Université de Montréal

Méthodes pour améliorer la qualité des implantations matérielles de systèmes informatiques

par

Noureddine Chabini

Département d'informatique et de recherche opérationnelle

Faculté des arts et des sciences

Thèse présentée à la Faculté des études supérieures
en vue de l'obtention du grade de
Philosophiæ Doctor (Ph.D.)
en informatique

Août, 2001

Université de Montréal

Faculté des études supérieures

Cette thèse intitulée

Méthodes pour améliorer la qualité des implantations matérielles de systèmes informatiques

présentée par

Noureddine Chabini

a été évaluée par un jury composé des personnes suivantes:

...	président du jury
...	examinateur externe
El Mostapha Aboulhamid	directeur de recherche
Yvon Savaria	co-directeur de recherche
...	membre du jury
...	représentant du doyen

Thèse acceptée le: .....

Abstract

In this thesis, we focus on improving the performance and on reducing the area and the power consumption for hardware implementations of synchronous digital systems.

We address a set of problems in a method based on software pipelining techniques that has been recently proposed to derive synchronous circuits that operate at the maximum throughput. We formulate mathematically those problems and propose methods to solve them. We develop methods to determine schedules for reducing register and phase requirements. Also, we devise methods to simultaneously (1) determine schedules that reduce the required number of registers, and (2) place optimally those registers.

To reduce the power consumption of circuits in CMOS technology, we use multiple supply voltages. Decreasing the supply voltage of a computational element implies decreasing its speed and its power consumption. To satisfy performance requirements, highest supply voltages are used for computational elements on critical paths. The other elements off critical paths can be operated at the lowest possible supply voltages. The problem of minimizing the power consumption using multiple supply voltages is NP-hard in general. We formulate mathematically that problem, and propose two methods to solve it. The first method gives the optimal solution and is based on a branch-and-bound technique. The second one is a heuristic formulated as a linear program.

We also devise an efficient method to speed up algorithms for solving the cost-to-time ratio problem. That problem has many practical applications. Algorithms to solve it have been used to determine the maximum throughput of synchronous circuits.

We implement all the methods we propose here in a tool we coded in C++ and test them on well known benchmarks.

Sommaire

Dans cette thèse, nous nous concentrons sur le développement de méthodes pour améliorer la performance ainsi que pour réduire la surface et la consommation de la puissance pour des systèmes informatiques à réaliser en matériel.

Nous traitons divers problèmes avec une méthode basée sur des techniques de pipeline logiciel, récemment proposée pour maximiser la vitesse à laquelle des circuits synchrones peuvent fonctionner. Nous formulons mathématiquement ces problèmes et donnons des méthodes de résolution. Nous développons des méthodes pour déterminer des ordonnancements permettant de réduire le nombre de registres nécessaires ainsi que le nombre requis de phases. Aussi, nous élaborons des méthodes pour déterminer simultanément (1) des ordonnancements qui permettent de minimiser le nombre requis de registres et (2) le placement optimal de ceux-ci dans le circuit.

Pour réduire la consommation de la puissance pour des circuits en technologie CMOS et fonctionnant à une vitesse donnée pouvant être maximale, nous exploitons une approche basée sur l'utilisation de plusieurs tensions d'alimentation. La réduction de la tension d'alimentation d'un élément de calcul entraîne le ralentissement de sa vitesse de traitement et la réduction de sa consommation de puissance. Afin de satisfaire la contrainte de performance, nous assignons les tensions les plus hautes possible aux éléments sur les chemins critiques et les plus basses possible pour les autres éléments. Obtenir la consommation minimale de la puissance en utilisant plusieurs tensions d'alimentation est un problème prouvé NP-Complet en général. Nous formulons mathématiquement ce problème et nous proposons deux méthodes de résolution. La première méthode est exacte et basée sur une technique de type "branch-and-bound". La deuxième méthode est une heuristique formulée sous forme d'un programme linéaire.

Aussi, nous développons une méthode pour accélérer un ensemble d'algorithmes parmi les meilleurs algorithmes connus à date pour le problème "Cost-to-Time Ratio Cycle". Ce problème est important en pratique. Des algorithmes pour sa résolution sont utilisés pour déterminer la vitesse maximale de circuits synchrones.

Nous implantons toutes les méthodes que nous avons développées dans un outil que nous avons codé en C++, et nous testons leur efficacité sur des suites de test largement utilisées dans notre domaine.

Table des matières

Liste des figures

Liste des tableaux

Remerciements

Après trois années d'efforts de recherche aboutissant à cette thèse, c'est le moment de remercier tous ceux qui ont participé d'une façon ou d'une autre à sa réalisation et/ou à l'amélioration de sa qualité.

