a atteint la valeur nombre_bits (dans notre cas 4). Si cette valeur est atteinte il faut concaténer Resultat_haut avec Resultat_bas. Resultat_bas et Multiplicateur sont contenu par la même variable. Dans le cas où la variable i n'a pas atteint 4, retourner au test du LSB de Multiplicateur.

On termine le cycle en activant le signal Done qui indique que la multiplication est terminée. Pour recommencer l'opération il faut attendre un nouveau flanc montant sur le signal Start.

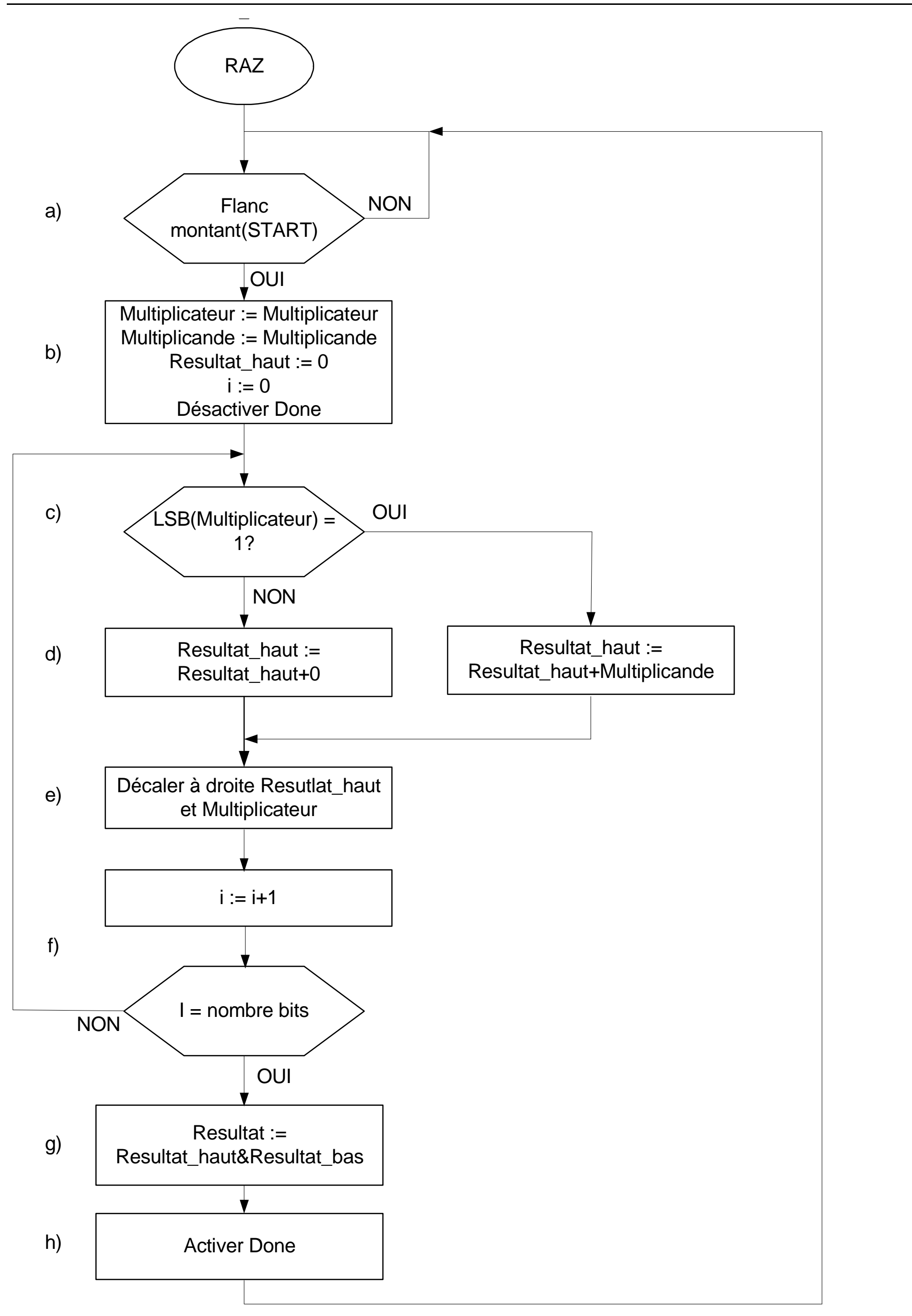


Figure 3- 3 : Organigramme grossier de la multiplication séquentielle

3-3 Partition unité de commande / unité de traitement

La décomposition d'une machine complexe en une unité de commande et une unité de traitement est l'élément clé de notre méthode de conception. Cette décomposition laisse une grande place à l'imagination, à l'expérience et au savoir faire du concepteur, relevant donc plus de l'art que de la science. Lors de chaque choix que nous aurons à faire, il ne faut jamais perdre de vue que toute fonction combinatoire, de la plus simple à la plus complexe, peut être décomposée en une séquence. Par exemple, le bout d'organigramme de la figure 3- 4 réalise une fonction ET à 3 entrées.

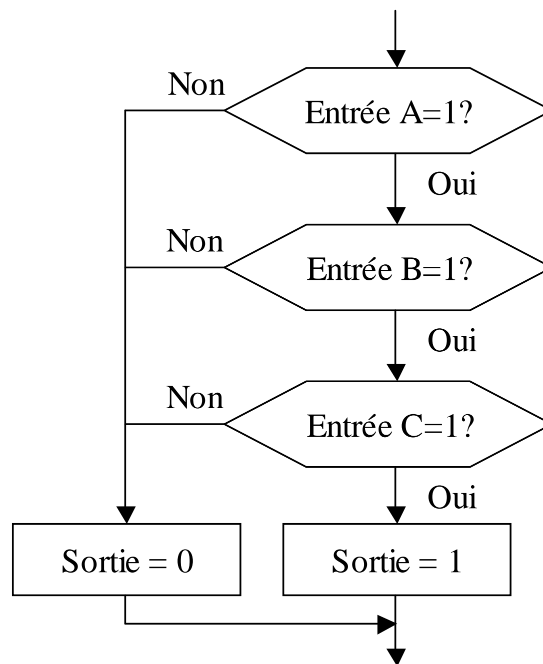


Figure 3- 4 : Organigramme de la fonction logique ET à 3 entrées

En utilisant les annotations visibles sur l'organigramme grossier de notre exemple (figure 3- 3), nous allons identifier les fonctions à effectuer dans l'unité de traitement ou dans l'unité de commande. Nous allons traiter les boîtes les unes après les autres.

a) La détection d’un flanc sera réalisée en testant l’état de l’entrée Start dans l’UC. Il n’y a pas besoin de matériel dans L’UT.

b) Ce bloc comporte l'initialisation des différentes valeurs utilisées dans la multiplication. L'équivalent d'une variable dans l'UT est un registre. Multiplicateur (Resultat_bas), Multiplicande et Resultat_haut seront donc contenus dans des registres. Ceux-ci seront placés dans l'UT. Ils n'interviennent pas dans le séquencement. Nous choisissons d'intégrer la variable de boucle dans l'UC. Il s'agit de répéter 4 fois l'opération de traitement d'un bit du Multiplicateur.

c) La valeur de Multiplicateur (Resultat_bas) se trouve dans l'UT. Pour tester le bit LSB, il faut relier celui-ci à l'UC, nous appellerons ce signal LSB_Mteur.

d) La réalisation de l'addition sera commandée par l'activation d'un chargement du résultat de l'addition. Selon le résultat du test de LSB (voir point b), on additionne Resultat_haut avec Multiplicande ou avec zero, correspondant aux deux chemins définis dans l'organigramme.

Remarque: Il est à noter que le signal du chargement du registre contenant Resultat_haut devra être distingué des autres registres (Load). Ces signaux proviennent de l'UC.

e) Pour décaler ces deux valeurs, nous allons utiliser des registres à décalage. Ces registres se trouvent dans l'UT et sont commandés par un signal provenant de l'UC (Decale).

f) Afin d'éliminer un compteur de boucle, nous avons choisi de dérouler la boucle de l'organigramme. Ceci implique de répéter 4 fois la séquence des opérations pour chaque bit du Multiplicateur dans l'organigramme.

g) L'UT doit fournir le résultat final de la mltiplication. Il faut simplement concaténer les deux registres Resultat_haut et Resultat_bas. Il s'agit de simple fils!.

h) Cette sortie sera activée directement par l'UC. L'activation de ce signal indique que le résultat final est valide. Nous regroupons ces deux dernières boites qui sont liées.

Suite à la partition, nous obtenons l'organigramme évolué (figure 3- 5) représentant la suite d'opérations gérées par l'UC.

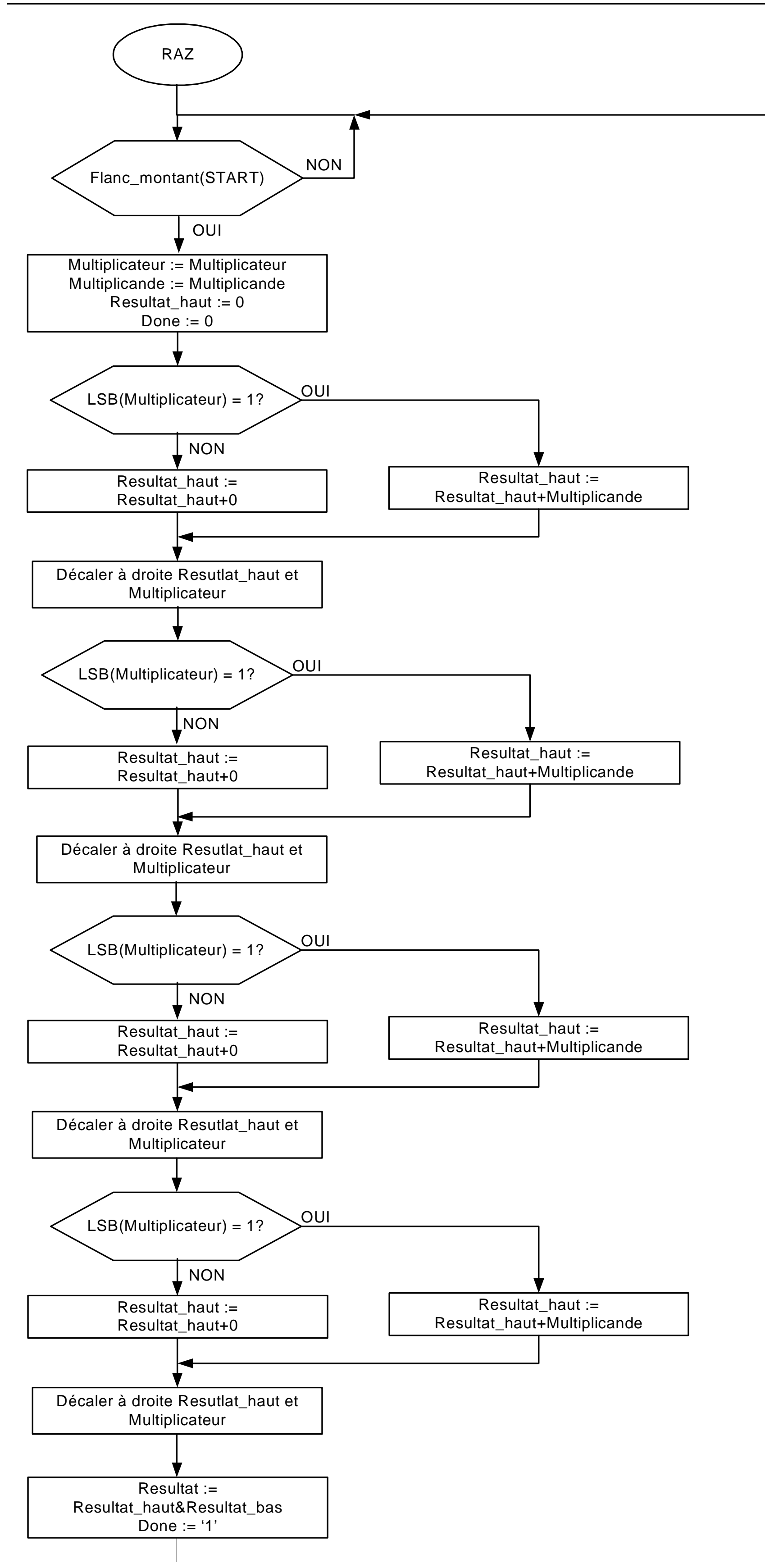


Figure 3- 5 : Organigramme évolué

Identification des fonctionnalités de l'UT

Le partitionnement UC/UT détermine les fonctions qui se trouveront dans dans l'UT. Il faut maintenant identifier l'ensemble des fonctions à implémenter dans l'UT.

L'UT comportera un additionneur commandé par le signal QplusP_nQplus0 afin d'additionner Resultat_haut avec Multiplicande ou avec zero. Les deux entrées P et Q comportent chacune 4 bits. Il les additionne de manière asynchrone, le résultat comporte 4 bits ainsi qu'un bit de report pour le dépassement.

Le décalage étant effectué après le chargement du résultat dans le registre contenant Resultat_haut, il faut s'assurer que le bit de report soit encore disponible au moment du décalage. Mais, dès que le registre contenant Resultat_haut a enregistré le résultat de l'addition, sa sortie change. Donc les valeurs de sorties de l'additionneur changent. La valeur du report n'est plus la même. Il faut donc mémoriser celui-ci dans une bascule (DFF) pour l'opération de décalage exécutée lors du cycle suivant.

Les 4 bits du Resultat_haut sont stockés dans un registre à décalage de 4 bits. Ce registre s'appelle Result_H. Le bit entrant provient de la bascule contenant le report de l'additionneur. Et le bit sortant est repris par Result_B_Mteur. Sa sortie de 4 bits est reliée à une des entrées de l'additionneur. Le registre charge le résultat de l'addition lorsque le signal Load_ResH est active. Le décalage de la valeur est activé par le signal Decale.

Lors de l'initialisation, il faut charger Result_H avec la valeur "0000". Mais son entrée doit aussi charger le résultat de l'addition. C'est pour cette raison qu'il faut utiliser un multiplexeur,il sélectionne la valeur "0000" uniquement à l'initialisation. Le multiplexeur est commandé par le signal Init_nAdd.

Les 4 bits de Resultat_bas (Multiplicateur) sont stockés dans un registre à décalage de 4 bits. Ce registre s'appel Result_B_Mteur. Il est identique au registre Result_H. Le bit entrant provient du registre à décalage de Result_H. Le bit sortant est perdu. Le contenu de ce registre est chargé à l'initialisation avec la valeur de Multiplicateur fournie au système. Le décalage de la valeur est activé par le signal Decale et le chargement de la valeur par le signal Load (actif lors de la remise à zéro du système). Le contenu du registre représente les 4 bits de poids faible du résultat final.

Les 4 bits de Multiplicande sont contenu dans un registre qui se nomme Mcande. Ce registre est chargé une fois à l'initialisation avec la valeur Multiplicande fournie au système. La sortie du registre est reliée à une des entrée de l'additionneur.

Pour concaténer les 4 bits de Result_H avec les 4 bits de Result_B_Mteur, il suffit de les regrouper afin de former les 8 bits de Resultat.

Unité de traitment

En résumé l'unité de traitement comporte les composants suivants:

- 2 registres, Result_H et Result_B_Mteur, avec chargement parallèle et décalage à droite
- 1 registre, Mcande, avec chargement parallèle
- 1 additionneur de 4 bits avec report
- 1 multiplexeur 2 à 1 sur 4 bits pour sélectionner la valeur de chargement pour le registre Result_H
- 1 bascule DFF pour mémoriser le report

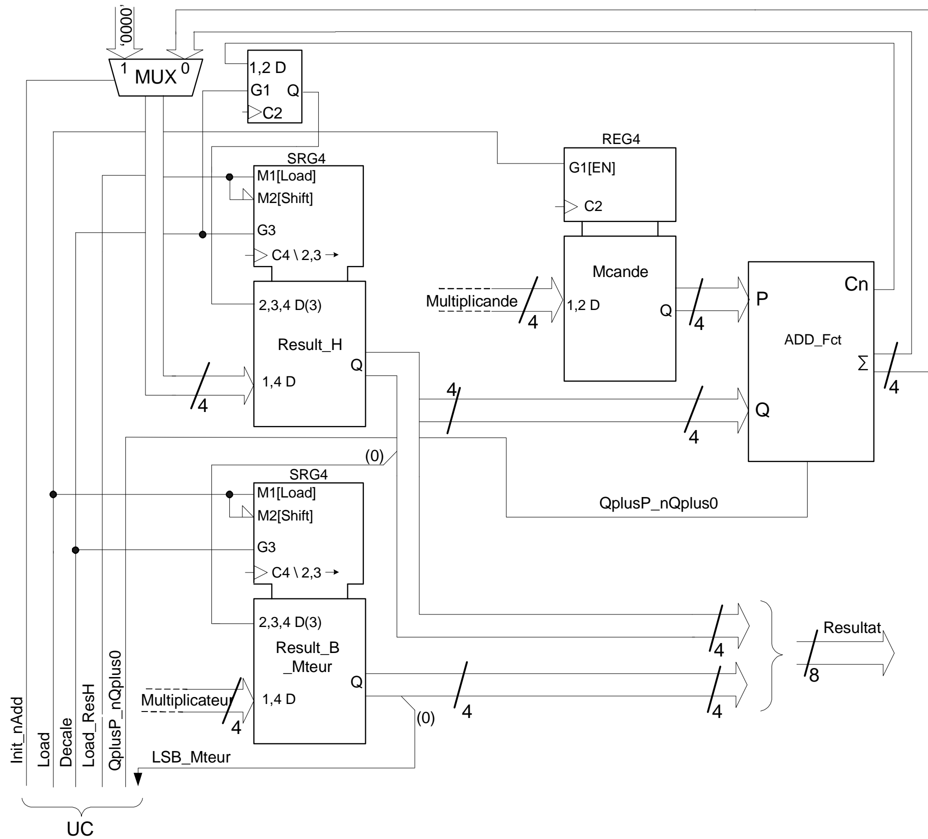


Figure 3- 6 : Unité de traitement

Unité de commande

L'unité de commande gère les signaux de contrôle de l'UT. Il s'agit des signaux Load, Load_ResH, Decale et LSB_Mteur. Elle gère aussi les signaux de sortie du système soit Done. Les signaux d'entrée Start et nReset aggissent aussi sur l'UC. La figure 3- 7 nous montre l'ensemble des signux connectés sur l'UC.

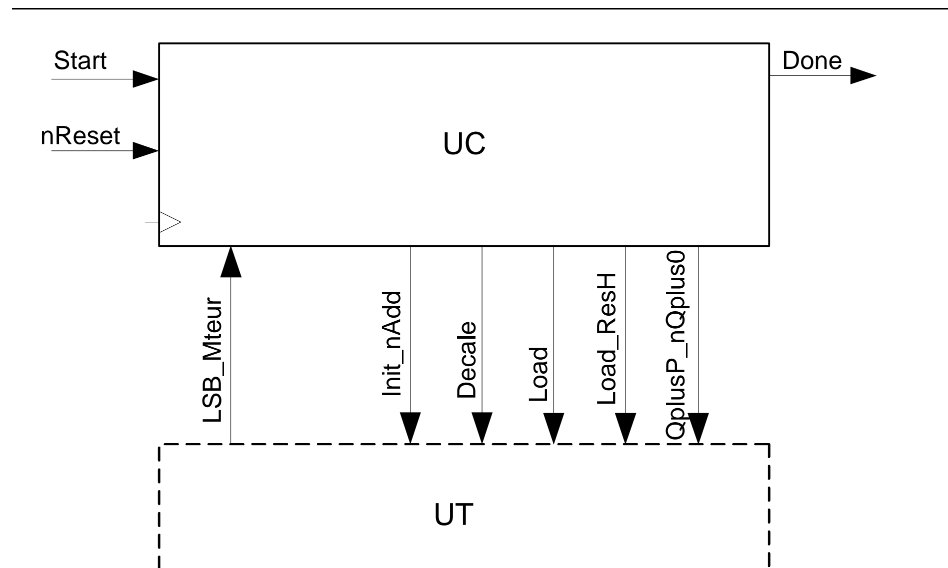


Figure 3- 7 : Unité de commande

3-4 Passage d'un organigramme à un graphe

Si nous détaillons suffisamment un organigramme décrivant le comportement d'une machine séquentielle, nous arriverons à ce que chaque boîte d'action ne contienne que des actions pouvant être effectuées dans une seule et même période d'horloge. Dans ce cas, chaque boîte d'actions correspond à un état interne de la machine séquentielle, soit une bulle dans le graphe d'états.

Intuitivement, on comprend aisément qu'il y a une parfaite correspondance entre un organigramme et un graphe d'états. Sans nous attarder à des démonstrations oiseuses, nous utiliserons les règles suivantes:

règle n°1: Une boîte d'actions dans un organigramme détaillé définit un état dans le graphe. Une état commence au début d'une boîte d'actions et se termine à l'entrée de la boîte d'actions suivante, quel que soit le chemin parcouru à travers des boîtes de condition. La figure 3- 8 donne des exemples de passage d'un organigramme à un graphe. Ll'extrait d'organigramme de la figure 3- 8(c) contient quatre états: (a), (b), (c) et (d), l'état (a) englobant toutes les boîtes de condition.

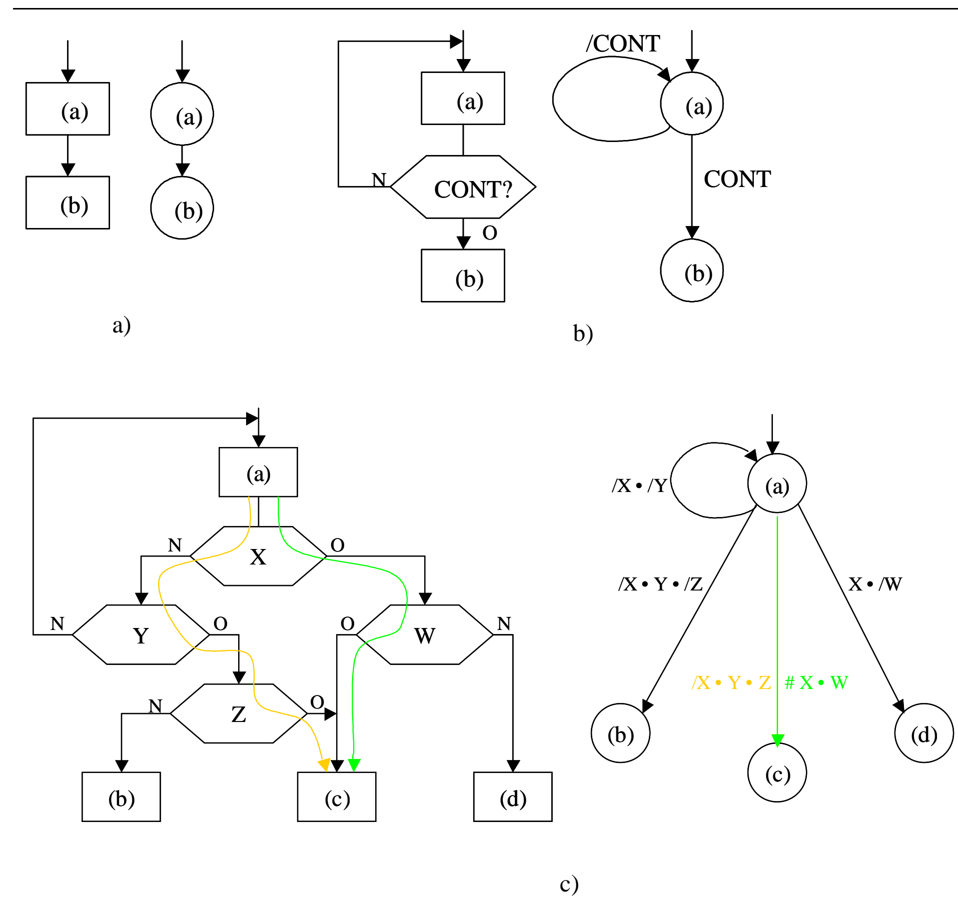


Figure 3- 8 : Passage d'un organigramme à un graphe

règle n°2: Les conditions d'entrée qui font passer d'un état à un autre dans un graphe peuvent être retrouvées dans un organigramme en suivant les chemins possibles à travers les boîtes de condition entre deux boîtes d'actions.

La figure 3- 8(c) met en évidence les différents chemins qui mènent de l'état (a) aux états (b), (c) ou (d), ainsi que les différentes conditions dont dépendent ces transitions.

Dans une MSS ayant beaucoup de variables d'entrée (c'est le cas si cette MSS est une unité de commande d'une machine complexe), il est évident qu'il ne faut pas mettre une flèche dans le graphe pour chaque combinaison des variables d'entrée, à partir de chaque état. Désormais, nous reporterons sur une flèche non pas une combinaison des variables d'entrée, mais l'ex-

pression logique représentant la condition (complexe) qui entraîne cette transition. Cela nous permettra de ne représenter qu'une seule flèche pour toutes les possibilités de passer d'un état (a) à un état (b), où (b) est l'état futur de (a).

Ainsi que nous l'avons vu lors du chapitre sur les MSS simples, les entrées asynchrones (qui ne changent pas en synchronisme avec l'horloge) peuvent nous poser des problèmes. Il est dès lors judicieux de les mettre en évidence, en les signalant par un astérisque ('*'), par exemple.

règle n°3: Les tests portant sur une entrée asynchrone seront signalés dans un organigramme à l'aide d'un astérisque placé dans la boîte de condition correspondante. De même, dans un graphe un état dont l'évolution dépend d'entrées asynchrones sera signalé par un astérisque.

règle n°4: Les changements d'état des sorties, ou l'activation des sorties doivent être indiqués par des actions dans l'organigramme. Nous distinguerons les sorties ou actions inconditionnelles, c'est-à-dire ne dépendant que de l'état interne du séquenceur (rappel : séquenceur = UC), des sorties conditionnelles, c'est-à-dire dépendant également d'une ou plusieurs entrées.

Dans un organigramme, les sorties conditionnelles apparaissent dans des boîtes d'action de forme oblongue, alors que les sorties inconditionnelles se trouvent dans des boîtes rectangulaires, comme nous le montre la figure 3- 9 où la sortie RUN dépend de l'entrée GO.

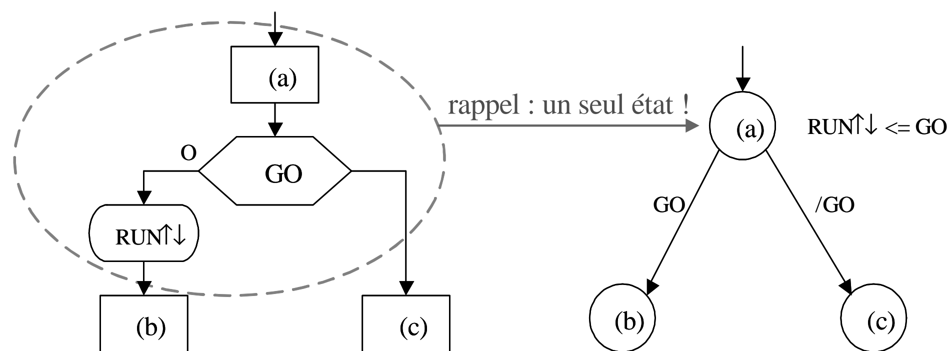


Figure 3- 9 : Mise en évidence des sorties conditionnelles et inconditionnelles

Une sortie conditionnelle ne définit pas un nouvel état. Dans l'exemple de la figure précédente, la sortie RUN est activée pendant l'état (a) lorsque l'entrée GO est active, d'où la notation utilisée dans le graphe: la sortie est indiquée à côté de l'état dans lequel elle sera active, à l'aide d'une expression partielle, soit RUN ??= (a) & GO dans cet exemple.

Il arrive fréquemment qu'une sortie doive être mise à l'état actif lorsque l'UC est dans un certain état de la séquence, puis soit maintenue à l'état actif jusqu'à ce que l'UC atteigne un certain autre état. C'est la raison pour la-

quelle une action a lieu: 99?? pour une impulsion à l'état actif, ??pour une impulsion à l'état inactif, ?pour l'activation, ?pour la désactivation. Lorsque l'activation et la désactivation d'une sortie ne se font pas dans le même état de l'UC, il faut soit maintenir celle-ci dans tous les états intermédiaires soit conserver la valeur de la sortie, entre deux changements, à l'aide d'une bascule (UT).

Remarque importante

Il faut éviter de créer des sorties conditionnelles dépendant d'une entrée asynchrone, à cause de la possibilité d'avoir une impulsion de sortie de durée infinitésimale. Si cela ne peut pas être évité, il faut s'assurer que l'état actif ou inactif de l'entrée dure plus d'une période d'horloge (afin qu'elle soit vue dans cet état lors du flanc actif de l'horloge) et introduire un état supplémentaire pendant lequel cette impulsion sera prolongé. Ainsi, l'organigramme de la figure 3- 10 (a) doit être adapté comme l'indique la figure 3- 10 (b), d'où le graphe de la figure 3- 10 (c). Les codes des états (a) et (a') doivent bien sur être adjacents.

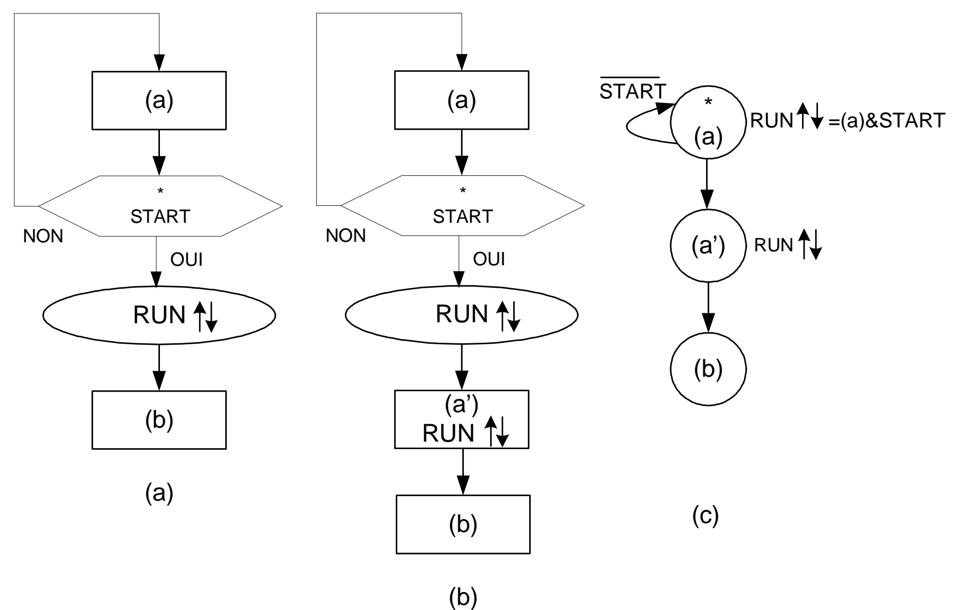


Figure 3- 10 : Prolongation de la durée d'une impulsion

Le problème est le même lorsque ce n'est pas le début mais la fin d'une impulsion de sortie qui dépend d'une entrée asynchrone. Dans ce cas, on ajoutera l'état supplémentaire avant l'état dans lequel l'impulsion se termine. Cette situation est illustrée par la figure 3- 11.

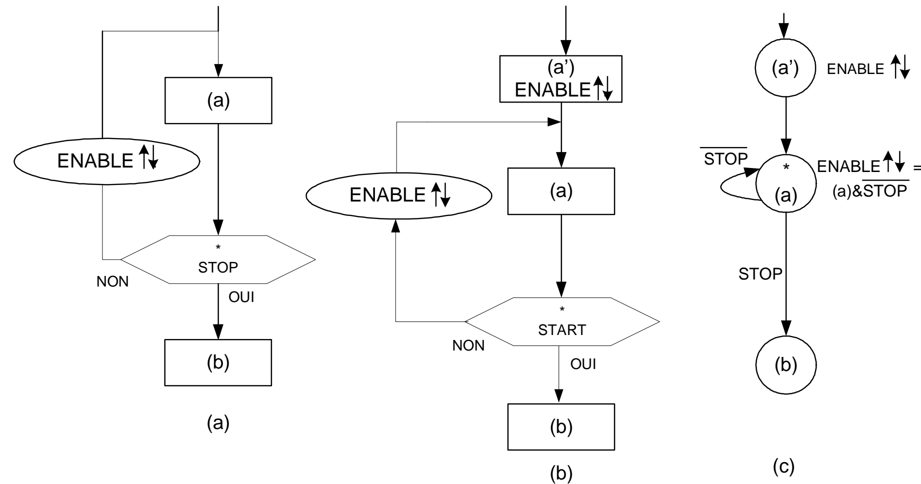


Figure 3- 11 : Ajout d’un état supplémentaire

Lorsqu’une boîte de test est rebouclée sur elle-même, il est nécessaire de rajouter un état dans la boucle.

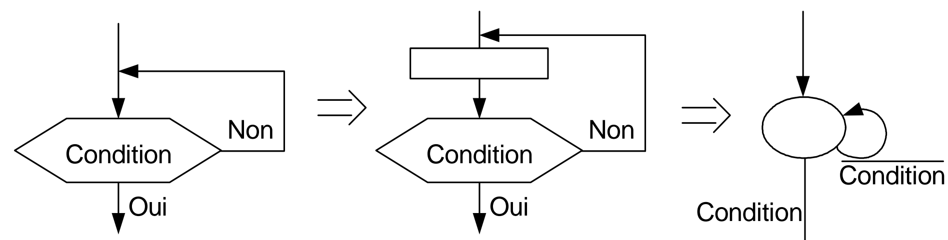


Figure 3- 12 : Adaptation d’une boîte de test rebouclée

3-5 Organigramme détaillé

En tenant compte de l'unité de traitement et de l'organigramme de l'UC de la figure 3- 11, il est maintenant possible d'établir un organigramme détaillé. L'organigramme détaillé représente l'organigramme grossier en intégrant la partition choisie pour l'UT et l'UC. Cet organigramme contient uniquement les signaux connectés sur l'UC. Nous allons présenter l’évolution de l’organigramme de la figure 3- 5 vers l’organigramme détaillé pour notre exemple de multiplication séquentielle.

Le système est activé lorsqu'il y a un flanc montant sur le signal Start. La détection d’un flanc montant est réalisée en testant le signal Start à l'état bas puis à l'état haut. Le système peut-être démarré à la suite de ces deux tests.

Il est nécessaire de décomposer cette opération pour l’organigramme détaillé (figure 3- 13).

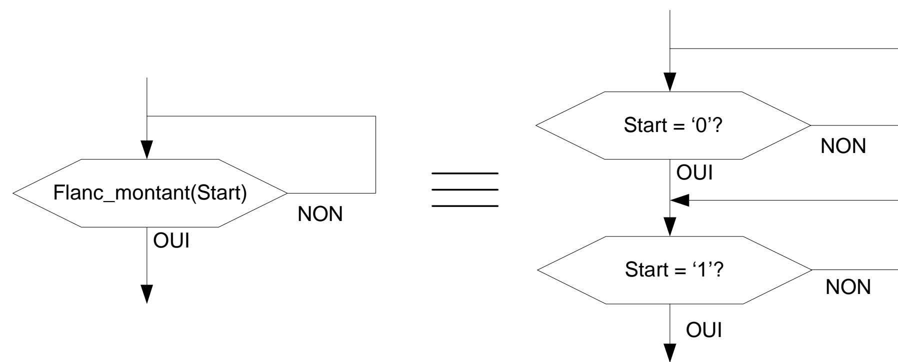


Figure 3- 13 : détection du flanc montant. Organigramme grossier et détaillé

Le chargement du Multiplicateur et du Multiplicande s'effectuent en activant le signal Load. Le résultat est initialisé à zéro en mettant le registre Result_H à zéro. Cette initialisation est réalisée en activant le signal Load_ResH tout en ayant le signal Init_nAdd actif (’1’). Le résultat de l'addition est chargé dans le Result_H en activant le signal Load_ResH. Le signal Init_nAdd doit être inactif (’0’).

Le décalage à droite de Result_H et de Result_B_Mteur est effectué en activant le signal Decale.

Dans l'organigramme général, l'avant dernière opération consiste à concaténer Result_H avec Result_B. Dans l'organigramme détaillé, cette opération disparaît car cela est fait par l’UT (câblage).

La figure 3- 14 présente l’organigramme détaillé pour notre exemple.

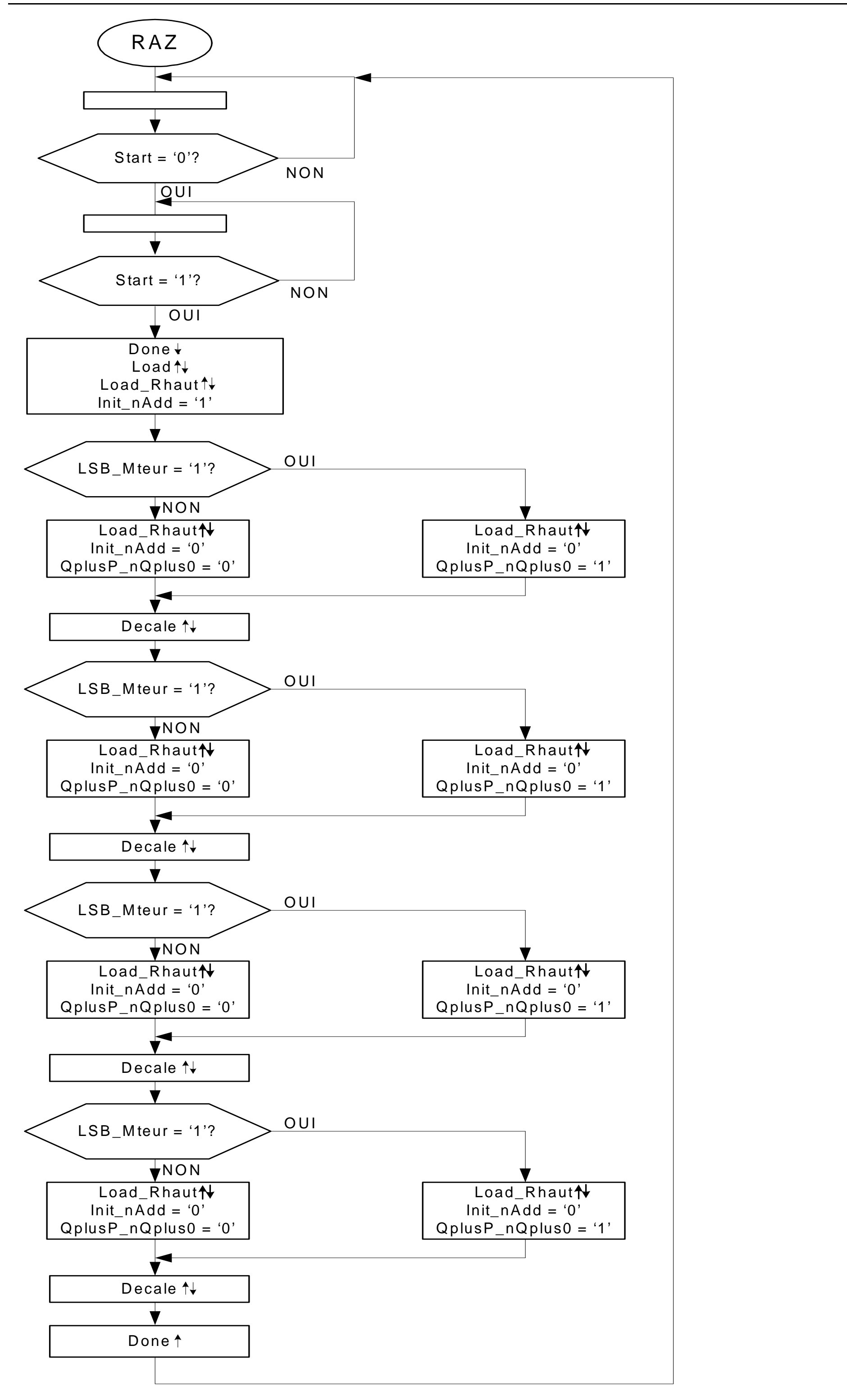


Figure 3- 14 : Organigramme détaillé de la multiplication séquentielle

Remarque: Le signal Init_nAdd est utilisé seulement lorsque Load_Rhaut est activé. Nous avons seulement indiqué le niveau requis pour les états concernés. Pour tous les autres états de l'organigramme le signal Init_nAdd n'est pas spécifié (cas '-').

3-6 UC câblée

Une UC câblée peut être développée avec les méthodes étudiées précédemment pour autant qu'elle soit d'un niveau de complexité compatible avec les PLDs et les outils de CAO dont nous disposons. Avec les FPGA (réseaux matriciels), le niveau de complexité maximal est très élevé actuellement, et il augmente chaque année.

Dans un organigramme détaillé, chaque boîte d'action ne contienne que des actions pouvant-être effectuées dans une seule et même période d'horloge. Dans ce cas, chaque boîte d'action correspond à un état interne de la machine séquentielle, donc à une bulle dans le graphe d'états.

Il faut être attentif au fait qu'il n'est pas possible de tester un signal durant le même cycle d'horloge nécessaire à son établissement. Car durant ce cycle, l'état est en train de se modifier. Si on test cette valeur, on obtiendra une valeur erroné. Il faut donc attendre un cycle pour tester un signal.

Prenons l'exemple d'un décompteur 4 bits. Nous souhaitons décompter celui-ci tant qu'il n'a pas atteint la valeur zéro. Sur le chronogramme de la figure 3- 15, nous observons un fonctionnement incorrecte du décomptage. Lorsque nous sommes dans l'état suite, le compteur est finalement à l'état 15 et non 0 comme souhaité.

Sur la figure 3- 15, la condition de sortie de l'état Dec est pourtant bien Cpt=0. Lorsque cette condition devient active, nous passons, au prochain flan montant de l'horloge, dans l'état suite. Mais simultanément, le signal Dec_Cpt est encore actif et le compteur est décrémenté encore une fois.

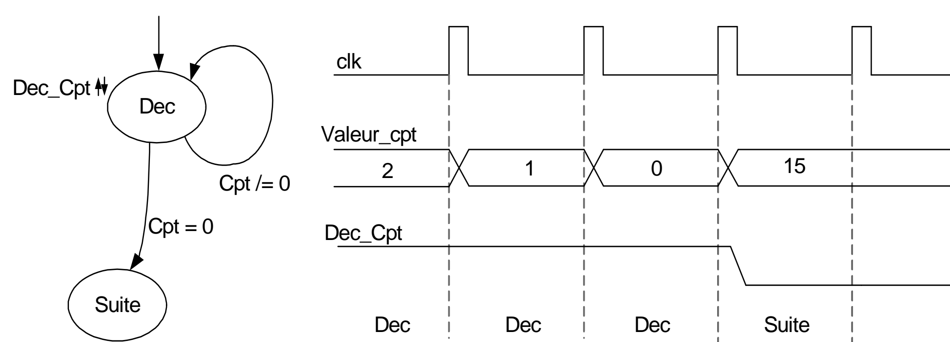


Figure 3- 15 : Fonctionnement incorrecte

Une solution à ce problème (figure 3- 16) consiste à ajouter un état supplémentaire appelé "Test" durant lequel le signal activant la décémentation est inactif. Si le décompteur n'a pas atteint zéro, il faut retourner à l'état de décrémentation. Si la valeur du décompteur est égale à zéro, il faut passer à l'état suivant et la valeur du décompteur restera égale à zéro. Dans ce cas l'état final du compteur est correct.

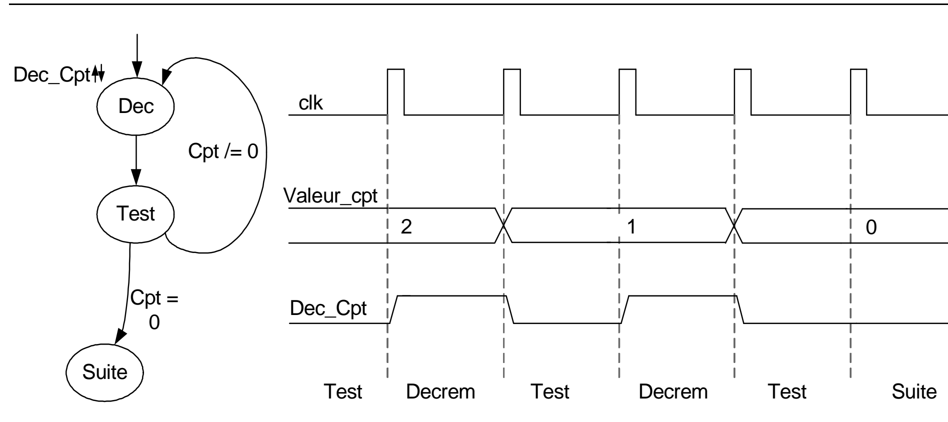


Figure 3- 16 : Fonctionnement correcte

Suite à l'ensemble de ces considération, nous pouvons établir l'organi-fgramme final de notre UC. Celuic-i est donnée dans la figure 3- 17.

Nous allons établir un graphe (figure 3- 18) à partir de l'organigramme détaillé de la figure 3- 14, ce qui nous permettra de vérifier le découpage en états. Nous avons ajouté les Att_Tst_LSB afin de palier au problème décrit précédemment.

Dans la figure 3- 18, seul l'activation des sorties est indiquée. Par défaut, les signaux de sorties sont à l'état inactif. Pour le signal Init_nAdd il y a des états où l'état de celui-ci est indifférent ('-'). Cela n'est pas indiqué dans le graphe de la figure 3- 18. Il y a une possibilité d'optimisation non exploitée dans cette version de graphe des états.

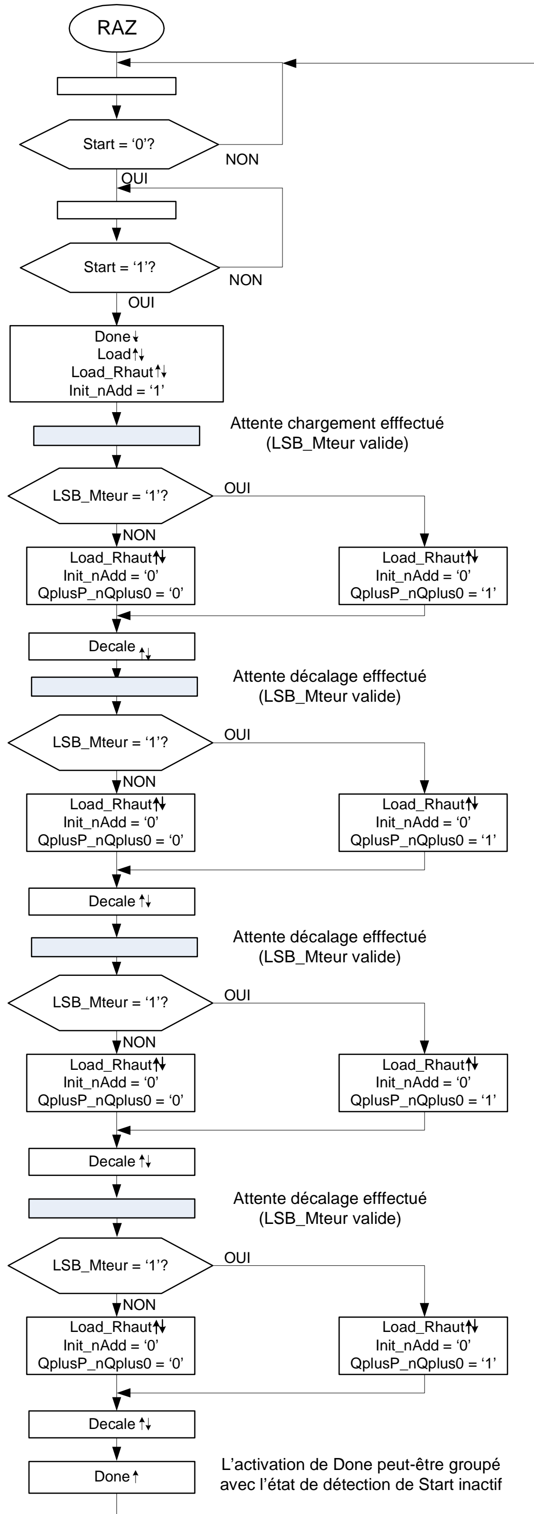


Figure 3- 17 : Organigramme détaillé final de la multiplication séquentielle

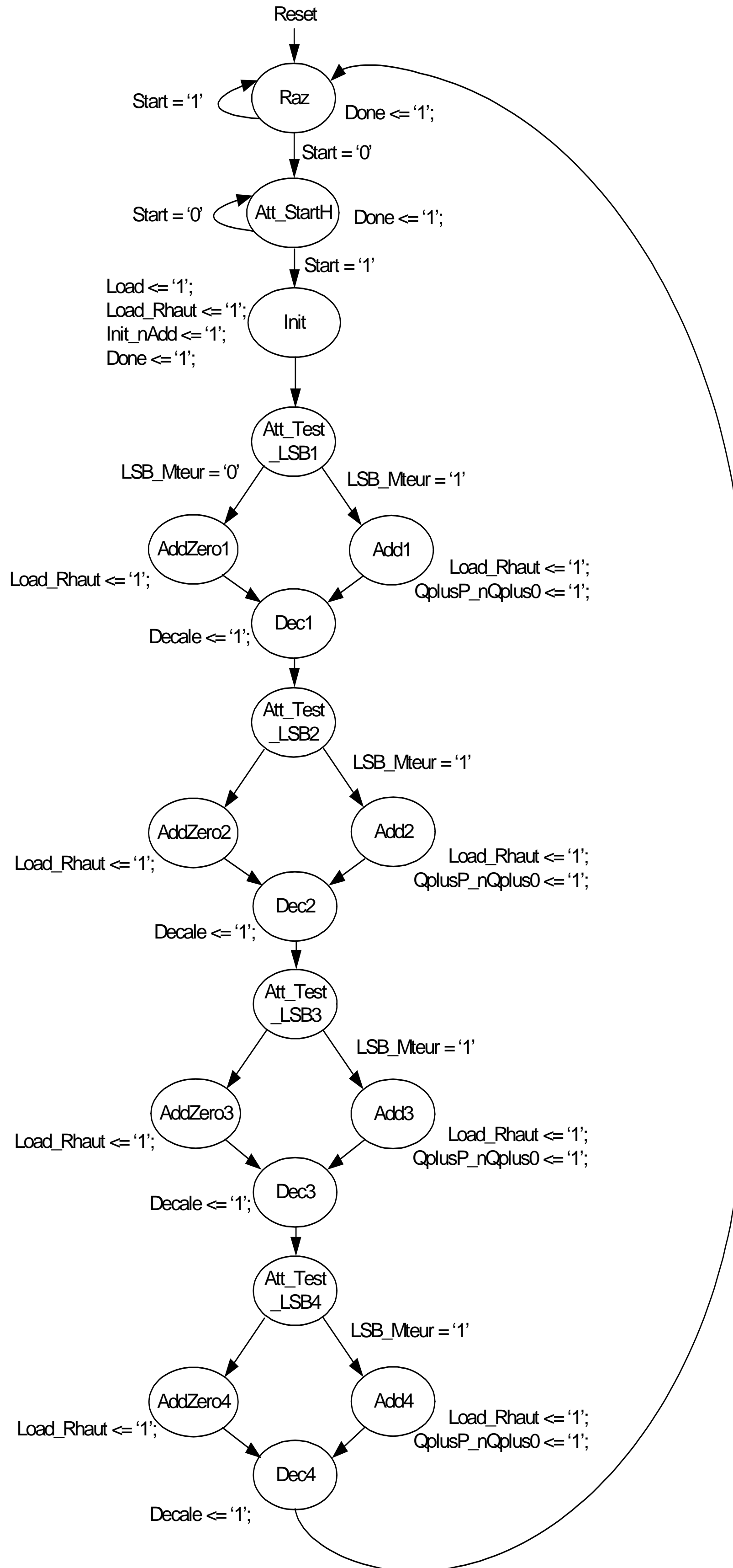


Figure 3- 18 : Graphe d'états

Decription VHDL de l'UC câblée

3-7 Exercices

1) Réaliser une version permettant la multiplication de deux nombres de n bits. Proposer une modification de l'organigramme et les évolutions de l'UT.