Qu'il me soit permis de remercier vivement mes directeurs de recherche Messieurs les Professeurs El Mostapha Aboulhamid (Université de Montréal) et Yvon Savaria (École Polytechnique de Montréal) pour l'encadrement de cette thèse, pour leurs encouragements continus, pour leurs conseils judicieux, pour la confiance qu'ils ont accordée à mes travaux et pour l'aide financière qu'ils m'ont octroyée. Qu'ils trouvent ici l'expression de ma profonde gratitude.

Je tiens à exprimer mes profonds remerciements aux membres du jury de cette thèse pour son évaluation.

Je remercie le Département d'informatique et de recherche opérationnelle, la Faculté des études supérieures, l'Université de Montréal, ainsi que le Groupe interuniversitaire en architecture des ordinateurs et VLSI pour les bourses d'excellence qu'ils m'ont accordées.

J'aimerais aussi remercier toutes les personnes du laboratoire d'analyse et de synthèse des systèmes ordinés. Vous avez su créer une ambiance de travail exemplaire.

Je ne crois pas que ce petit paragraphe suffit pour remercier une personne qui m'est très chère: mon frère Ismaïl. Je dois beaucoup à Ismaïl. Depuis mon jeune âge, Ismaïl m'a toujours servi comme un rare et précieux "role model" dans la poursuite de mes études. Ismaïl a aussi contribué, très généreusement, au financement de mes études. Malgré la charge, si lourde, comme Professeur au MIT, Ismaïl a toujours su trouver le temps pour me visiter, m'aider et me conseiller. Ismaïl, merci infiniment.

Finalement, tout mon respect va à ma famille pour ses nombreux sacrifices. Je considère que la réussite de cette thèse est également la sienne.

À la mémoire de mon père

À ma mère

À mon frère Omar

À mes soeurs, à mes frères

Chapitre 1

Introduction

Ce chapitre présente les motivations et les objectifs ainsi que les contributions de cette thèse. Aussi, il présente une revue de la littérature sur des méthodes d'amélioration de performance, de réduction de la consommation de puissance et de réduction de la surface. Il expose le besoin d'algorithmes efficaces pour la synthèse du matériel informatique. Il se termine par l'organisation de la thèse.

1.1- Motivations et objectifs

Un système informatique peut être conçu en matériel, en logiciel ou en matériel et logiciel. Réaliser en matériel un système informatique c'est concevoir un circuit intégré réalisant la fonction désirée du système. Nous nous intéressons dans cette thèse aux circuits numériques synchrones. La qualité d'une telle réalisation se mesure principalement par sa performance, sa surface et sa consommation de puissance. Concevoir un circuit ayant une certaine qualité c'est résoudre un ensemble de problèmes d'optimisation très complexes dont certains sont prouvés NP-complets en général. Par exemple, minimiser le nombre de registres pour un ordonnancement donné des éléments de calcul du système est un problème NP-complet en général [2].

Pour améliorer la qualité d'un circuit, de nombreuses méthodes basées sur une technique, appelée *retiming*, ont été développées. Telle qu'elle est développée au début, la technique de retiming relocalise les registres dans le circuit, tout en préservant sa fonctionnalité, afin d'atteindre l'un des trois objectifs suivants: (1) minimiser la période de l'horloge, (2) minimiser le nombre de registres et (3) minimiser le nombre de registres pour une période d'horloge donnée.

La technique de retiming n'est capable de garantir la détermination de la valeur minimale de la période d'horloge que pour le cas de circuits mono-phase. Dans le cas de circuits multi-phase, des extensions de cette technique ont été faites et elles permettent de déterminer la période

minimale seulement si un choix judicieux des phases est fait.

La technique du pipeline logiciel est une des techniques proposées pour augmenter le parallélisme au niveau des instructions pour des processeurs de type “Very Large Instruction Word” ou superscalaires. Appliquée à un traitement itératif (i.e., une boucle) sans modifier sa sémantique, la technique du pipeline logiciel change l’ordre d’exécution des instructions de la boucle permettant ainsi de lancer l’exécution d’une itération avant que celle qui la précède immédiatement n’ait terminé son exécution.

Une méthode basée sur le pipeline logiciel a récemment été développée pour minimiser la période d’horloge pour des circuits synchrones. Cette méthode est capable de déterminer la valeur minimale de la période d’horloge qu’on n’arrive parfois pas à déterminer en utilisant le retiming. Dans ce cas, le circuit fonctionnant à cette période d’horloge est un circuit multi-phase. Cette méthode comporte plusieurs étapes réalisées dans l’ordre suivant:

- Calcul de la période minimale de l’horloge,
- Détermination d’un ordonnancement valide des éléments de calcul du circuit,
- Placement de registres en utilisant l’ordonnancement déterminé précédemment,
- Assignation de phases.

Dans cette méthode les problèmes suivants s’imposent:

- $\mathbf{P_1}$: Comment déterminer des ordonnancements afin de réduire le nombre de registres.
- $\mathbf{P_2}$: Comment déterminer des ordonnancements afin de réduire le nombre de phases.
- $\mathbf{P_3}$: Comment placer le plus petit nombre de registres, même si un bon ordonnancement est déterminé.

Réduire le nombre de registres permet de réduire la surface du circuit et sa consommation de puissance, alors que réduire le nombre de phases permet de réduire la complexité des tâches de génération et de distribution de l’horloge.

Au cours de la dernière décennie, la consommation de puissance est devenue un des paramètres principaux pour mesurer la qualité d’un circuit. Pour certains cas, elle atteint un degré d’importance égal à la performance. Ceci est dû aux besoins de prolonger la durée de vie des batteries pour les systèmes mobiles et portables, ainsi qu’à celui de réduire l’échauffement pour les

systèmes à haute performance. Une réduction de puissance permet de réduire la taille des batteries ou de prolonger l'autonomie d'un système. D'autre part, la chaleur engendrée par la consommation de puissance, peut endommager le matériel, ralentir sa vitesse de traitement, et même modifier la fonctionnalité du système.

Pour les circuits en technologie CMOS, la puissance dynamique domine la consommation des circuits. La puissance dynamique est une fonction quadratique de la tension d'alimentation. En conséquence, réduire la tension d'alimentation peut réduire de beaucoup la puissance consommée par le circuit.

Pour réduire la consommation de puissance dynamique, des méthodes basées sur l'idée d'utiliser plusieurs tensions d'alimentation ont été récemment proposées dans la littérature [19]. Cependant, ces méthodes ne traitent que le cas où le circuit peut être représenté par un graphe acyclique. La littérature est pauvre pour le cas des graphes cycliques. Notons qu'obtenir la consommation minimale de la puissance, en utilisant plusieurs tensions d'alimentation, est un problème prouvé NP-Complet en général [25].

Dans cette thèse, nous nous concentrons sur le développement de méthodes pour améliorer la qualité des circuits synchrones. Plus précisément, nous nous concentrons sur le développement de méthodes pour réduire la surface et la consommation de puissance pour des circuits opérant à vitesse maximale. Nous traitons les trois problèmes P_1, P_2 et P_3 exposés ci-dessus. Nous traitons le problème de la réduction de la consommation de puissance dynamique en utilisant plusieurs tensions d'alimentation. Aussi, nous examinons l'accélération des meilleurs algorithmes connus à date pour résoudre le problème "Cost-to-Time Ratio Cycle".

1.2- Contributions

Nous élaborons deux formulations mathématiques pour résoudre les problèmes P_1 et P_2 présentés à la Section 1. Nous présentons comment nous pouvons dériver des programmes linéaires à partir de ces formulations. Nous donnons des procédures pour résoudre les programmes linéaires ainsi obtenus. Nous prouvons expérimentalement l'efficacité de notre approche.

Pour résoudre le problème P_3 présenté à la Section 1, nous développons une formulation mathématique qui permet de déterminer simultanément un ordonnancement des éléments de calcul du circuit à optimiser ainsi que le placement des registres, tout en minimisant leur nombre total. À

partir de cette formulation, nous dérivons un programme linéaire mixte ainsi qu'un programme linéaire facile à résoudre. Nous transformons ce programme linéaire en une formulation de flot à coût minimum, puisque ce dernier problème possède des algorithmes de résolution très efficaces. Nous démontrons expérimentalement l'efficacité de notre approche.

Pour réduire la consommation de la puissance pour des circuits en technologie CMOS, nous exploitons une approche basée sur l'utilisation de plusieurs tensions d'alimentation. Avec notre méthode, pour une période d'horloge donnée qui peut être minimale, nous affectons les tensions d'alimentation les plus hautes aux éléments de calcul qui sont sur les chemins critiques et des tensions les plus basses possibles pour les autres éléments. Nous formulons mathématiquement notre approche et nous proposons deux méthodes de résolution. La première méthode est basée sur une technique de type "branch-and-bound" et permet de déterminer la solution optimale. La deuxième méthode est une heuristique formulée sous forme d'un programme linéaire. Nous testons l'efficacité des deux méthodes sur un ensemble de cas de test.

Aussi, nous développons une méthode pour déterminer une borne sur la solution optimale du problème "Cost-to-Time Ratio". Cette méthode se caractérise par une faible complexité temporelle. Nous utilisons les bornes obtenues par cette méthode pour démontrer expérimentalement l'accélération d'un ensemble d'algorithmes de résolution de ce problème.

Nous testons l'efficacité des méthodes que nous développons en les implémentant dans un outil que nous avons codé en C++.