2) Etablir le graphe des états correspondant à l'exercice précédent (1.) Puis proposer une optimisation du décodeur de sorties sachant que les signaux Init_nAdd et QplusP_nQplus0 ne sont pas toujours utilisés.

3)Trouver un moyen de supprimer le coup d'horloge nécessaire au décalage. Dans la version proposée, il faut 2 coups d'horloge. Le premier mémorise le résultat du calcul, le second réalise le décalage. Proposer une modification permettant de faire ces deux opérations en un coup d'horloge.

4) Etablir le graphe des états correspondant à l'exercice 3.

Chapitre 4

Exemple :Distributeur automatique de billets

Pour illustrer la méthode de conception et les techniques de réalisation apparaissant dans ce chapitre, nous allons utiliser un second exemple: le développement d'un système digital gérant le fonctionnement d'un distributeur automatique de billets, donc les spécifications sont données ci-dessous. Cet exemple sera utilisé pour montrer l’évolution des unités de commande d’une version cablée à des structures micro-programmées.

Introduction.

La compagnie des Gyrobus Yverdonnois désire remplacer ses automates mécaniques de vente de billets par des machines à commande électronique. Nous sommes chargés de développer un prototype d'évaluation de la commande électronique, l’entreprise Bolomey S.A. est chargée de développer un collecteur de monnaie, un échangeur de monnaie et un distributeur de billets. Les spécifications de l’ensemble du système sont données ci-dessous.

4-1 Spécifications du distributeur

Spécifications préliminaires.

Le système de commande électronique doit diriger le fonctionnement du collecteur et de l'échangeur de monnaie, ainsi que du distributeur de billets, de façon à obtenir un système capable de vendre automatiquement des billets de Gyrobus à Fr. 0,90, et de rendre la monnaie si besoin est.

Fonctionnement désiré.

L'acheteur va introduire des pièces de monnaie dans le collecteur, pour un montant supérieur ou égal à 90 centimes. Lors de l'introduction de chaque pièce, notre commande doit en enregistrer la valeur et l'additionner au montant déjà payé, jusqu'à ce qu'il égal ou dépasse les 90 centimes, après quoi il faudra éjecter un billet et rendre la monnaie s'il y a lieu.

En cas de malfonctionnement, le client doit pouvoir récupérer l'argent versé, en appuyant sur un bouton poussoir, et remettre du même coup tout le système dans son état initial. Dans ce but, les pièces introduites sont d'abord conservées dans un godet intermédiaire du collecteur de monnaie. Par action mécanique directe, le bouton-poussoir vide ce godet intermédiaire dans le godet de retour.

Si tout se passe normalement, lorsque la monnaie a été rendue et le billet éjecté, la commande doit provoquer le vidage du godet intermédiaire dans la caisse, et se remettre dans l'état initial.

Collecteur de monnaie.

Le collecteur de monnaie a les caractéristiques suivantes:

a) une seule fente d'introduction des pièces.

b) grandeur de la fente limitant les pièces à 1Fr. maximum.

c) les pièces de 1Fr., 50 ct., 20 ct., et 10 ct. suisses sont reconnues et acceptées par l'appareil.

d) les fausses pièces ou les pièces endommagées sont automatiquement rejetées.

e) un godet de collecte intermédiaire peut être vidé dans la caisse en actionnant un signal de commande appelé ENCAISSE, ou dans le godet de retour en agissant manuellement sur le bouton-poussoir prévu à cet effet.

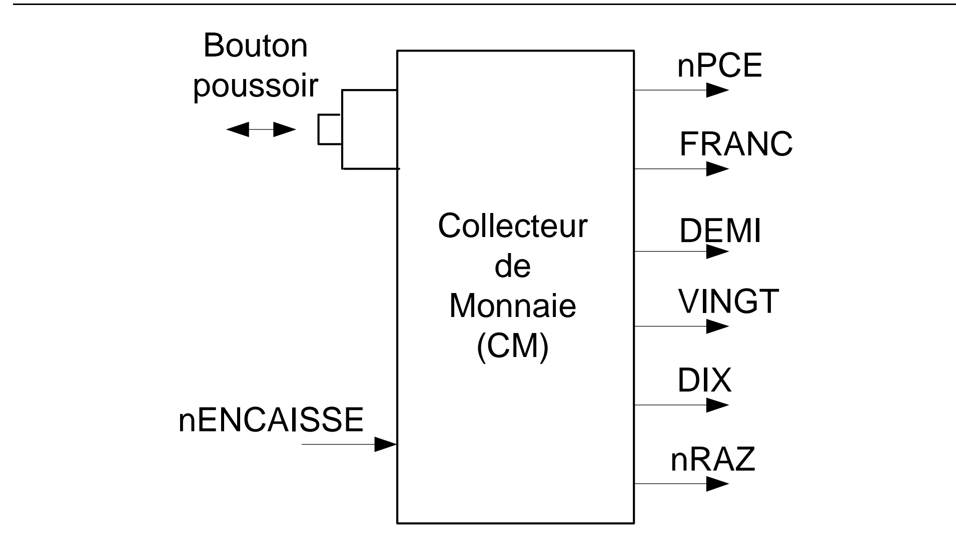


Figure 4- 1 : Schéma bloc du collecteur de monnaie

Description des signaux:

nENCAISSE : entrée, provoque le vidage dans la caisse (magasin) des pièces collectées lors d'une transaction.

nPCE : sortie, indique qu'une pièce valide vient d'être introduite et que sa valeur a été déterminée.

FRANC, DEMI, VINGT, DIX : sorties, indiquant la valeur de la pièce qui vient d'être introduite.

nRAZ : sortie, active tant que l'on presse sur le bouton-poussoir.

Le chronogramme de la figure 4- 2 suivante définit les relations temporelles entre nPCE d'une part et FRANC, DEMI, VINGT ou DIX d'autre part.

Si une fausse pièce est détectée, plusieurs des signaux FRANC, DEMI, VINGT ou DIX peuvent être actifs, mais le signal nPCE reste inactif. Si une pièce valide est détectée, un seul des signaux FRANC, DEMI, VINGT, ou DIX sera actif lorsque nPCE est actif.

La commande impulsionnelle nENCAISSE doit durer plus de 250msec.

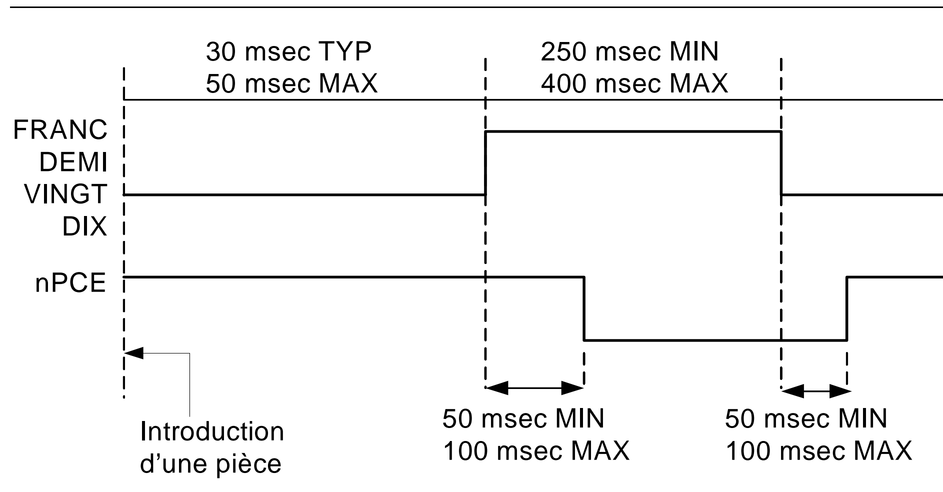


Figure 4- 2 : Relations temporelles entre différents signaux

Échangeur de monnaie.

Les caractéristiques de l'échangeur de monnaie sont les suivantes:

a) ce système électro-mécanique permet d'éjecter des pièces de 10ct dans le godet de retour.

b) les pièces de 10 ct. sont prises dans un magasin de pièces spécialement prévu à cet effet.

c) l'éjection est déclenchée par un signal d'entrée impulsionnel.

d) un signal de sortie indique si l'échangeur est prêt à éjecter une nouvelle pièce.

e) toute commande d'éjection faite pendant que l'échangeur n'est pas prêt est ignorée.

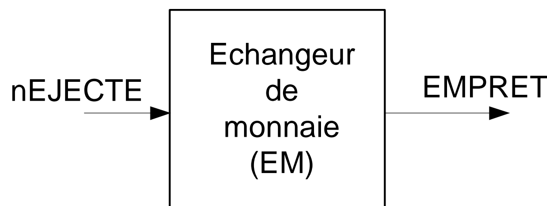


Figure 4- 3 : Schéma bloc de l’échangeur de monnaie

Description des signaux

nEJECTE : entrée, provoque l'éjection d'une pièce de 10ct.

EMPRET : sortie, indique que l'échangeur est prêt à éjecter une nouvelle pièce.

Les relations temporelles entre ces signaux sont définies par le chrono-

gramme de la figure 4- 4.

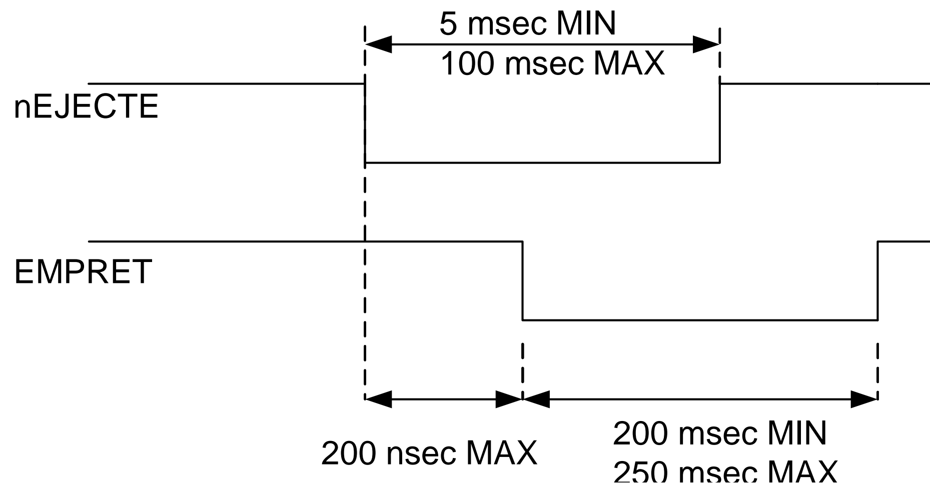


Figure 4- 4 : Relations temporelles entre les signaux nEJECT et EMPRET

Mécanisme distributeur de billets.

Le mécanisme distributeur de billets a les caractéristiques suivantes:

a) distribue un billet à la fois à l'aide d'un système électro-mécanique.

b) comporte une entrée pour commander la distribution d'un billet.

c) comporte une sortie indiquant si le distributeur est prêt à éjecter un nouveau billet ou non.

d)toute commande d'éjection d'un billet faite pendant que le distributeur n'est pas prêt est ignorée.

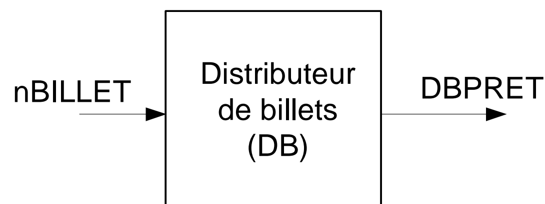


Figure 4- 5 : Schéma-bloc du mécanisme distributeur de billets

Description des signaux:

nBILLET : entrée, provoque l'éjection d'un billet, signal impulsionnel

DBPRE : sortie, indique que le distributeur est prêt à éjecter un nouveau billet.

Les relations temporelles entre les signaux nBILLET et DBPRET sont les mêmes que pour les signaux nEJECT et EMPRET.

4-2 Schéma-bloc général et organigramme grossier

Les étapes 1 et 2 de la méthode de conception sont traitées (bien que de façon incomplète et en partie insatisfaisante, comme c'est souvent le cas) dans le paragraphe qui précède, pour notre exemple. Nous pouvons relever que plusieurs spécifications importantes, telles celles relatives à la fiabilité, à l'entretient, aux possibilités de modifications (adaptation du prix, par exemple), à l'environnement dans lequel l'appareil fonctionne (salle d'attente, plein air, couvert), etc, n'ont pas été définies. Par ailleurs, une sur-spécification, un à-priori restreignant inutilement notre liberté de concepteurs, s'est glissé dans la description du fonctionnement désiré. Mais nous y reviendrons par la suite.

En ce qui concerne les relations temporelles, nous n'avons pas de contraintes particulières si ce n'est que de respecter les caractéristiques des trois mécanismes utilisés. En effet, notre commande n'a pas besoin de fonctionner bien vite par rapport aux possibilités de l'électronique numérique, vu que les trois mécanismes ont des temps de réponse supérieur au dixième de seconde.

Nous pouvons donc passer à l'étape 3 et établir un schéma-bloc général. Celui-ci est représenté à la figure 4- 6.

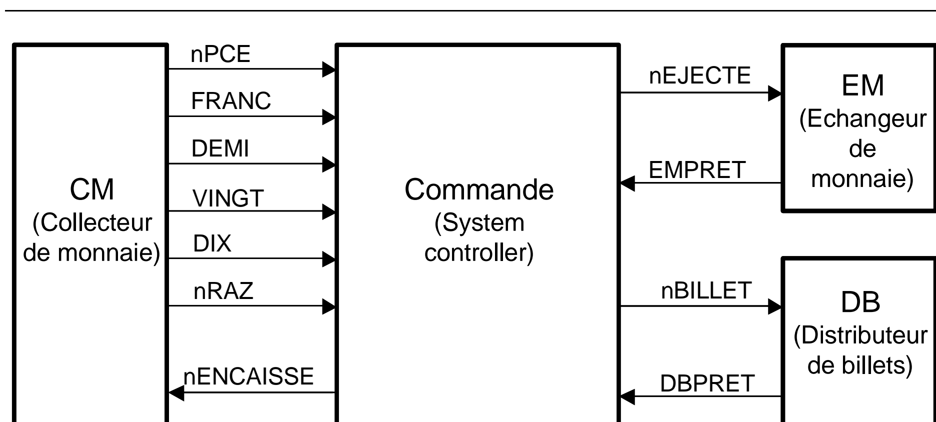


Figure 4- 6 : Schéma-bloc général

L'organigramme de la figure 4- 7 décrit un comportement globale possible de notre commande de distributeur de billets. Il s'agit ici d'une description grossière, que nous affinerons par la suite, mais on constate

immédiatement que le comportement envisagé est plus clairement décrit que par une suite de phrases.

Pour grossier qu'il soit, cet organigramme fait déjà apparaître quatre fonctions standard qui pourraient constituer l'ossature d'une unité de traitement.

1)l'addition de la valeur d'une pièce au total accumulé lors de cette transaction.

2)la mémorisation du total intermédiaire.

3)la comparaison entre le total et le prix.

4)la soustraction de 10ct. au total lorsqu'une pièce de 10ct. est rendue.

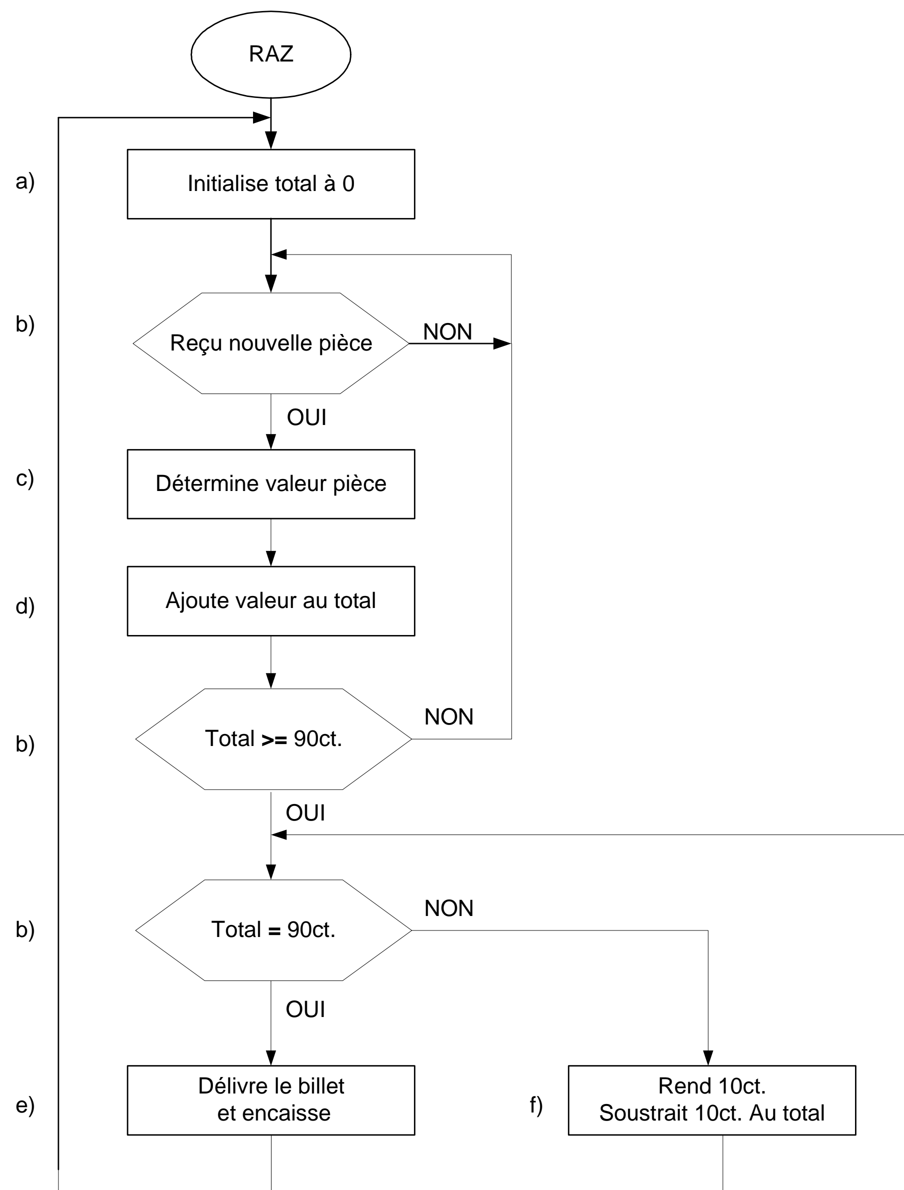


Figure 4- 7 : Organigramme grossier

4-3 Partition commande / traitement

En utilisant les annotations visibles sur l'organigramme grossier (figure 4- 7), nous allons identifier les fonctions à effectuer dans l'unité de traitement ou dans l'unité de commande.

a)Pour initaliser le total à zéro, il faut effectuer une initilisation du registre contenant le total. Celui-ci sera situé dans l'UT.

b) Le test de la réception d'une nouvelle pièce sera effectué dans l'UC. Cette dernière contrôle le séquencement du système.

c) Pour déterminer la valeur de la pièce, il faut transformer les signaux fournis par le collecteur de monnaie en une valeur utilisable par notre système, nombre en binaire (base 2). Nous allons utiliser un encodeur.

d) Pour ajouter la valeur de la piéce au total, il faut un additionneur ainsi qu'un registre. Ces deux éléments se trouvent dans l'UT.

e) Pour délivrer le billet et encaisser la monnaie, il faut activer les signaux nBILLET et nENCAISSE. Avant d'activer nBILLET, il faut s'assurer que le distributeur de billets soit pret. Il faut donc tester le signal DBPRET.

f) L'éjection de la pièce sera commandé par l'UC qui activera le signal nEJECTE après s'être assuré que l'ejecteur de monnaie soit prêt. Chque fois qu'une pièce est rendue, il faudra soustraire 1 du total. Nous allons utiliser un décompteur qui se trouvera dans l'UT.

Nous allons dès lors concevoir l'unité de traitement spécialisée pour notre vendeur de billets en choisissant tout d'abord de favoriser la vitesse de fonctionnement. Nous pouvons dès lors lister les éléments nécessaires:

-un additionneur pour effectuer l'addition.

-un registre pour mémoriser le total (ou un compteur chargeable).

-un comparateur pour comparer le total avec le prix

-un soustracteur pour décompter le total (ou un compteur-décompteur).

Le schéma bloc de la figure 4- 8 représente une des solutions possible. La soustraction et la mémorisation seront faites à l'aide d'un compteur-décompteur chargeable. Le total et la valeur des pièces seront exprimés en nombre équivalent de pièces de 10ct. (10ct. est l'unité de mesure).

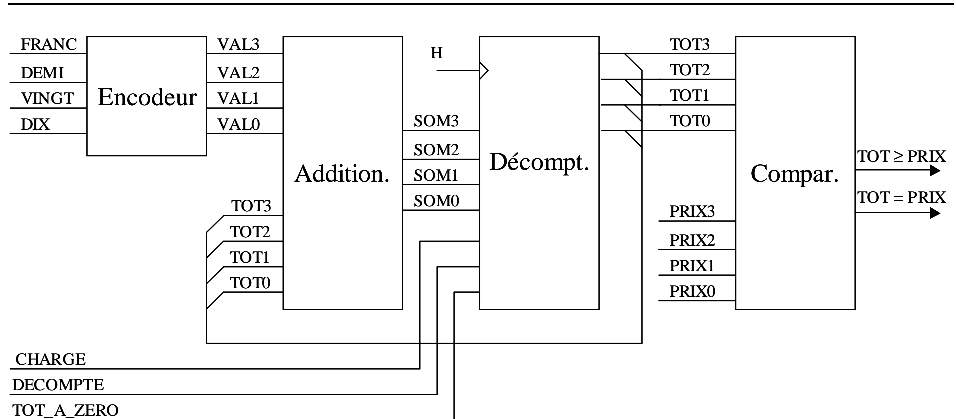


Figure 4- 8 : Schéma-bloc de la soustraction et de la mémorisation

Étant donné qu'un seul des signaux FRANC, DEMI, VINGT ou DIX sera actif à un instant donné, l'encodeur peut être réalisé très simplement comme nous le montre la figure 4- 9.

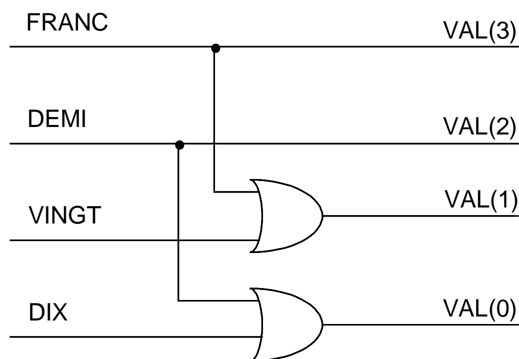


Figure 4- 9 : Encodeur

Bien que notre cahier des charges ne le spécifie pas, nous savons bien que le prix du billet aura tendance à augmenter. Nous allons donc prévoir l'adaptation du prix à l'aide de simples ponts enfichables(jumpers).