1.3- Amélioration de la performance

La synthèse de haut-niveau est le processus qui transforme une description algorithmique, d'un système à réaliser en matériel, en une description dite transfert de registres. Cette transformation ressemble à la compilation de logiciels. Afin d'améliorer la performance du système en cours de design, des techniques empruntées de la compilation de logiciels ont été largement utilisées. Parmi ces techniques, nous citons la réduction de la hauteur de l'arbre pour l'évaluation des expressions arithmétiques, la propagation de constantes et de variables, l'élimination du code mort, la transformation d'opérateurs en remplaçant par exemple une multiplication par des décalages et des additions ou soustractions, la conversion d'instructions conditionnelles comme par exemple if-then-else, le déroulage de boucles, etc. Pour des exemples

illustrateurs pour chacune de ces techniques, nous référons le lecteur intéressé à [35] pages 126-136. D'autres techniques spécifiques au matériel ont aussi été proposées. Par exemple, le chaînage et le pipelinage des opérateurs nécessitant plusieurs cycle d'horloges.

Une façon d'améliorer la performance d'un système est d'améliorer la performance de ses éléments de calcul. Ces éléments peuvent être des circuits combinatoires ou séquentiels. L'optimisation de circuits combinatoires est généralement appelée resynthèse. De nombreuses techniques pour l'amélioration de la performance durant la resynthèse ont été proposées. Citons, par exemple, la réduction de la longueur des chemins critiques basée sur l'associativité et la distributivité. D'autres techniques sont présentées dans [35].

Pour des systèmes séquentiels synchrones, l'amélioration de la performance de leurs circuits combinatoires peut ne pas améliorer la performance de ces systèmes. En effet, la vitesse de ce type de systèmes est limitée par la rapidité du signal de synchronisation, appelé horloge. Ce signal est en général périodique et sa période est appelée période de l'horloge. La valeur minimale de cette période est déterminée par la valeur maximale du délai parmi l'ensemble des chemins, formés seulement par des blocs combinatoires, entre chaque paire de registres adjacents. En conséquence, en déplaçant certains registres sans modifier la fonctionnalité du système, on peut réduire cette valeur maximale et donc réduire la période de l'horloge. En effet, la technique appelée *retiming* a été proposée par Leiserson et *al.* [71] pour atteindre cet objectif.

Il semble évident qu'en combinant le retiming et des techniques de resynthèse, on peut réduire la valeur de la période de l'horloge des systèmes séquentiels synchrones. Une technique combinant le retiming et la resynthèse est proposée dans [83]. Cette technique peut être décrite comme suit. Premièrement, si le circuit à optimiser est cyclique, on le rend acyclique en supprimant certaines connexions sur lesquelles il y a des registres. Ensuite, on applique le retiming pour dégager les registres vers les entrées et/ou les sorties. Durant cette étape, certaines connexions peuvent porter un nombre négatif de registres, mais le concept d'emprunter temporairement des registres de l'environnement est utilisé. Par la suite, le circuit combinatoire résultat est soumis à des technique de resynthèse. Finalement, on applique le retiming pour remettre les registres dans le circuit. D'autres travaux combinant le retiming et la resynthèse se trouvent dans [7, 35, 36, 89].

La technique de retiming est originellement proposée pour optimiser des systèmes séquentiels synchrones où les registres sont de type flip-flop. Ce type de registres est contrôlé par

le front montant ou descendant de l'horloge. Cette technique est capable de produire la valeur optimale de la période d'horloge seulement pour le cas de systèmes mono-phase. Des extensions de cette technique ont été faites pour traiter le cas de systèmes à plusieurs-phases et de systèmes où les registres sont des bascules qui sont contrôlées par le niveau haut ou bas de la période de l'horloge. Une revue de ces techniques est présentée dans le Chapitre 4.

Un nombre négatif de registres sur une connexion n'a pas de signification physique. En conséquence, une des conditions de la validité de la technique de retiming est qu'après son application sur un circuit, le circuit résultat ne doit contenir aucun nombre négatif de registres. Cependant, pour réduire davantage la valeur de la période de l'horloge, une méthode basée sur le retiming et qui permet d'avoir une valeur négative de registres sur certaines connexions est proposée dans [49, 50, 51]. L'idée de base est d'introduire des registres sur les chemins critiques. Mais, pour conserver la fonctionnalité du circuit, chaque registre introduit est suivi d'un registre dit négatif (i.e., on ajoute et on retranche la même quantité de registres). Chaque registre négatif est implémenté en utilisant des techniques de pré-calcul et de prédiction. Si le pré-calcul n'est pas possible, on utilise la prédiction. Si la prédiction échoue, des mécanismes pour corriger la situation ont été proposés.

Les "Telescopic Circuits" [4] ont été proposés pour améliorer la performance. L'idée est basée sur le fait que le délai d'un circuit dépend des valeurs actuelles et passées de ses entrées. Par exemple, pour un additionneur de type "Ripple Carry" à 32-bits, si ses entrées sont passées de {0...0, 0...0} à {0...0, 0...01} alors le délai de l'additionneur est égal au délai d'un additionneur complet à 1-bit. En conséquence, ce type de circuits possède un délai variable et ils sont appelés unités à délai variable [93]. Une méthodologie pour intégrer les unités à délai variable au niveau de la synthèse de haut-niveau a été proposée dans [93].