Le compteur sera chargé avec le résultat de l'addition à chaque fois qu'une nouvelle pièce est introduite. La décrémentation se fera lorsque une pièce de 10ct. est rendue. La remise à zéro du total doit se faire lorsque l'automate vide le godet intermédiaire dans la caisse (nENCAISSE), ou lorsque le client presse sur le bouton-poussoir (nRAZ), ou lors de la mise sous tension.

L'unité de traitement que nous avons conçue restre très perfectible. En effet, les spécifications initiales étaient incomplètes, mais aussi trop restrictives. Ainsi, plusieurs cas de fonctionnement marginal n'ont pas été pris en considération: que se passe-t-il si les pièces sont introduites trop rapidement?, ou , pendant que l'appareil rend la monnaie, si le client presse sur le bouton-poussoir lorsqu'il a déjà reçu une partie du change?, si le client in-

troduit une pièce de 1Fr. Après avoir déjà payé 80ct.?, etc. La description du fonctionnement désiré a été faite de façon tendancieuse: elle pré-suppose que les valeurs des pièces versées vont être additionnées ce qui devrait être laissé au libre choix du concepteur, une soustraction ou décrémentations pouvant aussi faire l'affaire. Finalement, le choix que nous avons fait de favoriser la vitesse de traitement ne se justifie guère dans un appareil dont le fonctionnement global est tributaire de sous-systèmes électro-mécaniques beaucoup plus lents que la commande que nous allons concevoir, quelle que soit la structure de cette dernière. Les diverses améliorations seront des sujets d'exercice.

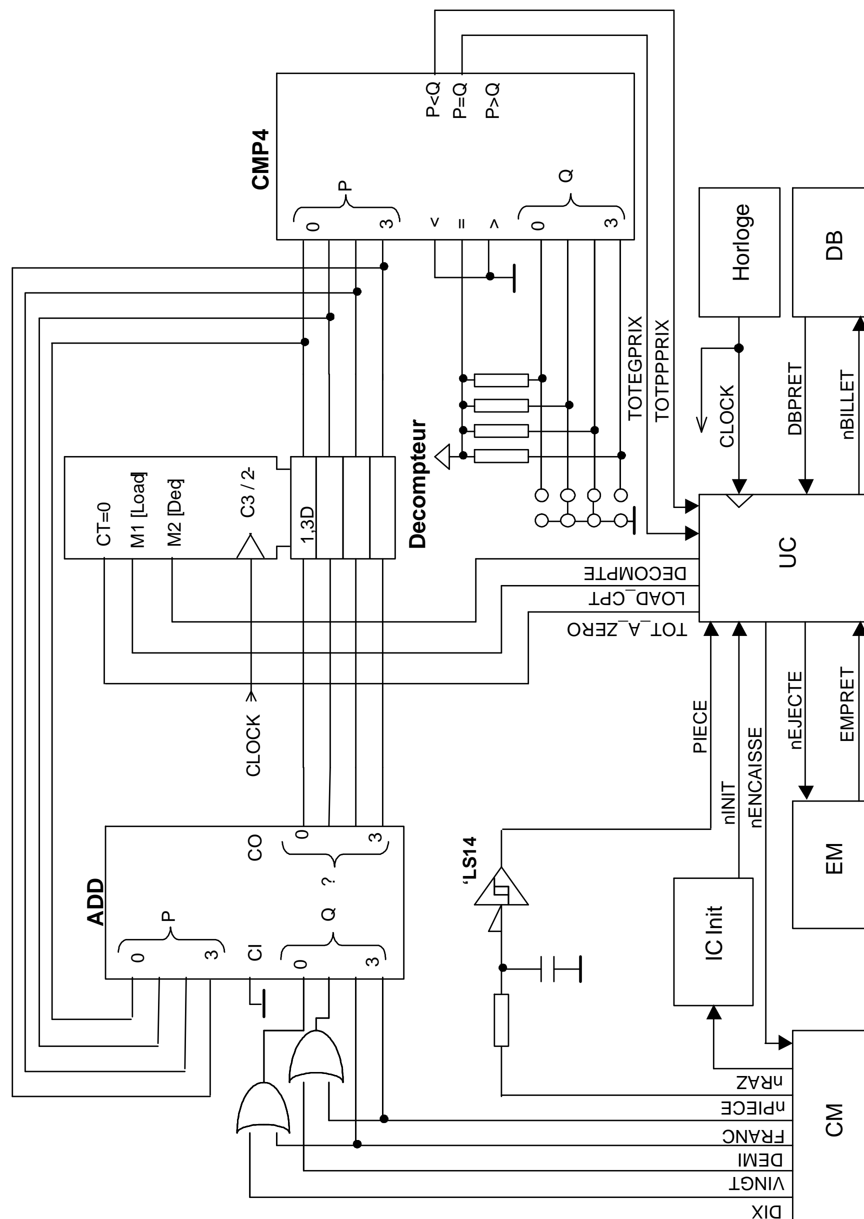


Figure 4- 10 : Schéma de l’UT

Le schéma de la figure 4- 10 appelle quelques remarques:

1) Les câbles qui relient le système de commande du vendeur de billets aux divers sous-systèmes électro-mécaniques sont suseptibles de capter des parasites. Il est souvent nécesaire de filtrer les signaux entrant dans un sous système logique. Une méthode souvent utilisée consiste à faire passer le signal à travers un réseau RC et un tigger de Schmitt, comme le signal PIECE.

2) les signaux DECOMPTE et nEJECTE peuvent être identiques, d'un point de vue purement logique. En effet, il s'agit de décrémenter le total à chaque fois qu'une pièce de 10ct. est rendue. Mais le signal nEJECT passera par un câble et peut donc être perturbé par des parasites. Pour éviter que ces parasites ne provoquent des décomptages intempestifs, le signal DECOMPTE sera généré séparement du signal nEJECTE.

3) à part l'unité de traitement, le reste du système logique est constitué de l'unité de commande, d'un circuit d'horloge et d'un circuit d'initialisation qui va activer le signal nINIT lors de la mise sous tension, lorsque le client presse sur le bouton-poussoir du collecteur de monnaie, ou lorsque le réparateur presse sur le bouton d'initialisation que nous ne manquerons pas de prévoir.

4) les symboles CEI d'un additionneur et d'un comparateur sont suffisamment explicites. Relevons toutefois que ces circuits sont cascadables (les fonctions d'addition et de comparaison peuvent toutes deux être décomposées en cascade) et disposent donc d'entrées et de sorties prévues à cet effet: le report d'entrée CI et de sortie CO pour l'additionneur, les entrées "<", "=", et ">", et les sorties "P<Q", "P=Q" et "P>Q" pour le comparateur.

4-4 Exercices

1) Modifier l'unité de traitement de la figure 4- 10 de façon à ce que le vendeur de billets fonctionne correctement lorsqu'un client introduit une pièce de 1Fr. après avoir déjà mis 80ct. dans l'appareil. Adapter l'organigramme de la figure 4- 7 s'il y a lieu.

2) Au lieu d'additionner les valeurs des pièces introduites jusqu'à ce que le total égale ou dépasse le prix, on peut imaginer de partir du prix et de soustraire les valeurs des pièces introduites. Le reste à payer pourra aussi être constamment affiché afin que le client sache où il en est. Concevoir un nouvelle unité de traitement suivant cette optique, après avoir établi un organigramme grossier qui décrive ce comportement.

3) Adapter le schéma de la figure 4- 10 et l'organigramme de la figure 4- 7, de façon à supprimer le comparateur et à effectuer la comparaison dans l'unité de commande.

4)Modifier l'organigramme de la figure 4- 7 et le schéma de figure 4-10, de façon à ce que la totalisation du montant payé se fasse par comptage, plutôt que par addition proprement dite.

5) En faisant la synthèse des résultats obtenus dans les exercices 1 à 4, concevoir une unité de traitement intégrable dans un PLD de 24 pattes au maximum.

4-5 Organigramme détaillé

En tenant compte de l'unité de traitement de la figure 4- 10, nous pouvons maintenant établir un organigramme détaillé pour l'UC de notre automate de vente de billets, et le graphe correspondant.

Un organigramme détaillé doit comporter les commentaires utiles à sa compréhension (à coté des boîtes d'action ou de test), des noms pour les états (dans un cercle à côté de chaque boîte d'action inconditionnelles voire même les codes choisis pour les états ce qui facilitera l'utilisation de cet organigramme pour la documentation et le test.

Pour établir un organigramme détaillé, il faut tenir compte de multiples facteurs: les entrées asynchrones, les contraintes temporelles, le découpage en état distincts, etc. Souvent, plusieurs étapes intermédiaires sont nécessaires avant d'aboutir à un organigramme détaillé satisfaisant. Ainsi, l'organigramme de la figure 4- 11 peut être considéré comme une première étape dans l'évolution de l'organigramme grossier de la figure 4- 7, vers l'organigramme détaillé de la figure VII33.

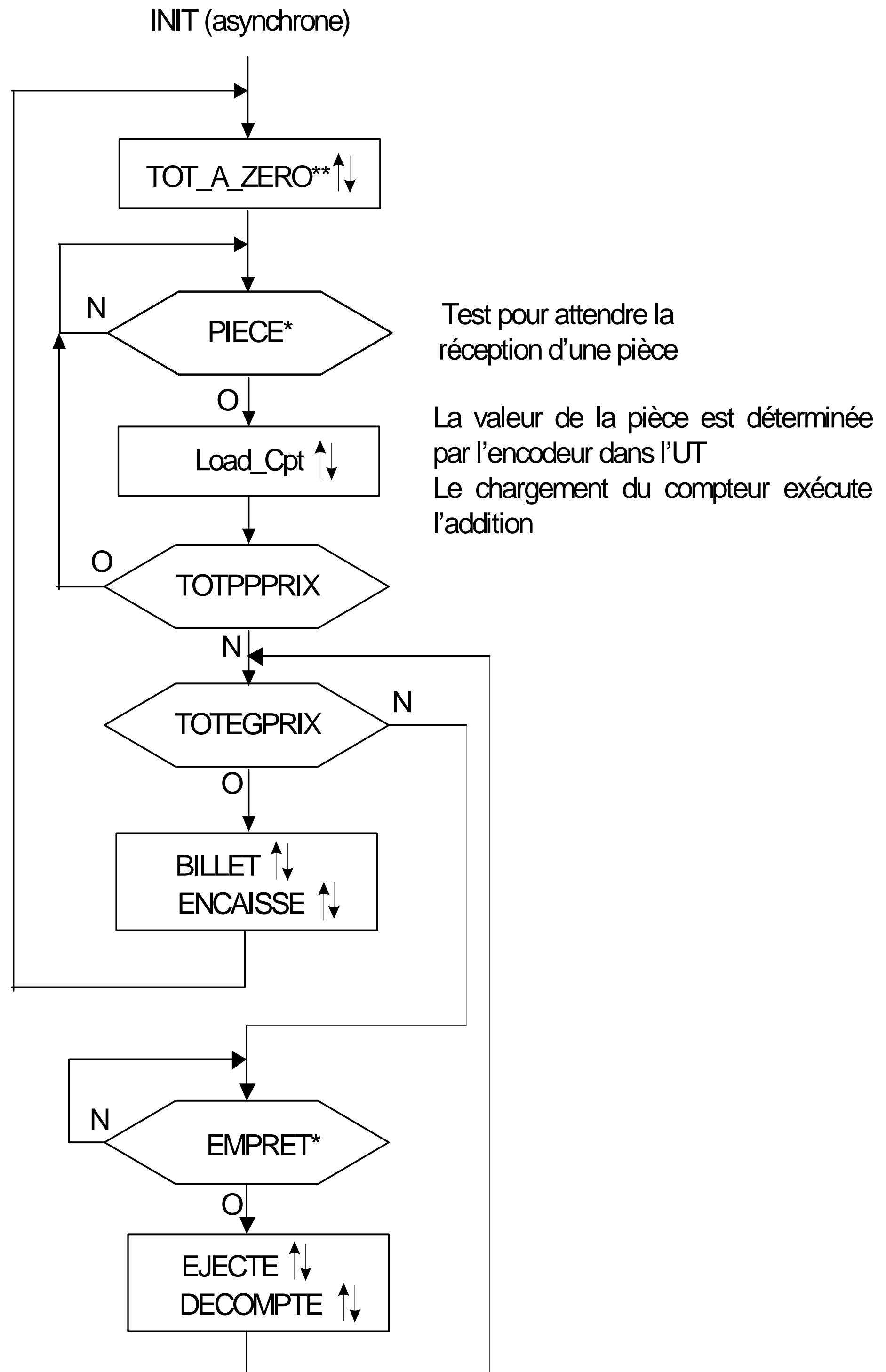


Figure 4- 11 : Organigramme détaillé de l'UC (premier raffinement)

L'organigramme de la figure 4- 11 a été établi à partir de celui de la figure 4- 7, en introduisant les noms des signaux que nous avons définis, en supprimant les opération réalisées par l'UT sans intervention de l'UC, et en tenant compte des caractéristiques de l'échangeur de monnaie.

Lors de l'activation du signal Load_Cpt, le total ne sera pas à jour immédiatement. Il faudra tester l'état des signaux TOTEGPRIX et TOTPPPRIX dans un état suivant.

Il faut également nous assurer que toutes les sorties de l'UC respectent les contraintes temporelles de l'UT et du reste du système. Ainsi, l'impulsion à l'état actif de TOT_A_ZERO devra durer plus de 20ms, celle de BILLET et celle d'EJECTE doivent durer plus de 5msec mais moins de 100msec, celle d'nENCAISSE doit durer plus de 250msec, alors que quelques nanosecondes suffisent pour DECOMPTE (mais DECOMPTE et EJECTE peuvent être activées en même temps). Une période d'horloge située entre 5 et 100 msec. fera l'affaire pour TOT_A_ZERO, BILLET, EJECT et DECOMPTE. Pour ENCAISSE nous avons le choix entre créer un monostable à l'aide d'un compteur, ou maintenir le signal actif pendant le nombre de périodes voulu, en adaptant l'organigramme en conséquence. Nous choisirons cette deuxième solution, car elle est plus économique, mais nous utiliserons un test de DBPRET pour créer une attente d'au moins 200msec, à laquelle il suffira d'ajouter 50msec. Une seule période d'horloge suffira pour ajouter ces 50msec. si nous choisissons une fréquence d'horloge entre 10 et 20Hz.

Afin de pouvoir générer notre signal d'horloge à partir du réseau électrique, nous choisirons f = 50/4=12,5Hz (ou 67msec.).

L'organigramme de la figure 4- 12 tient compte de ces réflexions. Cet organigramme suppose que la fréquence d'horloge est de 10 à 20Hz.

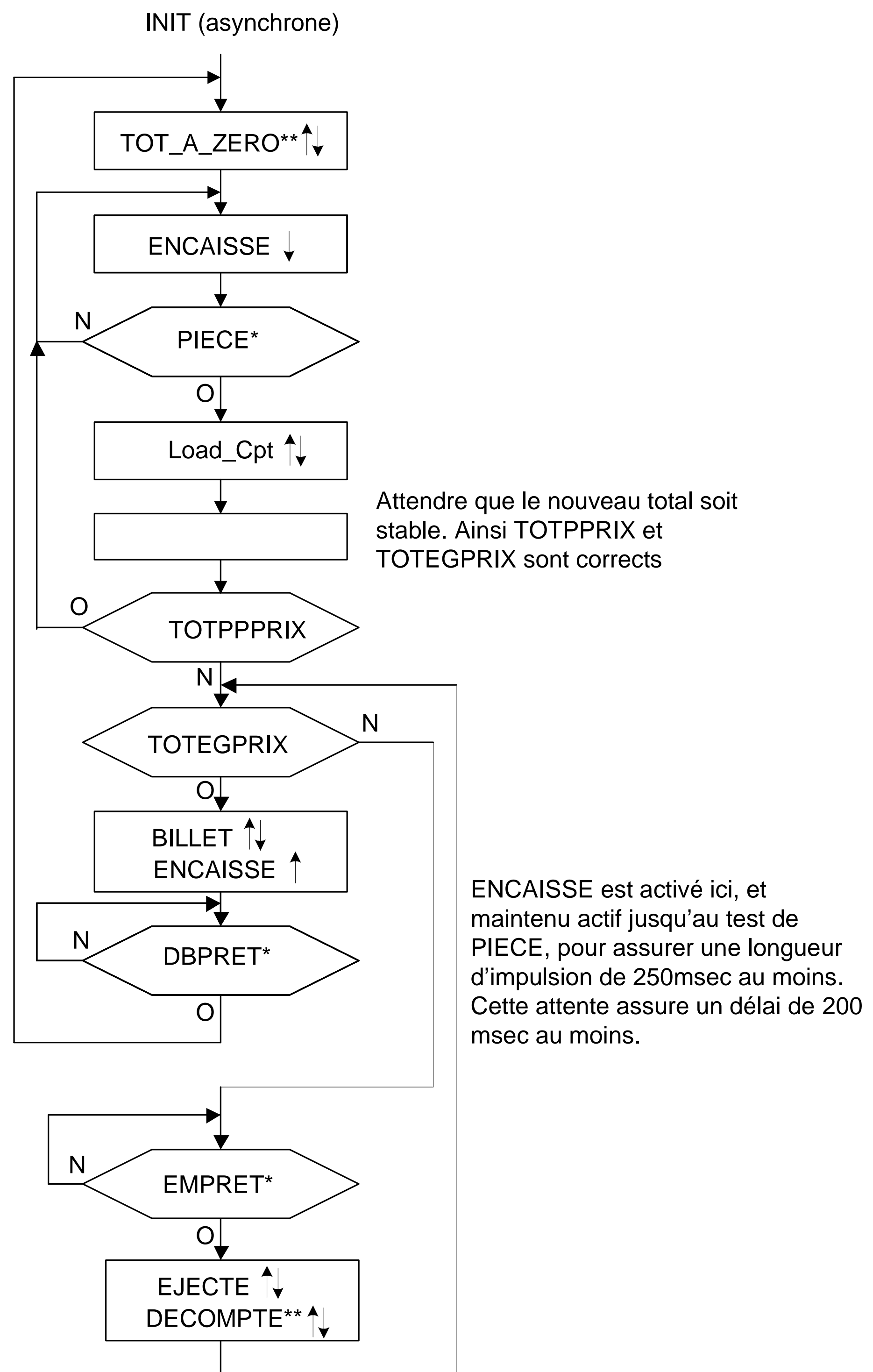


Figure 4- 12 : Organigramme détaillé de l'UC (deuxième raffinement)

Si nous choisissons de réaliser une UC de type microprogrammée, l'organigramme de la figure 4- 12 peut être considérée comme suffisamment détaillé. Pour une UC de type câblé par contre, il va falloir étudier le découpage en états. En d'autres termes, il va falloir compléter et/ou modifier

l'organigramme de façon à ce que le graphe qui en sera tiré suivant les règles du paragraphe 3-4 réponde aux spécifications, comporte le moins d'états possible et tienne compte des entrées ou conditions asynchrones.

L'organigramme de la figure 4- 13 a été complété en vue de la réalisation d'une UC câblée. Nous voyons ainsi apparaître des boîtes d'action vides (puisqu'il n'y a aucune action à entreprendre) dans les boucles d'attente de DBPRET et de EMPRET: elles sont nécessaires pour respecter la règle n°1 du paragraphe 3-4, puisque chaque boucle doit comporter au moins un état. L'attente de PIECE, par contre, ne nécessite pas la création d'un état: il suffit d'attendre dans l'état où ENCAISSE est désactivé. Les tests de TOTPPPRIX et TOTEGPRIX doivent se faire après un état d'attente après le chargement du registre. En effet l'action de chargement a lieu en sortant de l'état. Il faut donc rajouter une boîte d'action entre le chargement du compteur et les tests de TOTPPPRIX et TOTEGPRIX.

Les tests de TOTPPPRIX et TOTEGPRIX peuvent se faire dans le même état, puisque ces deux entrées ne seront testées que lorsqu'on est sûr qu'elles sont stables (il n'est donc pas nécessaire de respecter la règle de codage n°5 retreinte).

Il ne reste plus qu'à donner des noms aux états, à mettre les commentaires utiles et à indiquer de quelle façon la sortie TOT_A_ZERO sera générée sans transitoires. Par la suite, les codes d'état peuvent être ajoutés à l'organigramme, afin de compléter la documentation (paragraphe 3-4).

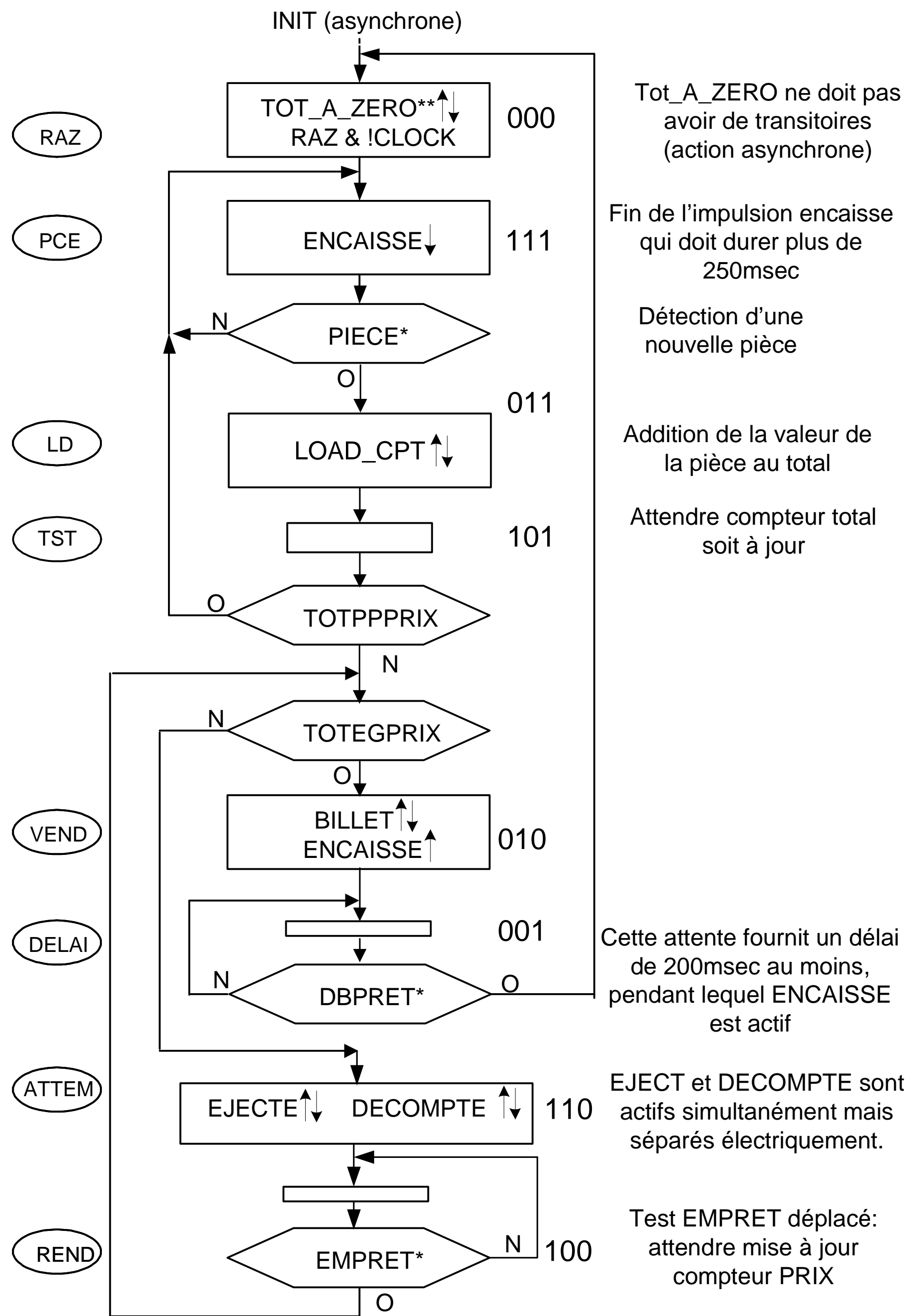


Figure 4- 13 : Organigramme détaillé avec indiquations pour la réalisation de l'UC câblée.

4-6 UC câblée

Nous allons établir un graphe à partir de l'organigramme détaillé de la figure figure 4- 13, ce qui nous permettra de vérifier le découpage en états, puis nous procéderons comme pour la réalisation d'une MSS à l'aide d'un PLD. Le graphe obtenu est celui de la figure 4- 14.