Pour des systèmes séquentiels synchrones, le décalage de l'horloge (i.e., "clock skew") est dû à l'inégalité des temps de l'arrivée du signal de synchronisation (horloge) aux entrées des registres. Le phénomène de décalage de l'horloge doit normalement être évité pour que le système fonctionne correctement. Cependant, ce phénomène a été exploité comme un moyen pour améliorer la performance [38]. Nous traitons du décalage de l'horloge dans le Chapitre 4.

Pour ce type de systèmes, la distance en unité de temps entre chaque paire de registres voisins ne doit normalement pas dépasser la valeur de la période de l'horloge. Cependant, en

utilisant le concept de “wave pipelining”, cette distance peut dépasser la valeur de la période de l’horloge [13, 56, 57, 113]. L’idée de base est que, pour ne pas changer la fonctionnalité du système, plusieurs ondes traversent en même temps le bloc combinatoire situé entre les deux registres qui sont séparés par une distance plus grande que la période de l’horloge. Pour avoir le bon fonctionnement du système, ces ondes ne doivent pas se chevaucher. Le “wave pipelining” est donc une autre façon pour réduire la période de l’horloge. Un tutoriel et une revue de la littérature sur le “wave pipelining” se trouvent dans [13].

1.4- Réduction de la consommation de la puissance

En technologie CMOS, la puissance dynamique est la composante la plus importante dans l’équation de la consommation moyenne de la puissance. La dissipation de la puissance dynamique est due au chargement et au déchargement des capacités dans le circuit. Pour une porte logique, la puissance dynamique consommée est défini comme suit:

$$P_{dyn} = K \cdot C \cdot f \cdot V^2, \quad (1)$$

où K est le facteur de commutation, C est la capacité de charge, f est la fréquence du fonctionnement de la porte et V est sa tension d’alimentation.

La littérature nous indique que pour produire des design qui consomment le moins possible de puissance, la dissipation doit être réduite durant tous les niveaux de hiérarchie du design [75]. En plus, c’est aux plus haut-niveaux d’abstraction du design que le pourcentage de réduction de la dissipation de la puissance est élevé.

Les techniques empruntées du design de compilateurs de logiciels que nous avons abordées dans la Section 3 sont aussi proposées comme techniques pour réduire la dissipation de la puissance. Des discussions détaillées de ces techniques, ainsi que d’autres techniques au niveau de la synthèse de haut-niveau, ont été présentées dans [26, 75].

À partir de l’équation (1), nous pouvons constater que la dissipation de la puissance dynamique peut être réduite en réduisant certains de ses paramètres K, C, f et V. À cause de la présence du terme quadratique en V, la dissipation de la puissance peut être réduite de beaucoup si on réduit V. Cependant, la réduction de la tension d’alimentation d’un élément de calcul entraîne le ralentissement de sa vitesse de traitement. Afin de réduire la consommation de la puissance tout en répondant aux contraintes de performance, une approche basée sur la réduction de V a été

proposée dans [26]. Dans cette approche, les contraintes de performance sont satisfaites par l'augmentation de la concurrence dans le système en cours de design. Pour cela, plus de matériel est utilisé pour exécuter plusieurs tâches en parallèle. Un des inconvénients de cette approche est l'augmentation de la surface du circuit. Notons que l'augmentation de la surface peut se traduire en une augmentation des capacités parasites dans le circuit et donc augmenter la consommation de puissance. Une autre façon de réduire la consommation est l'utilisation de plusieurs tensions d'alimentation. Pour cela, on alimente les éléments de calcul sur les chemins critiques avec les tensions les plus élevées. Les autres éléments sont alimentés par les tensions les plus faibles possibles. Nous abordons des approches utilisant plusieurs tensions d'alimentation dans le Chapitre 5.

La réduction de la consommation de puissance en jouant sur le terme ($K.C$) a été traitée dans la littérature. En effet, le concept de "shut down" a été introduit. L'idée est que si une partie du système n'est pas active, on la débranche temporairement de la source de tension jusqu'à ce qu'elle devienne active. Des techniques d'allocation de mémoire et de codage du bus ont aussi été proposées dans la littérature pour réduire ($K.C$) [75].

Le pré-calcul est proposé comme une technique pour réduire la consommation de puissance. L'idée est d'identifier des conditions logiques pour certaines entrées d'un bloc combinatoire qui, dans certains cas, n'ont pas d'influence sur la sortie de ce bloc. Étant donné que la sortie sera invariante pour ces entrées, ces derniers peuvent, dans ce cas, être débranchés temporairement du bloc combinatoire. De cette façon la valeur de ($K.C$) peut être réduite. Un exemple illustrant cette technique se trouve dans [75].

La technique de retiming a aussi été utilisée pour réduire la dissipation de puissance [87]. L'idée de base est que les registres peuvent être déplacés de façon à réduire la valeur totale du terme ($K.C$). Cependant, le nombre de registres peut augmenter et donc la consommation de puissance peut augmenter.