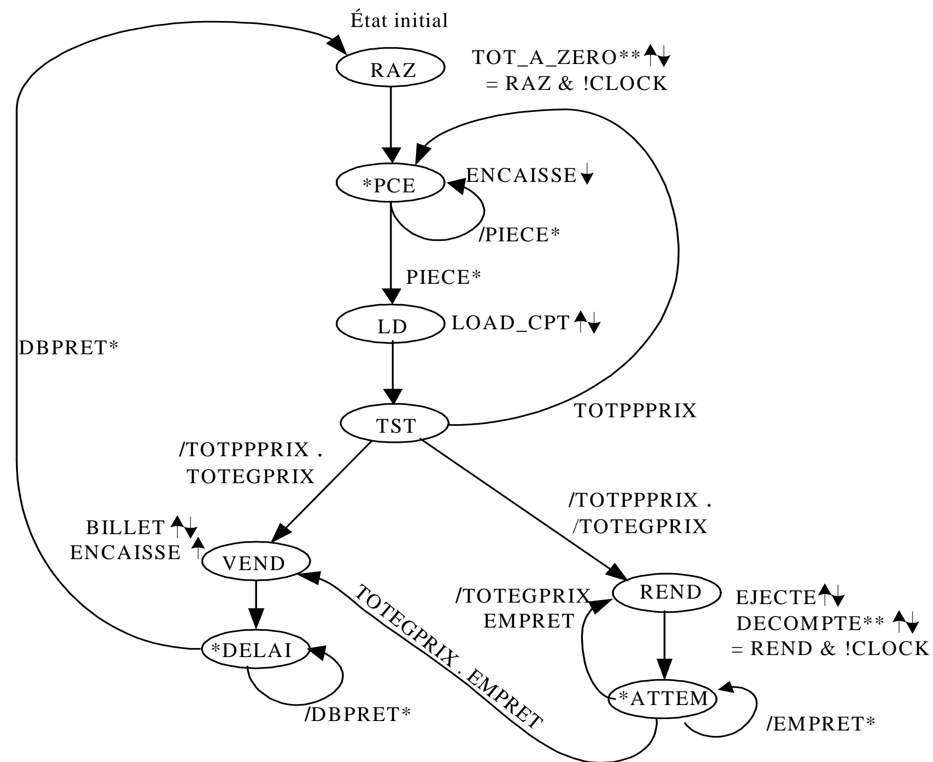


Figure 4- 14 : Graphe d'états

La sortie ENCAISSE pose un problème particulier, puisqu'elle doit être activée dans l'état VEND, désactivé dans l'état PCE, et ne doit pas être modifiée dans les autres états de la séquence. D'une façon générale, une telle sortie est générée à travers une bascule bistable (soit une bascule RS asynchrone, mais attention aux transitoires sur ses entrées, soit une bascule "edge-triggered" de type RS, JK ou D-CE), qui sera mise à 1 dans l'état VEND, et mise à 0 dans l'état PCE (ou l'inverse selon la polarité désirée). Notre problème accepte une solution plus simple, qui consiste à activer cette sortie dans les états VEND, DELAI et RAZ, et ne pas l'activer dans les autres états, car dans aucun état de notre séquence la sortie ENCAISSE ne dépend de ce qui s'est passé précédemment, rendant la mémorisation inutile.

Étant donné que nous avons des entrées asynchrones (en fait il faudrait plutôt dire "des entrées pouvant changer simultanément avec le flanc d'horloge qui détermine le passage de l'état où elles sont testées vers un état futur", mais c'est un peu long), nous devrons respecter la règle de codage n°5. La figure 4- 15 nous montre un codage possible.

	00	01	11	10
0	RAZ	VEND	ATTEM	REND
1	DELAI	LD	PCE	TST

Figure 4- 15 : Exemple d'un codage possible

Le développement de ce circuit en VHDL est documenté ce-après.

Description VHDL de l'UC câblée

```
entity UC is
   port(
      Clk_i        : in     std_logic;
      Db_Pret_i    : in     std_logic;
      Em_Pret_i    : in     std_logic;
      Piece_i      : in     std_logic;
      TotEgPrix_i  : in     std_logic;
      TotPpPrix_i  : in     std_logic;
      nInit_i      : in     std_logic;
      Decompte_o   : out    std_logic;
      Load_Cpt_o   : out    std_logic;
      Tot_A_Zero_o : out    std_logic;
      nBillet_o    : out    std_logic;
      nEject_o     : out    std_logic;
      nEncaisse_o  : out    std_logic
   );
end entity UC ;

architecture FSM of UC is
   type State_Type is (RAZ, PCE, LD, TST, ATTEM, REND,
                                                  VEND, DELAI);
   signal Cur_State, Next_State : State_Type;
begin -- FSM

   ---------------------------------------------------
   clocked : process( Clk_i, nReset_i)
   ---------------------------------------------------
   begin
      if (nReset_i = '0') then
         Cur_State <= RAZ;
      elsif (Clk_i'EVENT and Clk_i = '1') then
         Cur_State <= Next_State;
      end if;
   end process clocked;
```

```
---------------------------------------------------
nextstate : process (Cur_State, TotPpPrix_i, Piece_i,
                 TotEgPrix_i, Em_Pret_i, Db_Pret_i)
---------------------------------------------------
begin
case Cur_State is
      when RAZ =>
              Next_State <= PCE;

      when PCE =>
          if (Piece_i = '0') then
              Next_State <= PCE;
          else
          Next_State <= LD;
          end if;

      when LD =>
             Next_State <= TST;

      when TST =>
          if (TotPpPrix_i = '1') then
          Next_State <= PCE;
          else
          if (TotEgPrix_i = '1') then
          Next_State <= VEND;
          else
                  Next_State <= ATTEM;
          end if;
          end if;

      when REND =>
          if (TotEgPrix_i = '0') then
          Next_State <= ATTEM;
          else
          Next_State <= VEND;
          end if;

      when ATTEM =>
          if (Em_Pret_i = '0') then
          Next_State <= ATTEM;
          elsif (TotEgPrix_i = '1') then
          Next_State <= VEND;
          else
          Next_State <= REND;
          end if;

      when VEND =>
          Next_State <= DELAI;

      when DELAI =>
          if (Db_Pret_i = '0') then
          Next_State <= DELAI;
          else
          Next_State <= RAZ;
          end if;

when others =>
    Next_State <= RAZ;
```

```
        end case;

        end process nextstate;

        -- Affectation des sorties
        Tot_A_Zero_o  <= '1' when Cur_State = RAZ else '0';
        Encaisse_o    <= '1' when Cur_State = VEND or
                                  Cur_State = DELAI or
                                   Cur_State = RAZ else '0';
        Load_Cpt_o<= '1' when Cur_State = LD else '0';
        Billet_o<= '1' when Cur_State = VEND else '0';
        Eject_o<= '1' when Cur_State = REND else '0';
        Decompte_o<= '1' when Cur_State = REND else '0';

    end FSM;
```

4-7 Exercices

6) Pour chacune des unités de traitement développées dans les excercices 1 à 5, établir l'organigramme détaillé et le graphe de l'UC câblée correspondante.

7)Réaliser l'UC câblée de la page 57 à l'aide d'une mémoire PROM et d'un registre parallèle-parallèle.

8) Dans le graphe de la figure 4- 14, prenons le codage suivant : RAZ=000, VEND=001, LD=101, TST=110, ATTEM=111, REND=110, VEND=101, DELAI=100. Un compteur 4 bits avec chargement et comptage eset choisi comme générateur d'états (voir schéma-bloc de la figure 1-4). Etablir une table de vérité qui indique les commandes à appliquer au compteur de façon à ce qu'il se comporte comme spécifié par le graphe.

9) Un codage tel que celui de l'exercice 8 ne respecte pas la règle n°5. Il faudra donc synchroniser les entrées asynchrones. D'autre part, il n'y a pas plus d'une entrée qui influence l'évolution à partir d'un état quelconque du graphe de la figure 4- 14. Il est donc raisonable de multiplexer ces entrées et de ne synchroniser que la sortie MUX. Etudier la réalisation de l'UC de la page 57, dans cette optique, avec une CPLD EMP7128S

Chapitre 5

Unité de commande microprogrammée

L'exercice 8 page 61 nous a rappelé qu'une mémoire ROM peut-être utilisée pour réaliser ce que nous appelions "décodeur d'état futur" et "décodeur de sortie", et que nous appelons désormais "bloc de calcul des commandes de séquencement" et "bloc de calcul des sorties et des commandes de l'UT", respectivement. Pour la réalisation d'UC complexes, qui nécessiteraient l'utilisation d'un FPGA dans notre structure câblée, l'utilisation d'une mémoire apporte divers avantages:

1)La densité d'intégration d'une mémoire est plus élevée que celle d'une fonction identique réalisée en "logique câblée", à plus forte raison si cette "logique câblée" est en fait un réseau programmable. Cette plus grande densité d'intégration influence favorablement le coût et la consommation.

2) Il est très facile de modifier les fonction combinatoires générées par une mémoire, que ce soit dans un circuit mémoire standard, ou dans une mémoire faisant partie d'un circuit intégré spécifique ("custom circuit", "circuit sur mesure"), tant que le nombre d'entrées et de sorties n'augmente

pas. C'est loin d'être le cas en logique câblée, même lorsque le "câblage" est programmable. En effet, le nombre de portes requis peut varier sensiblement. De plus, une recompilation sera nécessaire, voire même une vérification du comportement temporel dans le cas d'une FPGA (les temps de propagation dépendent du câblage compilé).

3) Les mémoires ont une structure parfaitement modulaire, plus facilement extensible qu'un FPGA.

L'exercice 9 page 61 nous a montré qu'il est possible d'utiliser un compteur (pas n'importe lequel: il doit disposer d'un ENABLE et d'un chargement synchrone) au lieu des traditionnels flip-flops, pour mémoriser l'état de la MSS. En fait, le compteur peut faire plus que simplement mémoriser un code d'état: il peu générer le code de l'état futur à partir d'une simple commande d'incrémentation (ENABLE) Pour cela, il faut bien sûr que le code de l'état futur s'obtienne par simple incrémentation du code de l'état présent.

Si nous choisissons les codes d'état de façon à favoriser l'utilisation d'une compteur, c'est-à-dire de sorte que la séquence voulue s'obtienne en effectuant des incrémentations et le moins possible de chargements, nous ne pourrons plus coder selon la règle n°5. Il faudra donc synchroniser les entrées susceptibles de changer lors du flanc actif de l'horloge dans les états où l'évolution dépend de ces entrées, que nous avons appelé entrées asynchrones (mais à action synchrone). Si ces entrées sont nombreuses, leur synchronisation individuelle va nous coûter beaucoup de flip-flops.

En multiplexant les entrées (ou conditions), nous pouvons les synchroniser seulement au moment où elles vont être testées. Si nous nous limitons à ne tester qu'une entrée ou condition binaire à la fois, un seul flip-flop de synchronisation suffira.

Nous pouvons nous inspirer des idées émises ci-dessus pour concevoir une UC pour le vendeur de billets. La figure 5- 1 nous montre un exemple de réalisation. Nous reconnaissons le multiplexage des entrées, la synchronisation de l'entrée ou condition choisie, effectuée par un flip-flop sur le flanc descendant de l'horloge; le compteur de microprogramme (CTRDIV16) travaillant sur le flanc montant de l'horloge; la mémoire, qui réalise le calcul des commandes de séquencement et le calcul des sorties et de commandes de l'UT, composé de deux PROM32x8. Mais le schéma de la figure suivante soulève aussi plusieurs questions:

1)pourquoi synchroniser les conditions sur le flanc descendant de l'horloge alors que les changements d'état de la MSS se produisent sur le flanc montant?

2)pourquoi multiplexer les conditions d'après une adresse de sélection générée par la mémoire, au lieu d'utiliser directement l'état du compteur comme adresse de sélection?

3)pourquoi utiliser des mutliplexeurs pour calculer des commandes LOAD et COUNT du compteur, au lieu de faire ce calcul dans la mémoire?

En nous limitant à ce que l'évolution de la MSS à partir d'un état quelconque ne dépende que d'une condition binaire au maximum, nous avons pu multiplexer les conditions en fonction de l'état et ne synchroniser que la sortie du multiplexeur. Mais si cette synchronisation se fait sur le même flanc d'horloge qui détermine le passage à l'état suivant de la MSS, la valeur de la condition synchrone apparaitra trop tard pour influencer ce passage à l'état suivant. Nous pouvons soit synchroniser la condition dans la période qui précède son utilisation pour déterminer l'évolution de la MSS, ce qui impliquera certaines contraintes dans l'établissement d'un organigramme ou d'un graphe, soit synchroniser la condition dans la même période où elle sera utilisée pour déterminer l'évolution de la MSS, mais bien évidemment, avant le flanc d'horloge qui provoquera le passage à l'état futur. C'est cette deuxième possibilité que nous avons choisie en utilisant le flanc descendant de l'horloge pour synchroniser la condition. La figure 5-2 nous montre la chronologie des opérations.

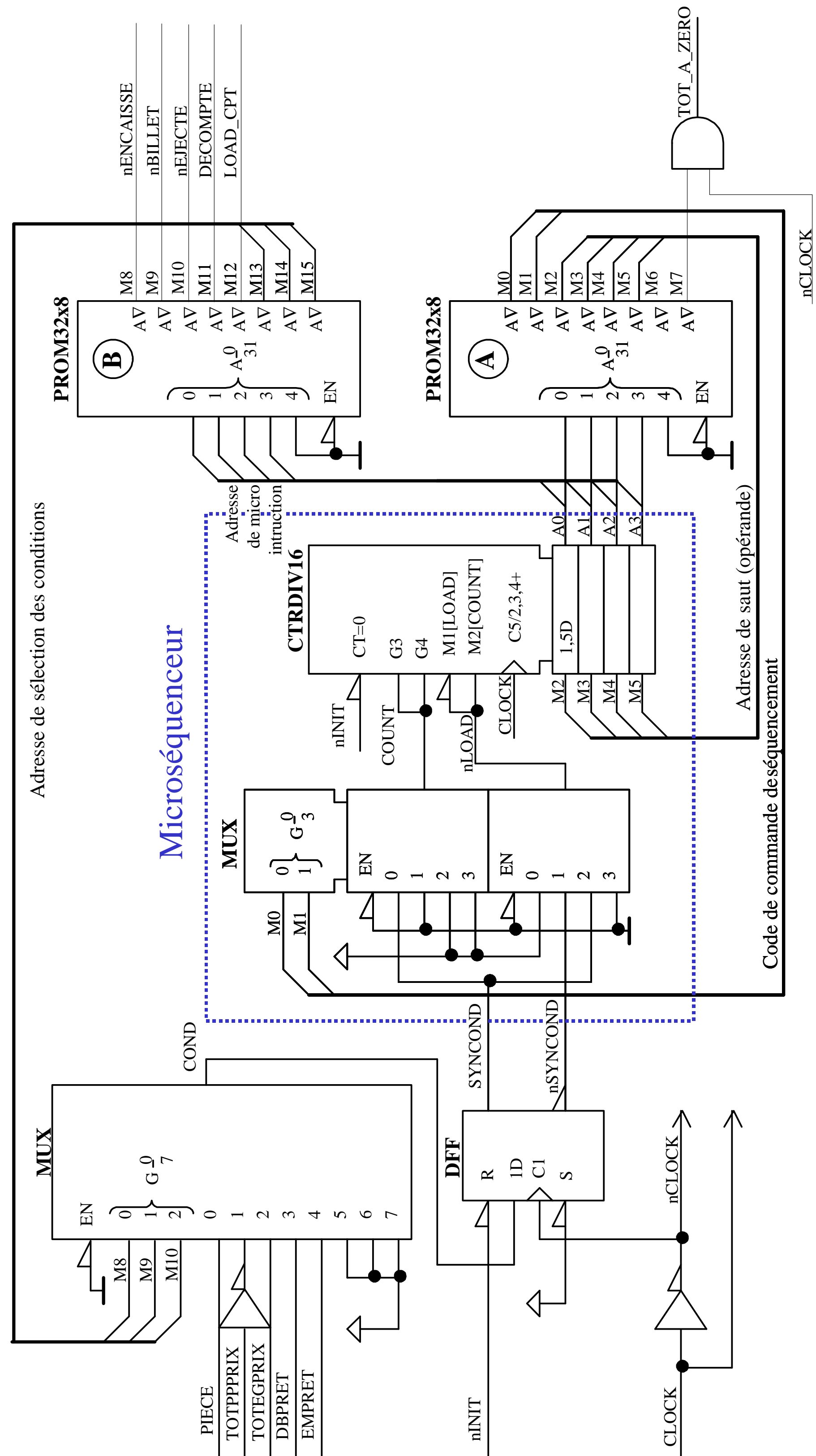


Figure 5- 1 : UC microprogrammée

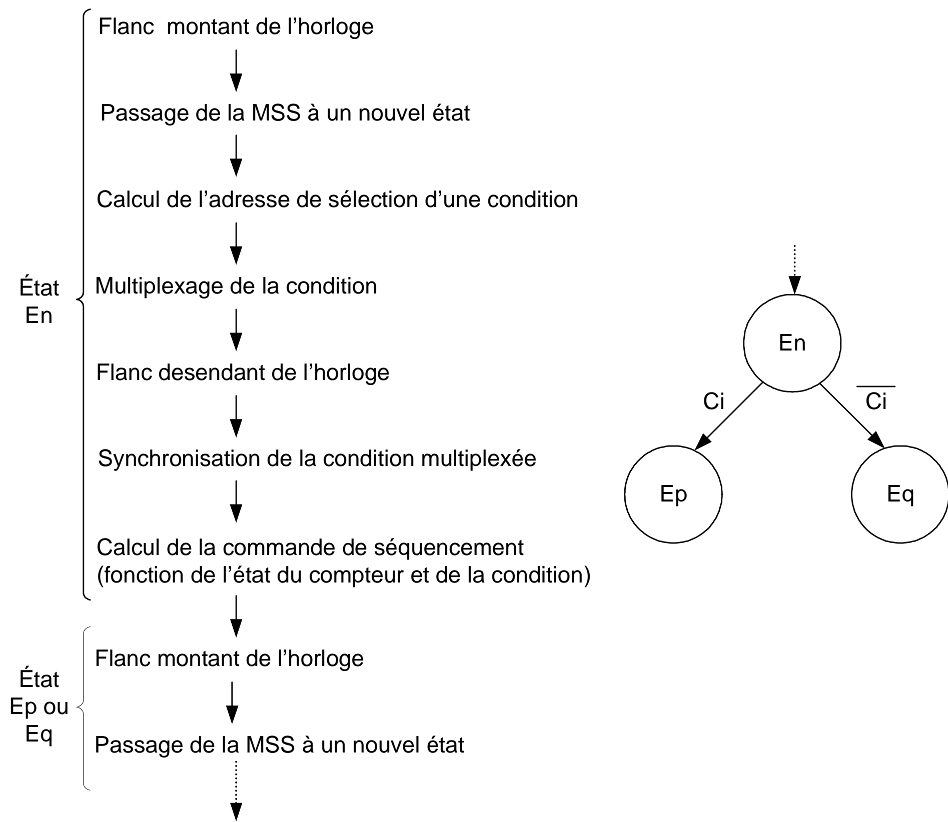


Figure 5- 2 : Sélection du nouvel état

Il n'est pas nécessaire de générer l'adresse de sélection d'une condition à travers la mémoire. L'état du compteur aurait pu être utilisé directement comme adresse de sélection, ce qui nous aurait permis de prendre une mémoire moins large. Ce type de solution est utilisé dans le schéma de la figure figure 5- 3.

Dans une UC microprogrammée, le nombre d'états est généralement beaucoup plus grand que le nombre d'entrées ou de conditions. En utilisant les bits d'état comme variable de sélection pour le multiplexeur de conditions, nous aboutirons alors à un multiplexeur beaucoup plus grand que nécessaire. D'où l'idée d'utiliser la mémoire pour générer l'adresse de sélection à partir de l'état du compteur. Nous pourrons ainsi dimensionner le multiplexeur uniquement en fonction du nombre de conditions distinctes, au prix d'une augmentation de la taille de la mémoire. C'est cette solution générale qui a été choisie dans l'exemple d'UC de la figure 5- 1, bien qu'elle ne soit pas optimale dans ce cas particulier.

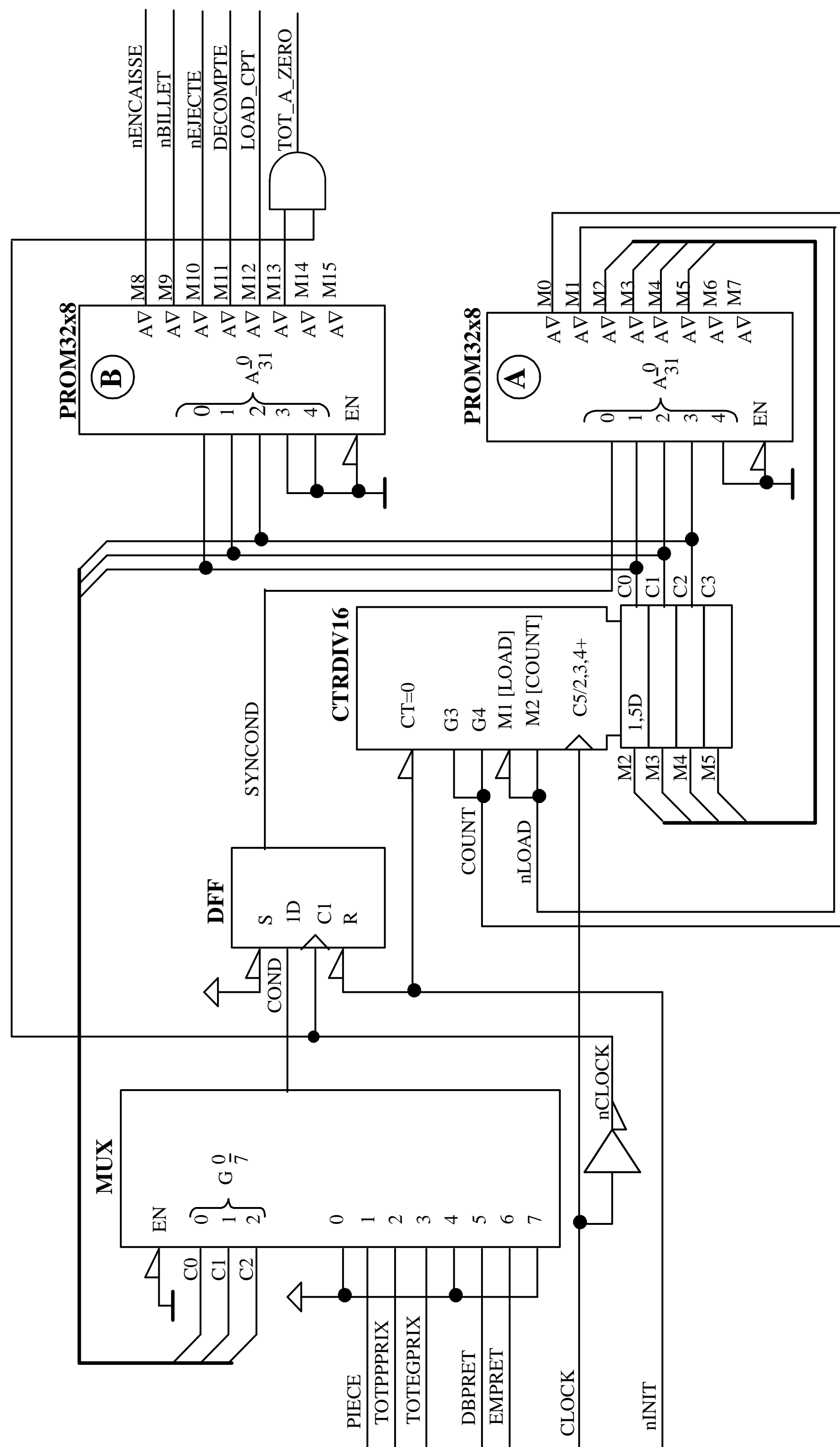


Figure 5- 3 : UC microprogrammée, 2ème solution

Le schéma de la figure 5- 3 nous montre que le calcul de la commande de séquencement peut être réalisée à l'aide d'une mémoire, à partir de l'état de l'UC (état du compteur de microprogramme) et de la condition. La com-

mande de séquencement est constituée ici de 6 bits (6 signaux): M0 (ou CONT) connecté à l'entrée ENABLE du compteur, M1 (ou nLOAD) connecté à l'entrée de mode du compteur, et M2,M3,M4,M5 connectés aux entrées de chargement du compteur.

Mais cette mémoire peut être réduite de moitié, si la prise en compte de la condition se fait séparément, comme c'est le cas dans la réalisation de la figure 5- 1. Les multiplexeurs qui calculent COUNT et LOAD à partir de SYNCOND et des bits M0 et M1 de la mémoire, constituent le bloc de "calcul des commandes du uPC" apparaissent dans le schéma-bloc de la figure 1- 4. Ensemble avec le compteur, ils constituent un micro-séquenceur obéissant aux 4 commandes ci-dessous.

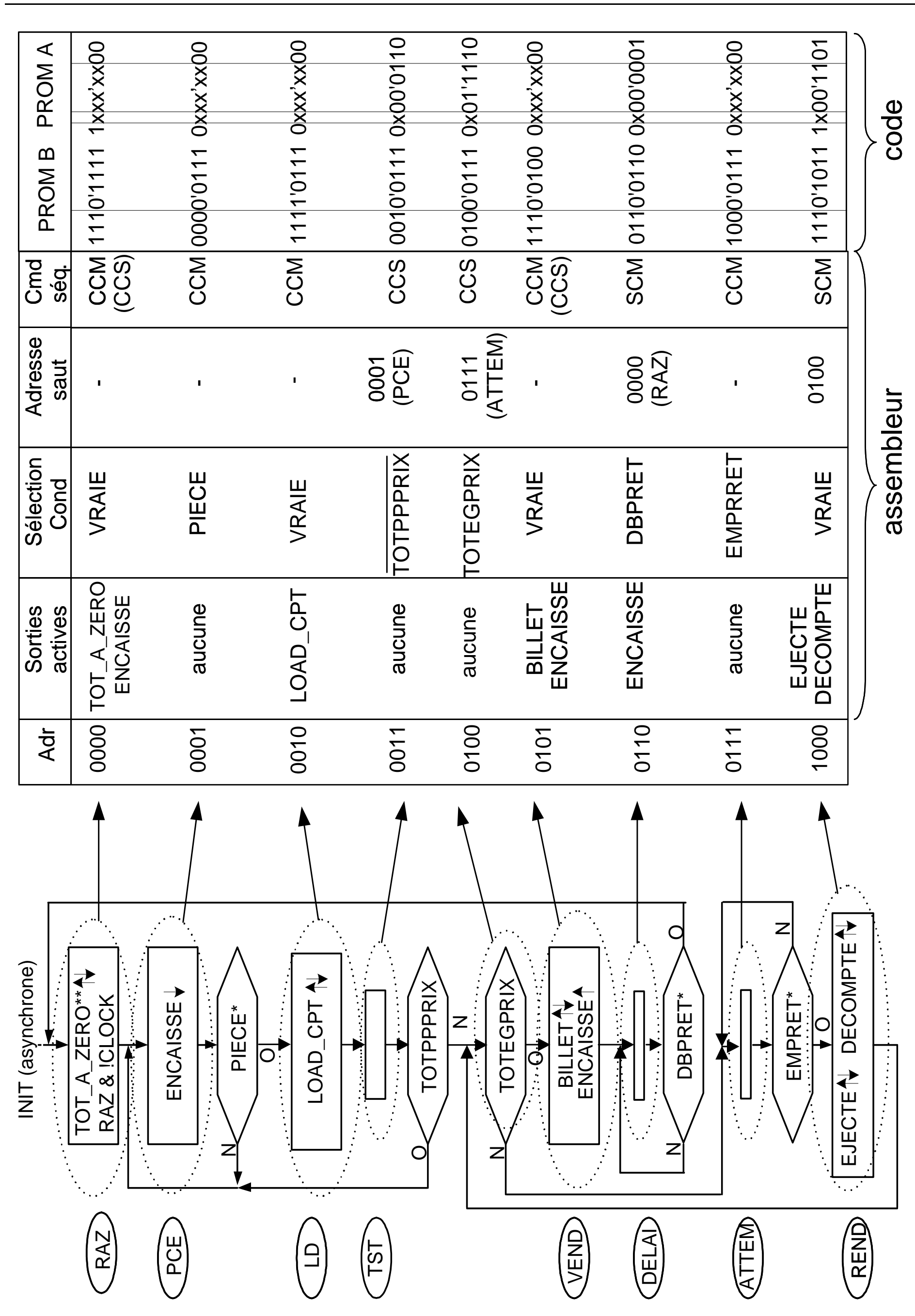
Action	Mnémonique	Code (M1,M0)	COUNT	nLOAD
Compte si Condition ou Maintient µPC <= µ PC+1 si condition vraie, µPC sinon	CCM	00	SYNCOND	1
Saute si Condition ou Maintient µPC <= Adr. (saut) si cond. fausse, µPC sinon	SCM	01	0	nSYNCOND
Compte si Condition ou Saute µPC <= µ PC+1 si cond. vraie, Adr. Sinon	CCS	10	1	SYNCOND
Saute Inconditionnellement µPC <= Adr . (saut inconditionnel)	SI	11	-	0

Table de vérité du décodeur de commandes

Figure 5- 4 : Commandes du micro-séquenceur

Les trois dernières colonnes de la table ci-dessus nous permettent de calculer COUNT et LOAD en fonction des codes choisis arbitrairement pour chacune de ces quatre opérations de séquencement que nous avons abrégées CCM, SCM, CCS, et SI. Dans ces mnémoniques "S" est mis pour saute puisqu'un chargement a justement pour but d'effectuer un saut dans la séquence naturelle du comptage.

Pour compléter le développement de l'UC de la figure 5- 1, il ne nous reste plus qu'à traduire l'organigramme détaillé de la figure 4- 13, en une liste de programmation pour les mémoires.



Adr	Sorties actives	Sélection Cond	Adresse saut	Cmd séq.	PROM B	PROM A
0000	TOT_A_ZERO ENCAISSE	VRAIE	-	CCM (CCS)	1110'1111	1xxx'xx00
0001	aucune	PIECE	-	CCM	0000'0111	0xxx'xx00
0010	LOAD_CPT	VRAIE	-	CCM	1111'0111	0xxx'xx00
0011	aucune	$\overline{\text{TOTPPPRIX}}$	0001 (PCE)	CCS	0010'0111	0x00'0110
0100	aucune	TOTEGPRIX	0111 (ATTEM)	CCS	0100'0111	0x01'1110
0101	BILLET ENCAISSE	VRAIE	-	CCM (CCS)	1110'0100	0xxx'xx00
0110	ENCAISSE	DBPRET	0000 (RAZ)	SCM	0110'0110	0x00'0001
0111	aucune	EMPRRET	-	CCM	1000'0111	0xxx'xx00
1000	EJECTE DECOMPTE	VRAIE	0100	SCM	1110'1011	1x00'1101

Figure 5- 5 : Passage de l’organigramme au microprogramme

Nous obtenons ainsi la table de la figure 5- 5.

Chaque ligne de cette table constitue ce que l’on appel à une microinstruction. La traduction d’un organigramme ou d’un graphe en une liste de microinstructions constitue un microprogramme.

Le format d’une microinstruction indique la répartition de l’information qu’elle contient, en diverses plages de bits. Dans l’exemple de la figure 5-1 et de la figure 5- 5, le format que nous avons choisi est le suivant:

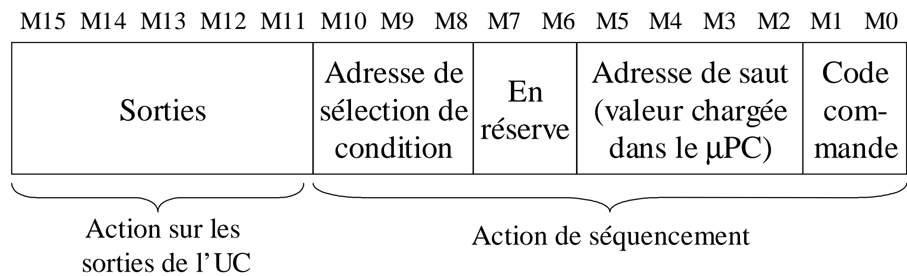


Figure 5- 6 : Format d'une microinstruction

En pratique, une microinstruciton est codée en un seul mot mémoire et exécutée en une seule période d'horloge, bien que rien n'empêche que l'on conçoive un microséquenceur exécutant des microinstructions mutli-mots et/ou en plusieurs périodes d'horloge.

Si nous examinons l'UC de la figure 5- 1 et celle de la figure 5- 3, toutes deux pouvant être appelées "UC microprogrammée", nous constantons qu'il est impossible d'en donner une définition générale. Nous nous contenterons donc de retenir la caractéristique principale: le comportement d'une UC microprogrammée est essentiellement déterminé par son microprogramme, soit une suite de microinstructions (commandes) stockées dans une mémoire.

Le schéma-bloc d'une UC microprogrammée le plus couramment utilisé est celui de la figure 1- 4.

5-1 Exercices

1) Dans l'UC de la figure 5- 1, nous décidons de synchroniser la condition sur le flanc montant de l'horloge, au lieu de le faire sur le flanc descandant. Modifier en conséquence l'organigramme détaillé et le microprogramme.

2) Modifier le micro séquenceur de la figure 5- 1, de façon à ce qu'il exécute le jeu de commandes de séquencement suivant:

action	mnémonique	code
Compte inconditionnellement	INC	00
Compte si la condition est remplie, maintient l'état sinon	WAIT	01
Saute si la condition est remplie, compte sinon	JUMPT	10
Compte si la condition est remplie, saute sinon	JUMPF	11

3) Adapter le microprogramme de la figure 5- 5 au micro-séquenceur développé dans l'exercice 2.

4) Établir les tables de programmation des mémoires de la figure 5- 3, de façon à ce que cette UC fonctionne selon l'organigramme de la figure 4-13.

5) Dans l'UC de la figure 5- 3, générer les sorties à l'aide d'un décodeur en lieu et place de la mémoire.

6) Adapter le micro-séquenceur de la figure 5- 1 de façon à le rendre modulaire, c'est-à-dire extensible par tranches de 4 bits à l'aide de modules identiques. Décrire ce micro-séquenceur en VHDL.

5-2 Minimisation de la mémoire de microprogramme

Le format de microinstruction de la figure 5- 6 fait apparaître 4 plages, dédiées respectivement au code de la commande de séquencement, à l'adresse de saut, en faisant abstraction des bits pas utilisés. Le microprogramme de la figure 5- 5 nous montre que la seule plage qui soit spécifiée pour chaque microinstruction est celle du code de la commande. Les autres plages sont rarement utilisées toutes en même temps. Par exemple, l'instruction CCM n'utilise pas la plage d'adresse de saut, alors que l'instruction SI n'utilise pas la plage de sélection de condition.

Nous pouvons réduire la largeur de la mémoire de microprogramme en réduisant le nombre de plages de nos microinstructions. Par exemple, nous pouvons imaginer un jeu de microinstructions à deux plages seulement: l'une pour le code de la commande de séquencement, l'autre pour une opérande (ou paramètre) qui sera utilisé tantôt comme adresse de saut, tantôt comme adresse de sélection de condition, et tantôt pour générer les sorties.

Lorsque chaque plage d'un jeu de microinstructions ne remplit qu'une seule fonction, comme c'était le cas dans le format de la figure 5- 6, nous parlerons de microprogrammation horizontale. Par contre, si une même plage remplit diverses fonctions selon le code de la commande, nous parlerons de microprogrammation verticale.

Le jeu d'instructions de la figure 5- 7 va nous fournir un exemple de microprogrammation verticale. Il s'agit d'un jeu d'instruction à un seul opérande.

Action	Mnémonique	Code (M1,M0)	Opérande
µPC => µPC+1 si cond. vraie µPC si cond. fausse	WAIT	00	SYNCOND
Saut inconditionnel	JUMP	01	0
µPC => µPC+2 si cond. Vraie µPC+1 si cond. fausse	SKIP	10	1
Modifie les valeurs des sorties et incrémente le µPC	OUT	11	-

Figure 5- 7 : Jeu d'instructions du micro-équenceur MISEQV0

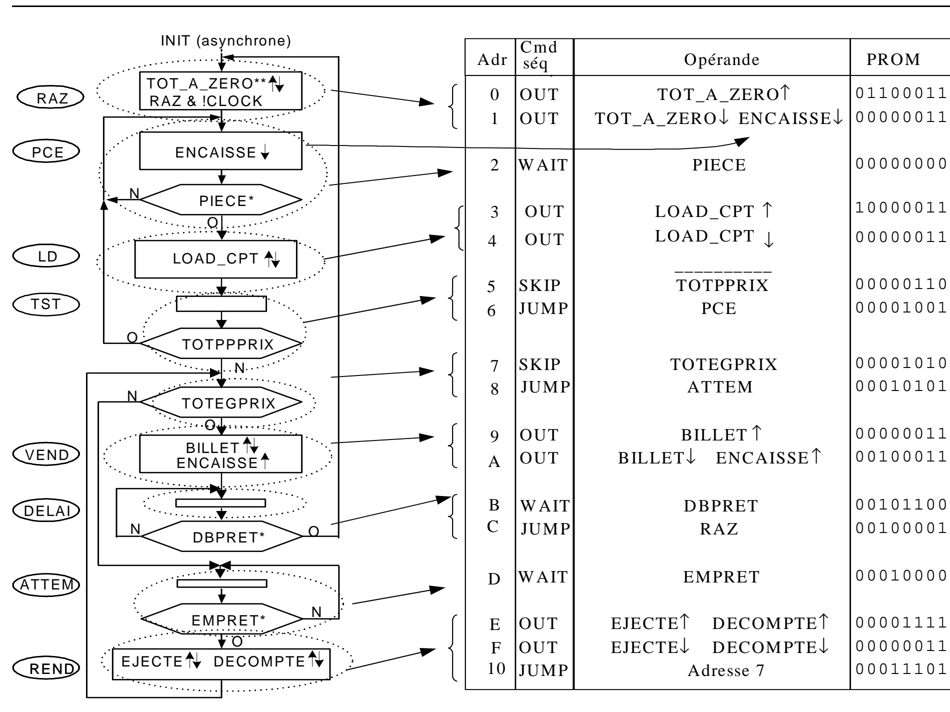
La microinstruction WAIT est similaire à la microinstruction CCM que nous avons vue dans la figure 5- 4, à ceci près que nous ne spécifierons pas de valeurs de sortie avec WAIT, puisque nous nous limitons à un seul opérande et que celui-ci doit donc être l'adresse de sélection de la condition.

Pour modifier l'état d'une ou plusieurs sorties, nous utiliserons la microinstruction OUT. Entre deux modifications, il faudra maintenir l'état à l'aide de latches ou de flip-flop.

La microinstruction JUMP correspond à la microinstruction SI (aux sorties près). Par contre, dans le jeu ci-dessus, nous n'avons aucun saut conditionnel comme CCS ou SCM. Pour obtenir un saut conditionnel nous devons utiliser deux microinstructions : SKIP suivie de JUMP. Par exemple : CCS si COND devient SKIP si not COND suivie de JUMP.

Ce petit jeu de microinstructions nous permet d'implémenter n'importe quelle séquence, pourvu que nous acceptions de réaliser certains états de l'organigramme ou du graphe en plusieurs microinstructions, donc en plusieurs périodes d'horloge.

La figure 5- 8 nous montre le microprogramme correspondant à l'organigramme de la figure 4- 13, pour un micro-séquenceur MISEQV0 exécutant le jeu de microinstructions de la figure 5- 7.



Adr	Cmd séq	Opérande	PROM
0	OUT	TOT_A_ZERO↑	01100011
1	OUT	TOT_A_ZERO↓ ENCAISSE↓	00000011
2	WAIT	PIECE	00000000
3	OUT	LOAD_CPT ↑	10000011
4	OUT	LOAD_CPT ↓	00000011
5	SKIP	TOTPPRIX (surligné)	00000110
6	JUMP	PCE	00001001
7	SKIP	TOTEGPRIX	00001010
8	JUMP	ATTEM	00010101
9	OUT	BILLET↑	00000011
A	OUT	BILLET↓ ENCAISSE↑	00100011
B	WAIT	DBPRET	00101100
C	JUMP	RAZ	00100001
D	WAIT	EMPRET	00010000
E	OUT	EJECTE↑ DECOMPTE↑	00001111
F	OUT	EJECTE↓ DECOMPTE↓	00000011
10	JUMP	Adresse 7	00011101

Figure 5- 8 : Microprogramme pour un micro-séquenceur MISEQV0

Comme précédemment, ce microprogramme a d'abord été établi en mnémoniques, y-compris pour les adresses de saut (PCE, ATTEM, etc), puis traduit en code à la main.

Les conventions utilisées sont les suivantes:

-Les bits de code non définis sont laissés à 1 (bits mémoire non programmés); ainsi, pour une microinstruction OUT, le bit M7 est systématiquement laissé à 1; il en va de même pour les bits M7, M6 et M5 dans le WAIT et le SKIP, M7 et M6 dans le JUMP.

-Seules les sorties qui changent sont mentionnées dans la microinstruction en mnémoniques.

Dans cette réalisation nous n'avons pas diminué la capacité utile de la mémoire. En fait elle a même augmenté d'un bit par rapport à la solution à microprogrammation horizontale de la figure 5- 5 (15 mots de 7 bits au lieu de 8 mots de 13 bits). Par contre, le gain en largeur est important, puisqu'il nous permet de n'utiliser plus qu'un seul circuit intégré au lieu de deux.

La figure 5- 9 nous montre le schéma de notre UC réalisée à l'aide de MISEQV0. On y remarque l'apparition d'un registre pour maintenir l'état des sorties entre deux microinstructions OUT. Comme son symbole l'indique, ce registre ne peut changer d'état que si son entrée G2 est active. Nous avons donc connecté cette entrée G2 à la sortie OUTEN de MISEQV0, qui n'est active que lors de l'exécution d'une microinstruction OUT.

Le micro-séquenceur MISEQV0 peut être décrit à l'aide du langage VHDL en vue de son intégration dans un PLD. Il comporte une entrée de reset, 5 entrées pour l'adresse de saut S4..0, 2 entrées pour le code de la microinstruction c1..0, une entrée de condition et une entrée d'horloge. Les sorties A4..0 sont les bits d'adresse du microprogramme générés par le uPC (micro-program counter). La sortie OUTEN est active lorsqu'une instruction OUT est exécutée. L'ensemble de l'UC, basée sur le MISEQV0, peut-être intégrée à l'aide d'un PLD, par exemple un circuit EPM7128S.

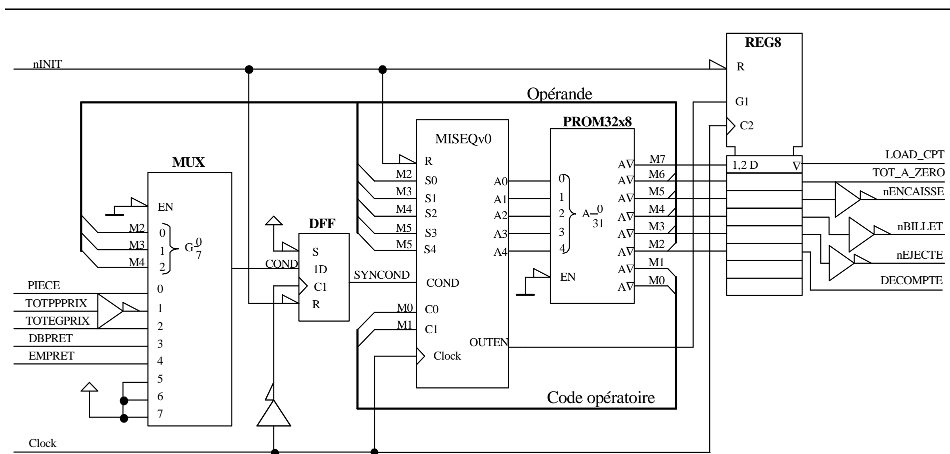


Figure 5- 9 : UC basée sur le micro-séquenceur MISEQV0

A part l'utilisation d'un même champ de la microinstruction pour remplir alternativement diverses fonctions, il y a trois autres méthodes couramment utilisées pour diminuer la largeur ou la capacité de la mémoire de microprogramme. La première consiste à rétrécir le champ d'adresse de saut en se limitant à ne sauter que vers certaines adresses. Par exemple : ne sauter que vers des adresses paires, ce qui évite de spécifier le bit de poids faible puisqu'il sera toujours à zéro.

La deuxième méthode est très utilisée. Elle consiste à coder le contenu d'un champ. Cela a un intérêt lorsque le nombre de bits nécessaire au codage des combinaisons utiles des signaux composant ce champ est plus petit que le nombre de signaux lui-même. Encore faut-il que le décodeur qu'il faudra alors introduire ne vienne pas réduire à néant l'économie réalisée sur la mémoire. Nous verrons un exemple de codage au paragraphe suivant.

La troisième méthode est une extension du codage. Mais au lieu de coder l'une ou l'autre plage d'une microinstruction, on attribue un code à chaque microinstruction distincte réellement utilisée dans un microprogramme donné. Ce codage est alors effectué à l'aide d'une deuxième mémoire appelée mémoire de nanoprogramme. Cette technique, illustrée à la figure 5- 10 est appelée "nanoprogrammation".

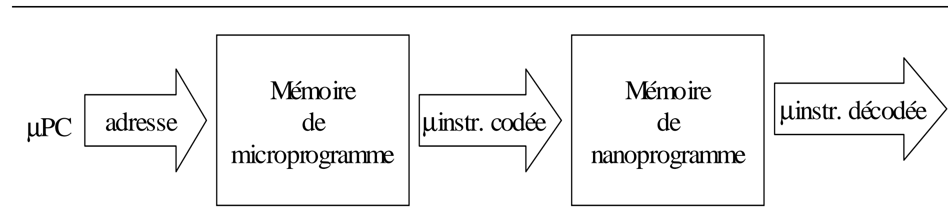


Figure 5- 10 : Nanoprogrammation

5-3 Codage des sorties

Dans une MSS complexe, le nombre de sorties changeant simultanément à un instant donné est beaucoup plus faible, généralement, que le nombre total de sorties. Souvent, il est même possible de se contenter de changer une seule sortie à la fois. Dans ce cas, nous pourrons quelquefois réduire encore la largeur de la mémoire de microprogramme en codant les sorties.

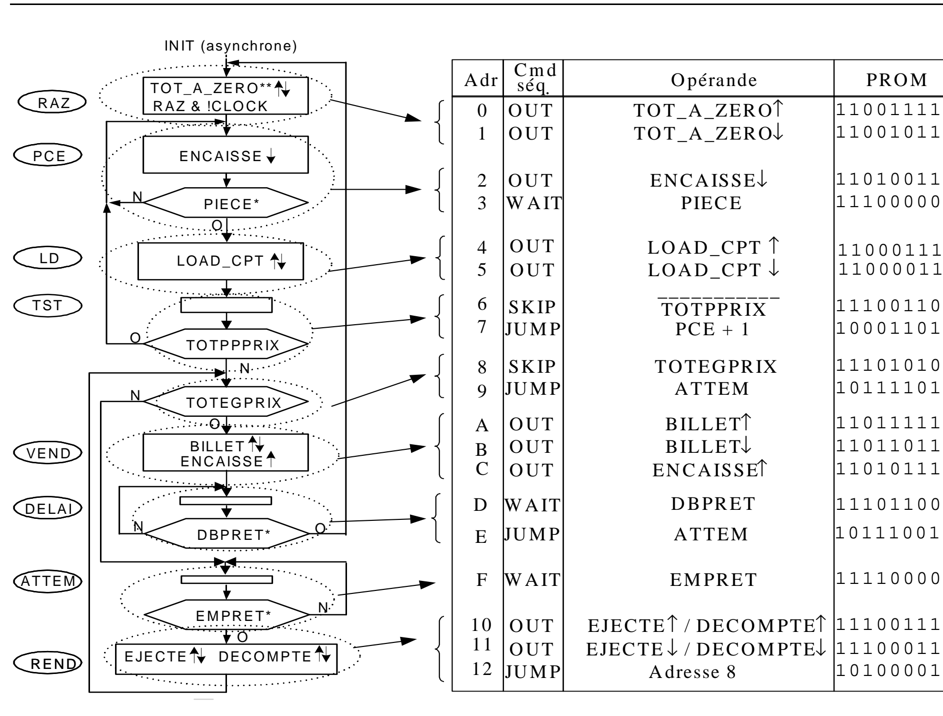
Les 6 sorties de notre UC peuvent être codées arbitrairement de la façon suivante : TOT_A_ZERO = (000), ENCAISSE = (001), BILLET = (010), EJECTE = (011), DECOMPTE = (100) et LOAD_CPT(101). Il suffit ainsi de 3 bits dans la microinstruction OUT pour désigner la sortie que nous désirons changer, au lieu de 6 bits si nous désirons pouvoir les changer toutes en même temps. Un PLD nous permettrait d'intégrer facilement le décodeur et les flip-flops de mémorisation. Mais nous pouvons aussi utiliser un registre transparent dont chaque bit peut être mis individuellement à 1 ou à 0. Dans ce cas, un bit de la microinstruction OUT sera utilisée pour indiquer la valeur de la sortie, comme nous le montre le schéma de la figure 5- 11 (M2 joue ce rôle ici).

Le signal OUTEN est utilisé pour commander l'activation de la mémorisation dans un bit du registre (G9). Ainsi le registre BAR8 ne sera modifier que lors de l'exécution d'une microinstruction OUT.

Les trois bits de la mémoire (M5..3), qui est une partie de l'opérande, permet de sélectionner la sortie à mettre à jour. Le bit M2 de la mémoire indique la valeur à mémoriser dans la sortie sélectionnée.

Figure 5- 11 : UC basée sur MISEQV0 avec latch adressable bit par bit

Le microprogramme de la figure 5- 12 est facilement adapté à partir de celui de la figure 5- 8, pour tenir compte du schéma utilisant les sorties codées (figure 5- 11).