1.5- Réduction de la surface

En plus du coût du matériel, la surface a des impacts sur la portabilité et la consommation de la puissance du circuit. Afin de réduire la surface, des techniques ont été proposées à travers les différents niveaux de hiérarchie des design. Des techniques empruntées du design des compilateurs

de logiciels ont été proposées pour réduire la surface. Parmi ces techniques, citons l'élimination des expressions communes et l'élimination du code mort.

La technique de retiming [71] permet de minimiser la période de l'horloge de systèmes séquentiels synchrones. Cependant, le nombre de registres peut diminuer ou augmenter. Pour réduire le nombre de registres, des algorithmes pour le retiming ont été développés par Leiserson et *al.* [71].

Des extensions de la technique de retiming ont été proposées pour réduire, d'une façon indirecte, la surface. Dans [54], la conversion de circuits formés de registres de type flip-flop en des autres constitués de registres de type *latche* a été examinée. Dans [72], les auteurs rapportent que le coût d'un *latche* est égale à la moitié de celui d'un flip-flop. Bien que la technique du "Peripheral Retiming" [83] permet d'améliorer la performance, elle permet aussi de réduire la surface grâce aux techniques de la resynthèse appliquées [95].

L'optimisation de la machine à états finis est un sujet largement étudié dans la littérature. L'objectif d'une telle optimisation est de réduire la surface du circuit implémentant cette machine. Le lecteur intéressé par des techniques d'optimisation des machines à états finis peut consulter [35].

1.6- Amélioration de l'efficacité d'algorithmes pour la conception du matériel

Pour exposer le besoin d'algorithmes efficaces durant le design en matériel de systèmes de grandes tailles, nous présentons les efforts faits pour rendre les algorithmes de retiming efficaces en pratique, même s'ils ont une complexité temporelle polynômiale. Dans la suite, pour éviter les répétitions, nous dénotons par *MinPeriod* et par *MinArea* les algorithmes de retiming pour réduire la période d'horloge et pour réduire la surface, respectivement. Dans [8], les résultats expérimentaux montrent que le temps d'exécution de *MinPeriod* est de 3.8 jours pour le cas d'un circuit de 12529 éléments de calcul et 24764 connexions. La littérature nous indique que le temps d'exécution de *MinArea* est beaucoup plus grand que celui de *MinPeriod*.

Dans [101], une implantation efficace de *MinPeriod* et *MinArea* est faite. Dans cette implantation, la taille mémoire requise est réduite de n^2 à n, où n est le nombre de noeuds du graphe sur lequel s'exécutent ces algorithmes. Aussi, dans cette implantation, des pointeurs de retour ont été utilisés pour accélérer l'étape de vérification de la faisabilité durant l'étape de la

recherche binaire de la valeur minimale de la période d'horloge. Dans [112], une technique pour réduire le nombre de contraintes du programme linéaire pour *MinArea* est présentée. Dans [99], les auteurs exploitent l'équivalence entre le retiming et le "clock skew" [37, 38] pour accélérer *MinPeriod* et montrent expérimentalement que cela a permis d'exécuter *MinPeriod* en moins de 2 min pour des circuits où $n \leq 20000$. Dans [77, 78, 81], cette équivalence est aussi utilisée pour dériver des bornes sur les variables du programme linéaire pour *MinArea*, et les auteurs rapportent que ces bornes ont permis d'exécuter *MinArea* en moins de 15 min pour des circuits où $n \leq 50000$.

1.7- Organisation de la thèse

Cette thèse suit un format par article. À l'exception de l'introduction et de la conclusion, le corps de chacun de ses autres chapitres est formé par un article. Chacun de ces chapitres débute par un résumé en français. Elle est organisée comme suit. Le Chapitre 2 se concentre sur la résolution des deux problèmes P_1 et P_2 présentés à la Section 1.1. Son corps se constitue de l'article [18]. L'objectif du Chapitre 3 est la transformation des formulations proposées dans le Chapitre 2, pour la résolution des problème P_1 et P_2, en des formulations d'un problème de flot à coût minimum qui peut être résolu efficacement. Le corps de ce chapitre est formé de l'article [20]. Le Chapitre 4 vise la résolution du problème P_3 présenté à la Section 1.1, et son corps comprend l'article [22]. Le Chapitre 5 se concentre sur la réduction de la consommation de la puissance par l'utilisation de plusieurs tensions d'alimentation. Son corps se compose de l'article [19]. Le Chapitre 6 traite de l'accélération d'algorithmes de résolution du problème "Cost-to-Time Ratio Cycle", et son corps est composé de l'article [17]. Le Chapitre 7 présente les contributions de cette thèse et propose des avenues de recherches futures.