Adr	Cmd séq.	Opérande	PROM
0	OUT	TOT_A_ZERO↑	11001111
1	OUT	TOT_A_ZERO↓	11001011
2	OUT	ENCAISSE↓	11010011
3	WAIT	PIECE	11100000
4	OUT	LOAD_CPT ↑	11000111
5	OUT	LOAD_CPT ↓	11000011
6	SKIP	$\overline{\text{TOTPPRIX}}$	11100110
7	JUMP	PCE + 1	10001101
8	SKIP	TOTEGPRIX	11101010
9	JUMP	ATTEM	10111101
A	OUT	BILLET↑	11011111
B	OUT	BILLET↓	11011011
C	OUT	ENCAISSE↑	11010111
D	WAIT	DBPRET	11101100
E	JUMP	ATTEM	10111001
F	WAIT	EMPRET	11110000
10	OUT	EJECTE↑ / DECOMPTE↑	11100111
11	OUT	EJECTE↓ / DECOMPTE↓	11100011
12	JUMP	Adresse 8	10100001

Figure 5- 12 : Microprogramme utilisant les sorties codées

Le fait de n'agir que sur un seul bit de sortie à la fois a eu pour conséquence de rallonger le microprogramme. C'est bien naturel puisque nous exécutons en série le travail que nous avions exécuté en parallèle dans le schéma de la figure 5- 10 et le microprogramme de la figure 5- 9. Afin de gagner quelques microinstructions, nous avons généré EJECTE et DECOMPTE avec le même latch. Par contre le compteur de micro-programme du MISEQV0 doit être étendu à 5 bits, voir la figure 5- 11.

Puisqu'à partir du code d'une sortie nous sélectionnons un latch ou un flip-flop dans lequel sera mémorisé l'état de cette sortie, nous parlerons de l'adresse d'une sortie plutôt que d'appeler cela son "code".

5-4 Exercices

1)Ajouter un sélecteur de polarité de la condition dans le schéma de la figure 5- 9. Proposer un format pour les microinstructions WAIT et SKIP de la table de la figure 5- 7, tenant compte du choix de la polarité de la condition.

2)Traduire le graphe partiel ci-dessous en un bout de microprogramme utilisant le jeu de microinstructions de la table de la figure 5- 7.

3)Pour raccourcir le microprogramme de la figure 5- 12, nous décidons d'optimiser le circuit de sortie de la façon suivante.

a) les sorties générant une brève impulsion sont codées; le circuit de sortie doit les décoder sans transitoires, sans les mémoriser.

b) les sorties dont les impulsions durent plusieurs périodes d'horloge ne seront pas codées; le circuit de sortie doit les mémoriser.

Concevoir le circuit de sortie (il remplacera le SRG8 ou le latch 8 bits) et le décrire en VHDL pour un PLD à choisir. Adapter le microprogramme de notre UC en conséquence.

4)Choisir un PLD approprié et décrire en VHDL le comportement d'un microséquenceur MISEQV1 ayant les mêmes caractéristiques que MISEQV0, mais qui soit cascadable (dont le uPC puisse être étendu).

5)Concevoir un circuit, que nous appellerons BAR8V0 (Bit Adressable Register, 8 bits), pour remplacer le latch 8 bits. Il doit comporter des flip-flops et non des latches. Prévoir deux entrées de sélection du circuit, qui doivent être actives pour que le circuit réagisse à une microcommande OUT.

6)Concevoir un microséquenceur MISEQV2 générant 5 bits d'adresse. Son jeu d'instructions, a 1 opérande, est le suivant:

Action	Mnémo-nique	Code et format 8b.	Opérandes
Mémorise la condition spécifiée par l'opérande, et incr. le compteur de microinstructions. Rem: i indique si la condition spécifiée doit être inversée (i=1) ou no (i=0) avant mémorisation	Mem	cccc i011	cccc = adresse de la condition i = inversion de la condition
Incrémente le compteur de microinstructions si la condition mémorisée est à 1, n'incrémente pas dans le cas contraire; de plus cette instruction mémorise la condition comme Mem.	Wait	cccc i111	cccc = adresse de la condition i = inversion de la condition
Charge le compteur de micro-instructions avec l'adresse spécifiée par l'opérande, si la condition mémorisée est à 0. Incrémente ce compteur dans le cas contraire. La condition mémorisée est mise à 0 dans tous les cas.	Jumpf	--aa aaa0	aaaaa = adresse de saut
Met à jour les sorties. Active le signal Outen, utilisé pour commander les décodeurs de sortie, et remet à 0 la mémorisation de la condition; le compteur de microprogramme est incrémenté.	Out	ssss sv01	*Format indicatif de l'opérande dépend décodeurs :* sssss = adresse de la sortie concernée et v = valeur de la sortie

Le Reset met le compteur de microprogramme à 0, la condition mémorisée à 0, et empêche l'activation de Outen.

5-5 Sous-programmes et interruptions

Lorsqu'une même suite de microinstructions apparaît plusieurs fois dans un microprogramme, il peut être avantageux de la transformer en un sous-microprogramme: cela améliorera la lisibilité et réduira la longueur du microprogramme, exactement comme une procédure en C ou Ada.

Un sous-microprogramme est une suite de microinstructions dont l'exécution est déclenchée à l'aide d'un saut spécial que l'on nomme généralement "appel" (call, subroutine branch), et qui se termine par un retour à la microinstruction suivant l'appel.

La figure 5- 13 illustre l'utilisation d'un microprogramme constitué des microinstructions A,B,C et RET.

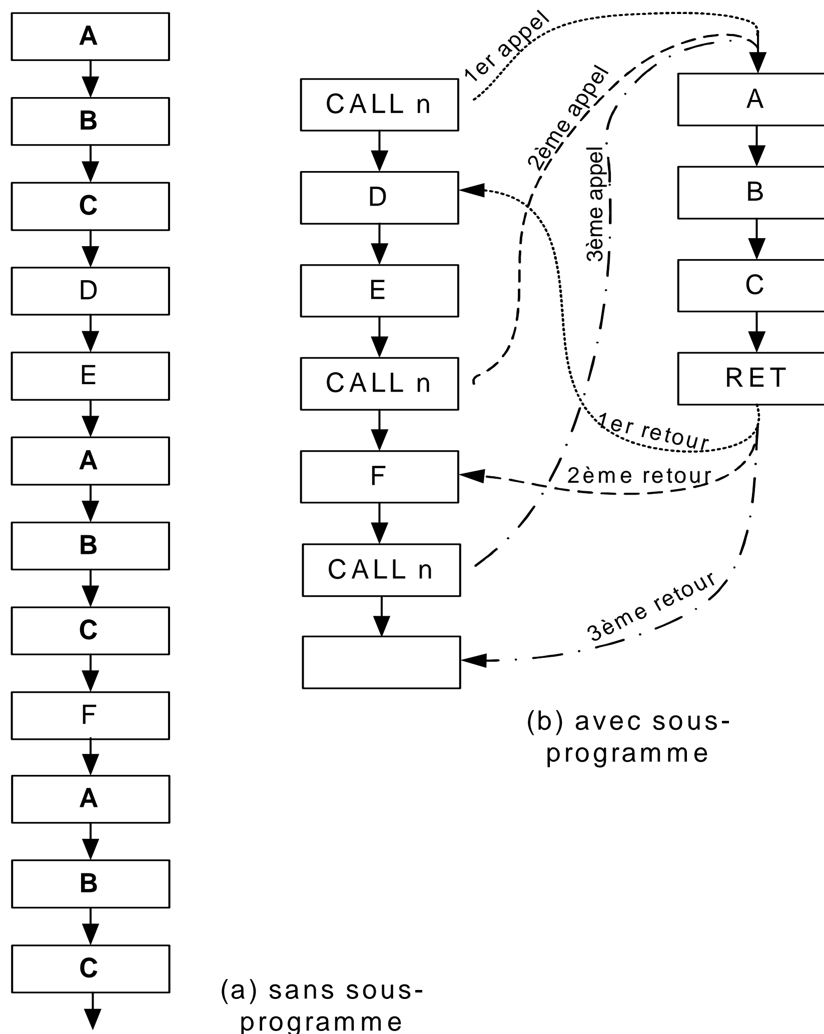


Figure 5- 13 : Microprogramme constitué des microinstructions A,B,C et RET

Pour un résultat final identique, le microprogramme (b) utilise une microinstruction de moins que le micro programme (a), soit 11 au lieu de 12. Plus la longueur du sous-microprogramme et le nombre d'appels seront grand, plus grande sera l'économie.

Si l'utilisation de sous-microprogrammes permet de réduire la longueur du microprogramme, elle en augmente par contre le temps d'exécution, en règle générale. Dans l'exemple de la figure 5- 13, le microprogramme (a) est exécuté en 12 périodes d'horloge alors qu'il en faut 19 pour exécuter le microprogramme (b).

Pour que notre microséquenceur supporte des sous-microprogrammes, il suffit de lui ajouter deux microinstructions, CALL et RET, ainsi qu'un registre pour garder l'adresse de retour. Il faut en effet que lors de l'exécution du CALL l'adresse de retour soit mémorisée, puisqu'elle varie d'un appel à l'autre et ne peut donc pas être spécifiée dans la microinstruction RET.

Si un sous-microprogramme peut en appeler un autre, nous devrons mémoriser non pas une mais deux adresses de retour, qui seront utilisées dans l'ordre inverse de celui dans lequel elles auront été mémorisées. Pour plusieurs niveaux de sous-microprogrammes, tout se passe comme si les adresses de retour étaient déposées sur une pile au fur et à mesure des appels, puis retirées du dessus de la pile au fur à mesure des retours. Rien d'étonnant donc à ce que la structure de mémoire utilisée pour empiler les adresses de retour s'appelle une "pile" (stack). Une telle structure peut être réalisée comme un registre à décalages, comme nous le verrons en exercice, mais le plus souvent elle est réalisée à l'aide d'un compteur-décompteur et d'une mémoire à accès aléatoire (RAM).

Dans certains cas, il peut être nécessaire d'interrompre le fonctionnement normal d'une MSS pour traiter en urgence un événement particulier. Par exemple : événement catastrophique du genre chute de tension ou erreur de fonctionnement, événement de synchronisation tel que l'intant d'échantillonage dans un régulateur échantillonnée, etc.

Cette interruption, qui déclenche l'exécution d'un bout de microprogramme particulier à la fin duquel le microséquenceur doit retourner à l'exécution du microprogramme interrompu, peut fonctionner de façon presque identique à l'appel d'un sous-programme. Lors de l'interruption , l'adresse de retour est mise sur la pile et l'adresse de sous-microprogramme d'interruption est mise dans le uPC. Cette dernière peut être fixe, ou au contraire dépendre de l'événement qui a déclanché l'interruption, auquel cas

elle est généralement envoyée au microséquenceur par le demandeur d'interruption, au travers des entrées d'adresse de saut.

La figure 5- 14 nous montre le schéma-bloc d'une réalisation possible d'un microséquenceur pouvant exécuter des sous-microprogrammes, et pouvant être interrompu. L'interruption est obtenue ici en forçant une microinstruction CALL au lieu de la microinstruction qui allait être exécutée. L'adresse de cette microinstruction est sauvée sur la pile en exécutant le CALL, puis sera rechargée dans le compteur de microprogramme en exécutant l'instruction de retour d'un sous-microprogramme d'interruption, IRET. Si le CALL résulte de l'exécution normale du microprogramme et non pas d'une interruption, l'adresse sauvée sur la pile devra être incrémentée avant d'être chargée dans le compteur de microprogramme, à l'aide d'une instruction de retour distincte, RET.

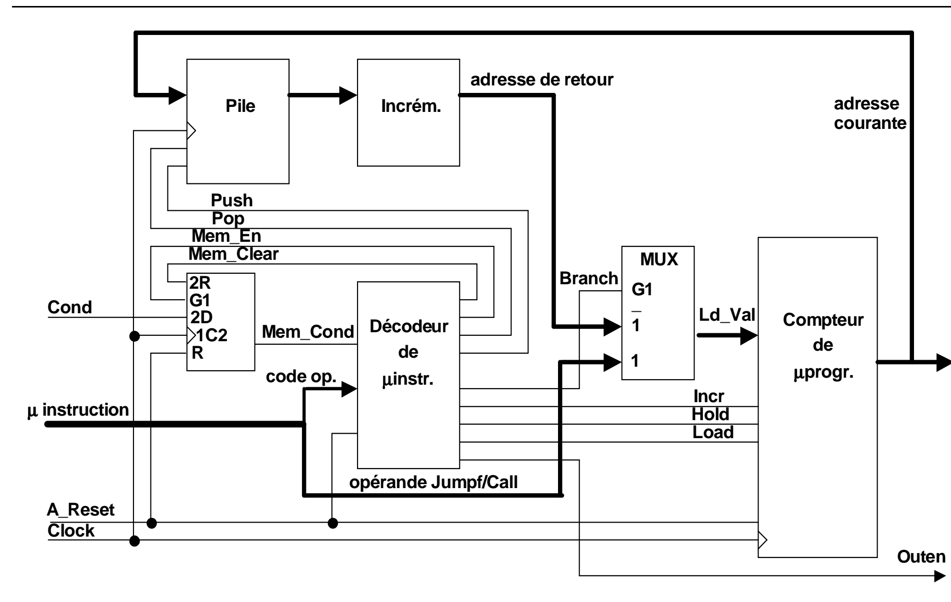


Figure 5- 14 : Microséquenceur pouvant exécuter des sous-programmes

Le microséquenceur de la figure 5- 14 peut être réalisée dans un circuit EPM7128S, avec 5 bits d'adresse et 2 niveaux de pile. Les pages suivantes listent sa description en VHDL, implémentant un jeu d'instruction qui englobe celui de MISEQV2 (voir série d'exercices précédente). Ce nouveau circuit, que nous appellerons MISEQV3, utilise cependant des codes différents de MISEQV2 : le codage des microinstructions influence bien sûr la complexité du circuit.

Action	Mnémo-nique	Code et format 8b.	Opérandes
Mémorise la condition, inversée ou non uPC <= uPC+1	*Mem*	cccci011	cccc = adr. cond. i = invers. cond.
Attend que la cond. mémorisée soit vraie, mémorise la condition à chaque clock (inv. ou non) uPC <= uPC+1 si MemCond vrai, sinon uPC	*Wait*	cccci001	cccc = adr. cond. i = invers. cond.
Saute si la condition mémorisée est fausse, met à 0 la condition mémorisée (dans tous les cas) uPC <= Adr. si MemCond fausse, sinon uPC+1	*Jumpf*	aaaaa000	aaaaa = adresse de saut
Saute à l'adresse spécifiée et sauve l'adresse actuelle sur la pile. La condition mémorisée est maintenue uPC <= Adr. ; Pile <= uPC (val. actuelle)	*Call*	aaaaa100	aaaaa = adresse de saut
Active la sortie /Outen de MISEQv2, met à 0 la condition mémorisée uPC <= uPC+1	*Out*	ssssv010 (par ex.)	ssss = adr sortie concernée v = valeur sortie
Charge uPC avec l'adresse fournie par la pile+1. Fonctionne comme Out sauf que la condition mémorisée n'est pas modifiée uPC <= AdressePile +1	*Ret*	ssssv110 (par ex.)	ssss = adr sortie concernée v = valeur sortie
Idem que Ret, mais l'adresse prise sur la pile n'est pas incrémentée uPC <= AdressePile	*IRet*	ssssv111 (par ex.)	ssss = adr sortie concernée v = valeur sortie
Opcode non utilisé	*NUsed*	-----101	

Figure 5- 15 : Jeu d'instructions du micro-équenceur MISEQV0

Description du MISEQV3 dans HDL Designer.

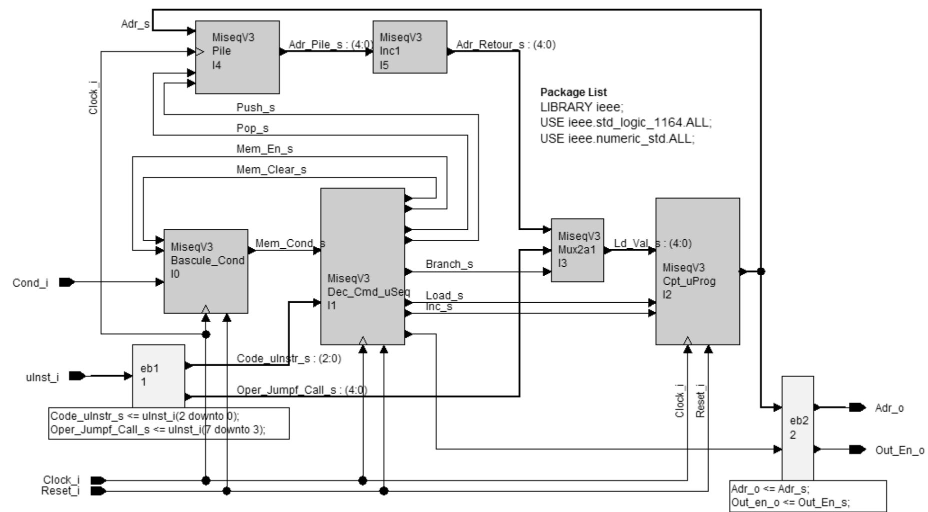


Figure 5- 16 : Vue structurelle HDL Designer du MISEQV3

Voici la description VHDL du compteur de micro-programme.

```
entity Cpt_uProg is
    port(
      Ld_Val_s  : in     Std_Logic_Vector (6 downto 0);
       Clock_i   : in     std_logic;
       Inc_s     : in     Std_Logic;
       Load_s    : in     Std_Logic;
       Reset_i   : in     std_logic;
      Adr_s     : out    Std_Logic_Vector (6 downto 0)
       );
end entity Cpt_uProg ;

architecture Comport of Cpt_uProg is
  signal Cpt_Pres_s, Cpt_Fut_s : unsigned(Adr_s'range);
begin

  --| Decodeur d'etat futur |--------------------------
  Cpt_Fut_s <= unsigned(Ld_Val_s)
                 when Load_s = '1' else -- chargement
              Cpt_Pres_s + 1
                 when Inc_s  = '1' else -- comptage +1
              Cpt_Pres_s;               -- maitient

  --| Process memoire |-------------------------------
  process(Clock_i, Reset_i)
  begin
    if (Reset_i = '1') then
      Cpt_Pres_s <= (others => '0');
    elsif Rising_Edge(Clock_i) then
      Cpt_Pres_s <= Cpt_Fut_s;
    end if;
  end process;
```

Chapitre 6

Unité de traitement universelle

Pour réaliser une UT spécialisée, nous avons cherché à identifier dans l'algorithme de fonctionnement les fonctions réalisables à l'aide de circuits standards spécifiques. Par exemple, un comptage avec un compteur, une addition avec un additionneur, etc. Le choix de ces circuits et leur interconnexion sont étroitement liés au fonctionnement que l'on désire obtenir. Il est donc difficile d'adapter une UT spécialisée aux besoins d'un autre système.

L'utilisation de circuits programmables diminue les problèmes de câblage, améliore la souplesse et la standardisation, mais n'évite pas de refaire la conception et le test d'une UT spécialisée pour chaque nouveau système.

Une UT universelle est une UT qui n'est pas spécifiquement conçue pour un système particulier, mais au contraire étudiée pour satisfaire aux besoins les plus généraux, ce qui permet sa standardisation. Cet avantage est obtenu en utilisant un circuit permettant de réaliser une à toutes les fonctions élémentaires indispensables, et en décomposant le comportement désiré en une suite de ces opérations. La complexité du traitement (et de la conception) est ainsi reportée au niveau de l'UC, raison pour laquelle

une UT universelle n'a généralement de sens que couplée à une UC micro-programmable.

Les opération élémentaires indispensables sont l'addition (si l'on désire effectuer des calculs sur des nombres autrement que bit à bit), le ET logique, le OU logique et l'inversion, pour ce qui concerne les traitements combinatoires. Ainsi, le circuit combinatoire effectuant ces diverses opérations est appelé une "unité arithmétique et logique", plus connue sous le sigle ALU, pour Arithmetic Logic Unit.

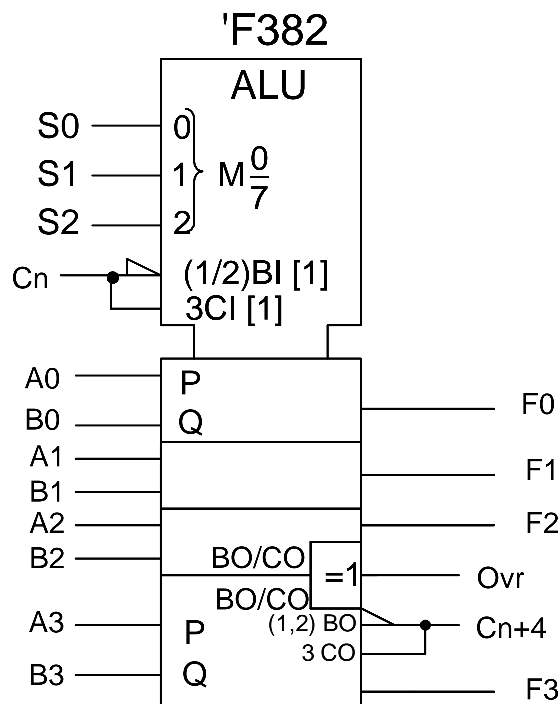


Figure 6- 1 : ALU

Les fonctions autres que celles directement exécutées par l'ALU devront être décomposées en une séquence. Par exemple: additionner N moins une fois le nombre de A à lui-même pour obtenir une multiplication NxA. Il faudra donc conserver des résultats partiels dans des registres, et être en mesure d'acheminer ces résultats de la sortie de l'ALU vers ces registres et des registres vers l'entrée de l'ALU. Une ALU disposant de cette circuiterie dans la même entité est appelée RALU, pour "Registered Arithmetic Logic Unit".

La structure la plus simple que l'on puisse imaginer pour une UT universelle est utilisée dans les calculatrices de poche 4 opérations. Elle n'utilise qu'un seul registre, appelé registre accumulateur, qui mémorise le

résultat d'une opération et sert d'opérande dans l'opération suivante. Le schéma-bloc de cette structure apparaît à la figure 6- 2.

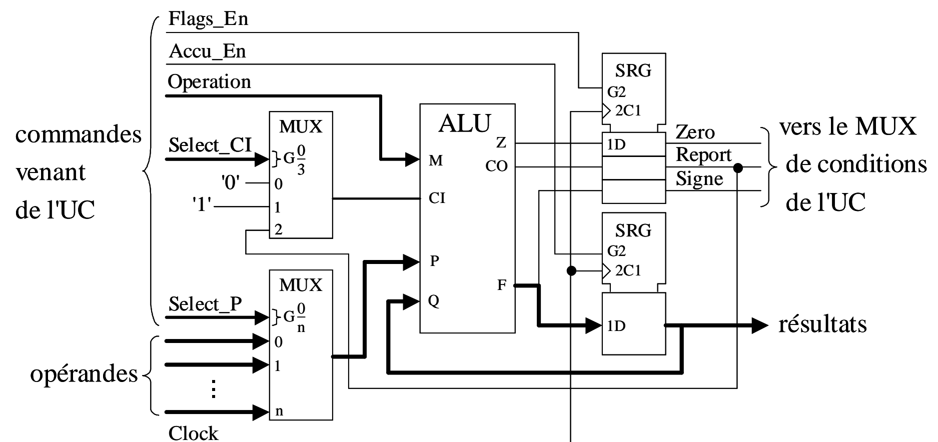


Figure 6- 2 : UT universelle simple

Nous pouvons facilement construire une UT ayant la structure ci-dessus, à l'aide de circuits standard et l'utiliser pour notre vendeur de billets. Cette structure est en effet suffisante, puisque d'après l'organigramme grossier du distributeur, la seule chose que nous ayons à mémoriser est le total versé, et cela peut se faire dans le registre accumulateur.

Prenons par exemple une ALU et ajoutons un MUX à 3 entrées (puisque nous n'avons que 3 opérandes à entrer, soit: la valeur de la pièce, le prix du billet et +1), et un registre accumulateur, le détecteur de zéro qui nous sera nécessaire lors des comparaisons, et les flip-flops de mémorisation du report et de la détection de zéro. Nous obtenons ainsi le schéma de la figure suivante.