Chapitre 2

Réduction du nombre de registres et du nombre de phases pour des circuits synchrones à débit maximal

Le débit d'un circuit synchrone mono-phase est l'inverse de la période de l'horloge du circuit. En réduisant la période de l'horloge, le débit du circuit augmente. Cependant, il ne peut dépasser une valeur T_{crit} due à la contrainte de dépendance de données. Comme nous l'avons présenté dans le Chapitre 1, divers méthodes ont été proposées pour augmenter le débit. Une de ces méthodes qui est capable d'avoir toujours un circuit ayant un débit égal à T_{crit} a été récemment proposée. Cette méthode est basée sur le pipeline logiciel qui est une technique connue pour augmenter le parallélisme au niveau des instructions pour des processeurs parallèles. Cette méthode comporte plusieurs étapes réalisées dans l'ordre suivant:

- Calcul de la période minimale de l'horloge,
- Détermination d'un ordonnancement valide des éléments de calcul du circuit,
- Placement de registres en utilisant l'ordonnancement déterminé précédemment,
- Assignation de phases.

Dans cette méthode les problèmes suivants s'imposent:

- **P_1**: Comment déterminer des ordonnancements afin de réduire le nombre de registres.
- **P_2**: Comment déterminer des ordonnancements afin de réduire le nombre de phases.

Réduire le nombre de registres permet de réduire la surface du circuit et sa consommation de puissance, alors que réduire le nombre de phases permet de réduire la complexité des tâches de génération et de distribution de l'horloge.

Dans ce chapitre, nous nous concentrons sur la résolution des problèmes P_1 et P_2. Nous développons deux formulations mathématiques pour déterminer des ordonnancements permettant de réduire le nombre de registres et le nombre de phases. Nous donnons des procédures pour les résoudre. Nous démontrons expérimentalement l'efficacité de notre approche sur un ensemble de cas de test largement utilisés dans notre domaine.

Nous poursuivons ce chapitre par la présentation de l'article [18].

Reducing Register and Phase Requirements for Synchronous Circuits Derived Using Software Pipelining Techniques

Noureddine Chabini[1], El Mostapha Aboulhamid[1], Yvon Savaria[2]

1: *LASSO, DIRO, Université de Montréal C.P.6128, Suc. Centre ville, Montréal, Qc, Canada, H3C 3J7. Email:{chabinin, aboulham}@iro.umontreal.ca*

2*: GRM, DGEGI, École Polytechnique de Montréal, C.P. 6079, Suc. Centre-ville, Montréal, Qc, Canada, H3C 3A7. Email: savaria@vlsi.polymtl.ca*

Abstract

A method based on a modulo scheduling algorithm for software pipelining has been recently proposed to optimize clocked circuits. The resulting circuits are multi-phase clocked circuits, where all clocks have the same period. To preserve the functionality of the original circuit, registers must be placed after minimizing the clock period. The placement of these registers is derived from an arbitrary schedule determined during a clock period minimization step. A good schedule may allow to decrease the number of registers and the number of phases needed in the final circuit. Decreasing the number of registers contributes to minimizing the area occupied by the circuit and reduces its power consumption; while decreasing the number of phases reduces the complexity of the clock generation and distribution tasks. In this paper, we propose polynomial-time-solvable methods to choose a good schedule once the clock period is minimized. The methods have been tested on a subset of the ISCAS89 benchmarks. Experimental results show that the number of registers which must be inserted in the final circuit, and the number of phases, have been significantly decreased compared to the case where an arbitrary schedule is chosen.

2.1- Introduction

The performance of clocked digital circuits can be improved by minimizing their clock period. Retiming [71] is a technique often used for that purpose. This technique changes register placement in the circuit to minimize the maximum combinational delay between any two neighboring registers. Basic retiming supposes that the circuit is clocked by a single-phase clock,

which limits the search space for effective timing solutions. Various extensions to retiming have been proposed [54, 72]. In [72], retiming with multi-phase clocks was proposed. With this method, the phases are fixed before retiming, which can give an optimal clock period if good phases are chosen.

Software pipelining has been proved to be a powerful technique for increasing the instruction-level parallelism for parallel processors. It has recently been used for optimizing clocked circuits [8]. The resulting circuit is a multi-phase clocked circuit, where all clocks have the same period. That method may be described as follows. First, the optimal clock period is determined, and a schedule of all the functional elements of the circuit is computed. Second, in order to preserve the behavior of the original circuit, registers are placed independently of their initial placement. The placement of registers is done using the fixed schedule. Finally, once the registers are placed, the phases are determined.

In the previous method, As Soon As Possible (ASAP) or As Late As Possible (ALAP) schedule is used. Choosing a good schedule may allow to decrease the number of registers that must be placed in the final circuit, and its number of phases. Decreasing the number of registers contributes to minimizing the area occupied by the circuit and reduces its power consumption; while decreasing the number of phases reduces the complexity of the clock generation task. In this paper, we propose polynomial-time-solvable methods to determine schedules for reducing the number of registers and the number of phases. The methods have been tested on various benchmarks selected from the ISCAS89 set. Experimental results show that the number of registers that must be inserted in the final circuit, and the required number of phases, can be substantially reduced compared to the case where an ASAP or an ALAP schedule is chosen.