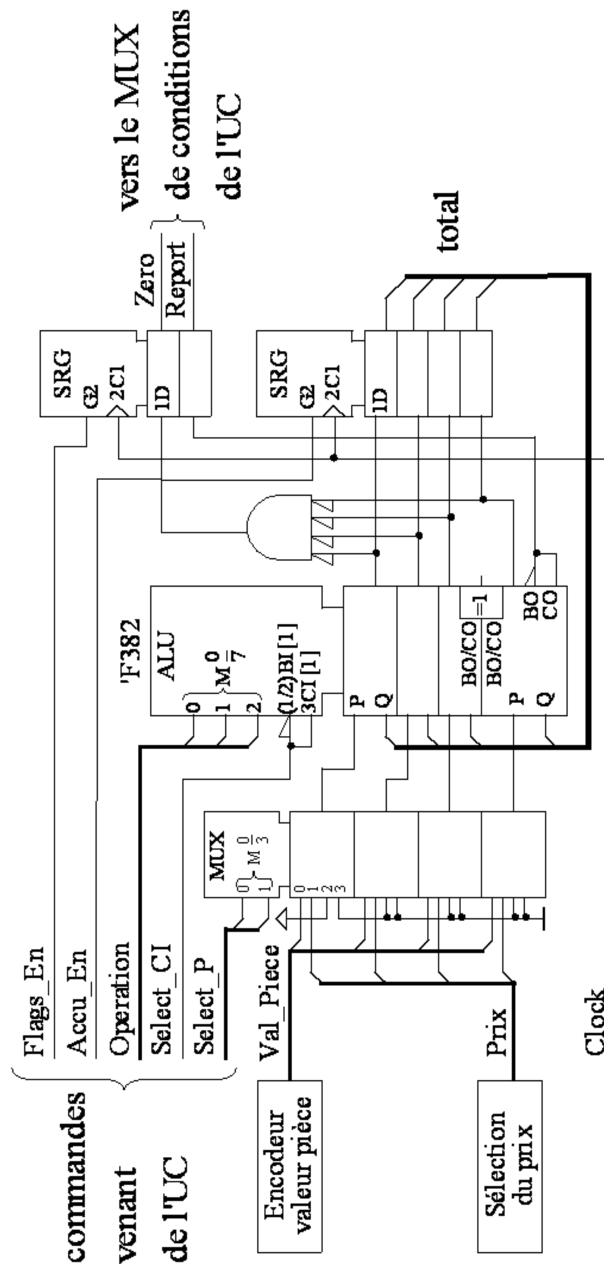


Figure 6- 3 : UT universelle pour le vendeur de billets

Pour comprendre le schéma ci-dessus, il faut tenir compte des remarques suivantes:

1) pour faire une comparaison (total >= prix) à l'aide de l'ALU, il faudra en fait effectuer une soustraction et utiliser les renseignements fournis par la détection de zéro et le report.

2)mais il faudra conserver le total dans l'accumulateur et non pas le remplacer par le résultat de la soustraction: il faut donc une entrée d'enable, que nous avons appelée ACEN, pour permettre ou empêcher la modification de l'accumulateur selon nos besoins.

3) afin de pouvoir tester successivement la détection de zéro et le report de sortie avec un microséquenceur comme MISEQV1, V2 ou V3, il est préférable de les mémoriser lors de la soustraction et de les maintenir ensuite jusqu'à la fin des tests. Il faut donc aussi une entrée d'enable pour commander les flip-flops générant z et CO. Ces signaux sont appelés bit d'état, flags en anglais, d'où le nom de FLEN donné au signal d'enable.

4) le multiplexeur à l'entrée CI de l'ALU apparaissant dans la figure 6-2 n'est pas nécessaire pour notre application, car il nous suffit de préciser s'il doit être à 0 ou à 1. Il en irait autrement si nous devions faire des calculs par tranches successives (par exemple, une addition de 2 nombres de 12 bits réalisée à l'aide de 4 additions de 4 bits) ou des multiplication ou des divisions: dans ce cas il faut pouvoir tenir compte du report précédent.

Avec une UT universelle, il serait fastidieux d'utiliser une UC qui ne peut changer qu'un seul bit de sortie à la fois. La solution la plus économique avec un MISEQV1, 2 ou 3 serait de créer un circuit de sortie dans le genre du BAR8V0 (voir exercices du chapitre précédent) mais permettant de modifier 4 sorties à la fois et comportant au moins 11 sorties au total. Mais, nous allons utiliser la solution la plus couramment employée dans les UC microprogrammées: nous élargirons la mémoire de microprogramme de façon à ce que chaque microinstruction (et pas seulement OUT) permette de commander l'UT. La microinstruction OUT ne sera utilisée que pour générer ENCAISSE, EJECTE et BILLET à travers un registre.

En utilisant MISEQV2, nous obtenons ainsi le schéma de l'UC de la figure 6- 4.

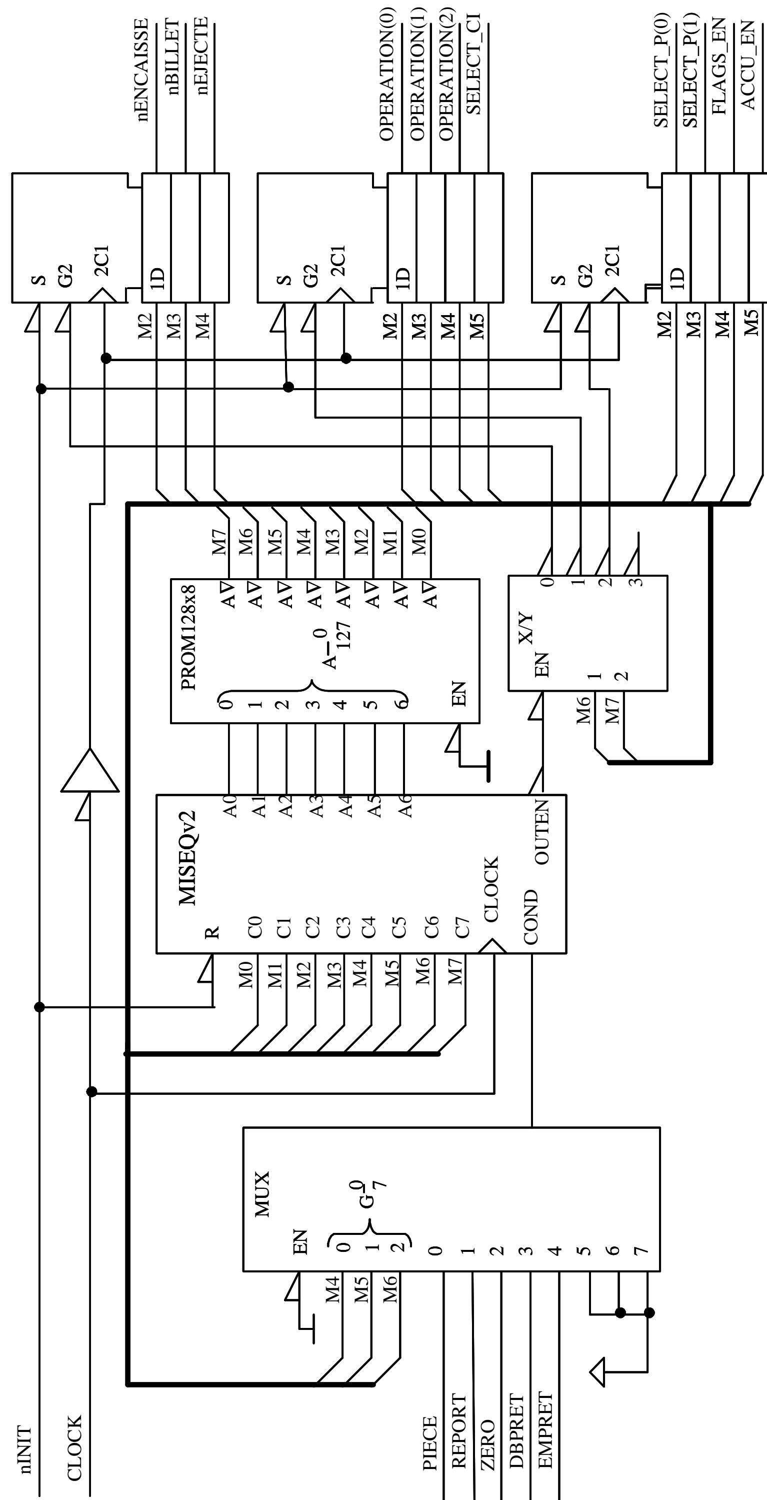


Figure 6- 4 : UC utilisant miseq V2

Pour cette nouvelle UT, l'organigramme détaillé que nous avions établi à la figure 4- 13, n'est plus valable. Il nous faut repartir de l'organigramme grossier de la figure 4- 7 et le raffiner en tenant compte de l'UT choisie. Il en résulte l'organigramme de la figure suivante.

Le microprogramme pour l'UC de la figure 6- 4 apparaît à la figure figure 6- 7. Le format d'une microinstruction est le suivant:

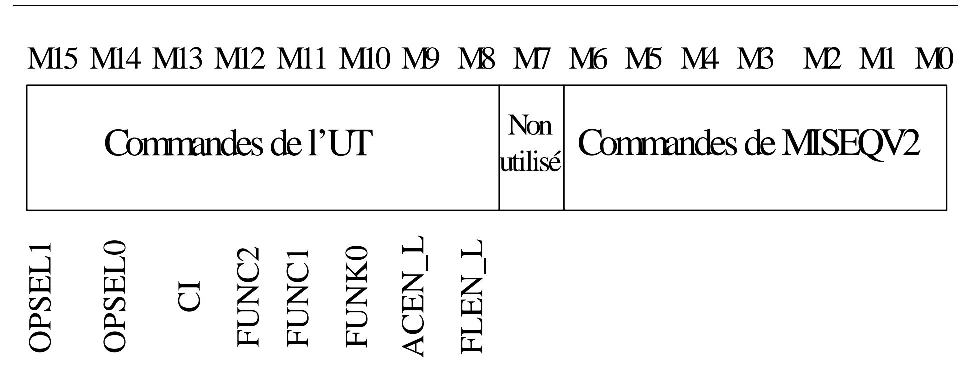


Figure 6- 5 : Format d'une microinstruction

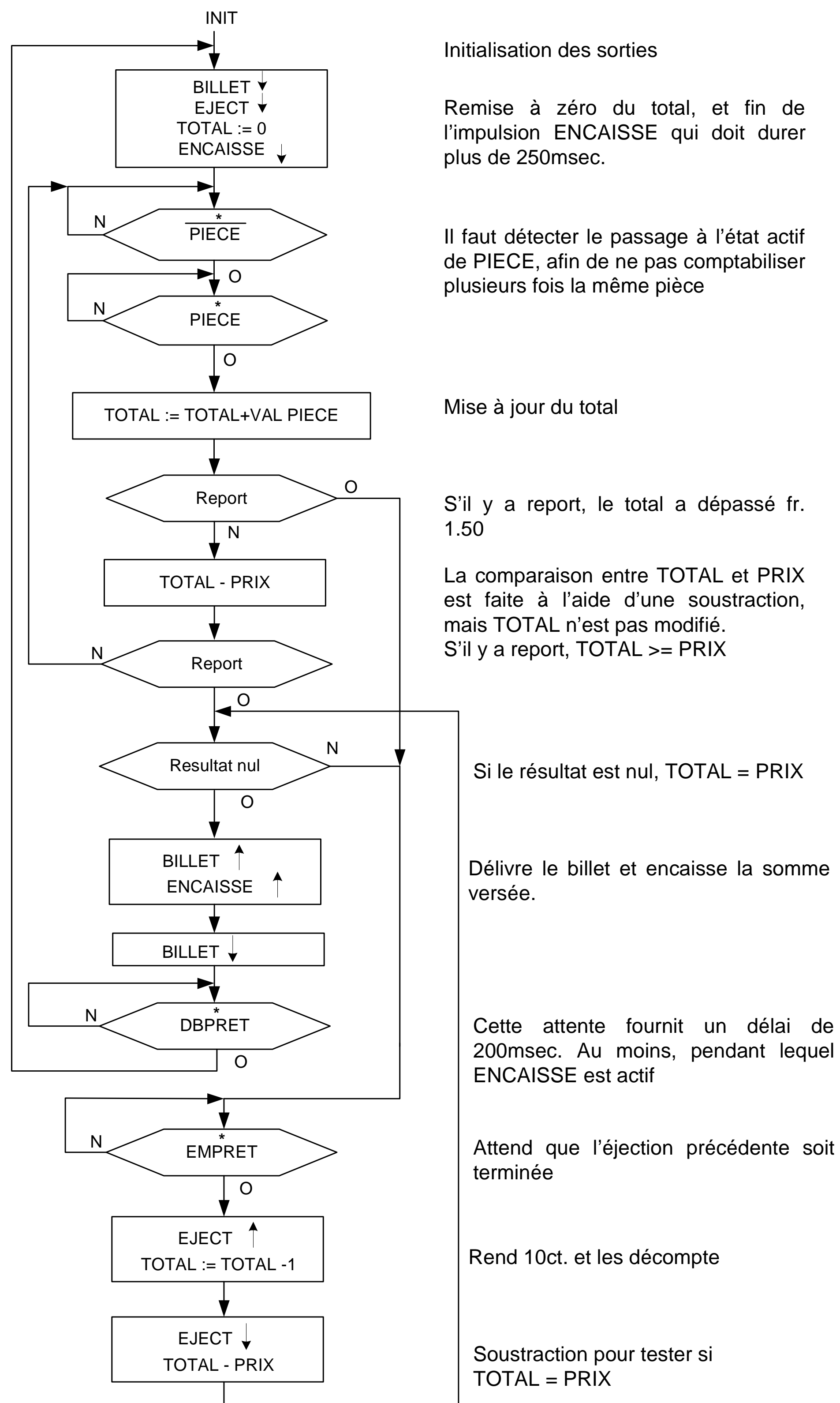
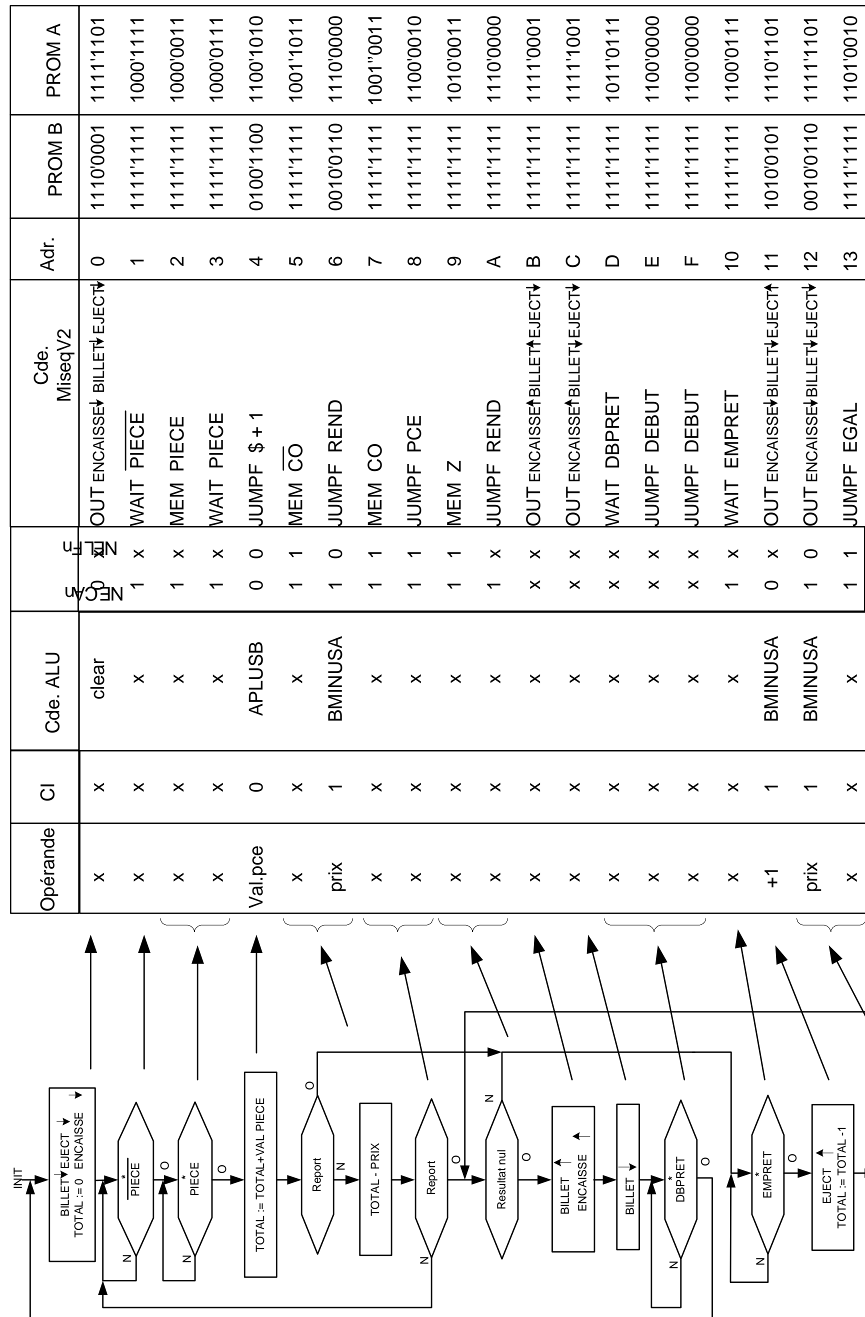


Figure 6- 6 : Organigramme détaillé



Opérande	CI	Cde. ALU	NECAn	NELFn	Cde. MiseqV2	Adr.	PROM B	PROM A
x	x	clear	0	x	OUT ENCAISSE↓ BILLET↓ EJECT↓	0	1110'0001	1111'1101
x	x	x	1	x	WAIT $\overline{\text{PIECE}}$	1	1111'1111	1000'1111
x	x	x	1	x	MEM PIECE	2	1111'1111	1000'0011
x	x	x	1	x	WAIT PIECE	3	1111'1111	1000'0111
Val.pce	0	APLUSB	0	0	JUMPF $ + 1	4	0100'1100	1100'1010
x	x	x	1	1	MEM $\overline{\text{CO}}$	5	1111'1111	1001'1011
prix	1	BMINUSA	1	0	JUMPF REND	6	0010'0110	1110'0000
x	x	x	1	1	MEM CO	7	1111'1111	1001''0011
x	x	x	1	1	JUMPF PCE	8	1111'1111	1100'0010
x	x	x	1	1	MEM Z	9	1111'1111	1010'0011
x	x	x	1	x	JUMPF REND	A	1111'1111	1110'0000
x	x	x	x	x	OUT ENCAISSE↑ BILLET↑ EJECT↓	B	1111'1111	1111'0001
x	x	x	x	x	OUT ENCAISSE↑ BILLET↓ EJECT↓	C	1111'1111	1111'1001
x	x	x	x	x	WAIT DBPRET	D	1111'1111	1011'0111
x	x	x	x	x	JUMPF DEBUT	E	1111'1111	1100'0000
x	x	x	x	x	JUMPF DEBUT	F	1111'1111	1100'0000
x	x	x	1	x	WAIT EMPRET	10	1111'1111	1100'0111
+1	1	BMINUSA	0	x	OUT ENCAISSE↓ BILLET↓ EJECT↑	11	1010'0101	1110'1101
prix	1	BMINUSA	1	0	OUT ENCAISSE↓ BILLET↓ EJECT↓	12	0010'0110	1111'1101
x	x	x	1	1	JUMPF EGAL	13	1111'1111	1101'0010

Figure 6- 7 : Passage de l'organigramme à une suite de micro-instructions.

Pour comprendre la programation de CI dans les soustractions, et l'utilisation de CO pour les comparaisons, il faut savoir que l'ALU effectue l'opération BMINUSA en calculant en fait B plus /A plus CI. En forçant CI à 1, nous obtenons une soustraction en complément à 2.

Par exemple:

B(TOTAL)	= 1Fr.	= 1010
A(PRIX)	= 80ct.	= 1000
$\overline{A}$		= 0111
B		1010
+$\overline{A}$		0111
+1		0001
	report	1 0010

On constate que le report est à 1 lorsque B>=A, et à 0 lorsque B<A.

Au vu de notre exemple d'application, le vendeur de billets, l'UT universelle ne semble guère intéressante. Mais l'UT que nous avons développé même si elle est très simple, pourra être utilisée pour réaliser d'autres systèmes, y-compris des systèmes de complexité sensiblement supérieurs à celle du vendeur de billets, avec des adaptations minimes. Cela constitue un indéniable avantage.

Comme nous le verrons par la suite, les microprocesseurs sont des MSS complexes comportant une UT universelle. C'est sous la forme de microprocesseur que les UT universelles sont surtout utilisées, dans l'état actuel de la techique. Mais les applications à haute performance peuvent nécessiter un développement basé sur une RALU.

Les structures des RALUs standard sont fortement influencées par les besoins propres aux opérations de multiplication et de division. Nous attendrons donc d'avoir quelques notions de calcul arithmétique binaire avant d'en étudier une.

Exercices

1) Avec MISEQV3, est-il possible de passer un paramètre booléen d'un microprogramme à un sous-microprogramme et dans le sens inverse. Si oui, de quelle(s) façon(s)?

2) MISEQV3 ne sauvegarde pas la condition mémorisée lorsque survient une interruption.après un MEM ou pendant un WAIT. Sans se préoccuper du type de PLD qui conviendrait, modifier la description de MISEQV3 de façon à sauvegarder MemCond lors d'une interruptionm, et le récupérer lors d'un IRET (pas lors d'un CALL ou d'un RET). Y a-t-il une solution logicielle?

3) Modifier MISEQV3 de la façon suivante:

-au lieu du CALL, créer 2 microinstructions CALL1 et CALL2.

-au lieu de RET et IRET, créer 2 microinstructions RET1, RET2, correspondant au retour de CALL1 et CALL2 respectivement.

4) Refaire la conception du vendeur de billets en admettant qu'il peut rendre des pièces de 10ct. et de 20 centimes.

5) Créer un circuit de sortie pour l'UC de la figure 6- 4 qui permette de modifier 4 sorties à la fois et comporte 12 sorties au total.

6) Ecrire le microprogramme correspondant à l'organigramme de la figure 6- 6, pour l'UC modifiée selon l'exercice 5.

7) Dans le microprogramme de la figure 6- 7, à quoi sert la microinstruction MEM à l'adresse 2?

Même question pour le JUMPF de l'adresse E? Peut-on remplacer l'opération BMINUSA par une autre, dans la microinstruction de l'adresse 12?

Annexe 1

Bibliographie

Manuel de la HEIG-VD

[ElecNumT1] Electronique numérique TOME1, Messerli/Meyer
[ElecNumT2]
[NumArith]
[IntroVHDL]

Litérature

Ronald J.Tocci -Circuits numériques (théorie et applications) 2ème édition. Dunod

Philippe Larcher -Introduction à la synthèse logique. Eyrolles

Jacques Weber, Maurice Meaudre -Le langage VHDL (cours et exercices) 2ème édition. Dunod

Noël Richard -Electronique numérique et séquentielle (pratique des langages de description de haut niveau). Dunod

John F.Wakerly -Digital design (principles & practice) third edition updated. Prentice Hall

Etienne Messerli -Manuel VHDL EIVD

Philippe Darche -Architecture des ordinateurs. Vuibert

Alexandre Nketsa -Circuits logiques programmables Mémoires, CPLD et FPGA. Ellipse

Médiagraphie

http://www.xilinx.com/

http://jeanlouis.salvat.free.fr/A7/coursWeb/ROM

http://perso.wanadoo.fr/xcotton/electron/coursetdocs.htm

http://artemmis.univ-mrs.fr/iufm-genelec-forum/VHDL/page_html/
1_asic_fpga_cpld_w2000_html.htm

Annexe 2

Lexique

ABEL : langage de programmation des circuits de faible densité d'intégration.
ASIC (Application Specific Integrated Circuit) : circuit non programmable configuré lors de sa fabrication pour une application spécifique.
CPLD (Complex Programmable Logic Device) : circuit intégrant plusieurs PLD sur une même pastille.
EEPROM ou E2PROM (Electrical Erasable Programmable Read Only Memory) : mémoire ROM programmable et effaçable électriquement.
E2PAL (Electrical Erasable PAL) : voir GAL
EPLD (Erasable PLD) : voir GAL.
EPROM (Erasable PROM) : PROM effaçable par UV.
Flash EEPROM : EEPROM utilisant 2 transistors par point mémoire ; utilisé pour les connexions dans les CPLD.
FPGA (Field Programmable Logic Array) : réseau programmable à haute densité d'intégration.
FPLD (Fiel Programmable Logic Device) : terme générique pour les CPLD et FPGA.
FPLS (Fiel Programmable Logic Sequencer) : ancien nom des PAL à registre.
GAL (Generic Array Logic) : PLD programmable et effaçable électriquement
ISP (In Situ Programmable) : caractérise un circuit reprogrammable sur l'application.
JEDEC : organisme de normalisation, donnant son nom aux fichiers de programmation des PLD.

LSI (Large Square Integration) : circuits intégrants quelques centaines à quelques milliers de transistors.
LUT (Lock Up Table) : nom donné aux cellules mémoire réalisant les fonctions combinatoires dans les CPLD et FPGA.
MSI (Medium Square Integration) : circuits intégrants quelques centaines de transistors.
MUX : abréviation pour multiplexeur
PAL (Programmable Array Logic) : ancien nom des PLD.
PLA (Programmable Logic Array) : réseau à matrice ET et OU permettant la réalisation de fonctions combinatoires.
PLD (Programmable Logic Device) : circuit logique programmable intégrant une matrice ET programmable, une matrice OU fixe et plusieurs cellules de sortie.
PROM (Programmable Read Only Mémory) : mémoire ROM programmable.
SPLD (Simple PLD) : par opposition aux FPLD, voir PLD.
SRAM (Static Random Acess Memory) : technologie utilisée pour les connexions dans les CPLD et FPGA.
SSI (Small Square Integration) : circuits intégrant quelques portes.
Verilog : langage de synthèse des circuits numériques
VHDL (Very high speed or scale integrated circuits Hardware Description Language) : langage de modélisation et de synthèse des circuits numérique.