The remainder of this paper is organized as follows. In Section 2, we introduce the notations and definitions used in this work. Section 3 briefly reviews the registers placement step in the method based on software pipelining, which was outlined above. Our methods to determine schedules for reducing the number of registers and the number of phases are presented in Sections 4 and 5. Section 6 presents experimental results. Conclusions are provided in Section 7.

2.2- Preliminaries

2.2.1- The cyclic graph model

In order to minimize the clock period of a synchronous sequential circuit, it is modeled (as in [8, 71]) as a directed cyclic graph $G = (V, E, d, w)$, where V is the set of functional elements in the circuit, and E is the set of edges which represent interconnections between vertices. Each vertex v in V has a non-negative integer propagation delay $d(v) \in N$. Each edge $e_{u, v}$, from u to v, in E is weighted with a register count $w(e_{u, v}) \in N$, representing the number of registers on the wire between u and v.

Figure 2.1 presents an example of a circuit and its directed cyclic graph model. In this figure, large rectangles represent functional elements, and small rectangles represent registers. Wires are oriented to show the propagation direction of the signals. The propagation delay of each functional element of this circuit is specified as a label on the left of each large rectangle. This example will be used through this paper, and will serve as an example of initial specification for the problem to optimize. The initial specification is in general a synchronous circuit with a single-phase clock period. The clock period of the circuit in Figure 2.1 is 6, which is equal to $d(v_4) + d(v_1)$.

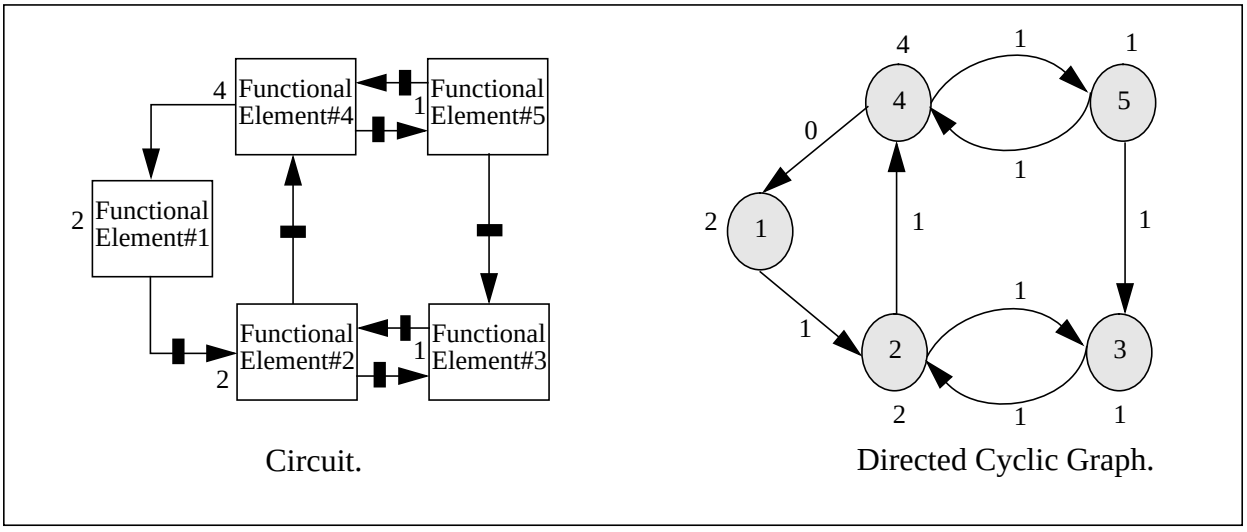


Figure 2.1 : Sample circuit and its directed cyclic graph model.

2.2.2- Periodic and *k*-periodic schedules

We define a schedule *s* [8, 3] as a function $s : N \times V \rightarrow Q$, where $s_n(v) \equiv s(n, v)$ denotes the schedule time of the n^{th} iteration of operation *v*. In multi-phase flip-flop based circuits, the schedule time is the start time of the operation. A schedule *s* is called periodic with period *P*, if:

$$\forall n \in N,\ \forall v \in V\text{: } s_{n+1}(v) = s_n(v) + P\,. \tag{1}$$

A schedule *s* is called *k*-periodic if there exist integers n_0 and *k*, and a positive rational number *P* such that:

$$\forall n \geq n_0,\ \forall v \in V : s_{n+k}(v) = s_n(v) + k \cdot P\,. \tag{2}$$

Both periodic and *k*-periodic schedules have the same throughput $T = 1/P$. The *k*-periodic schedules have a period of $(k \cdot P)$. When there is no resource constraint, a schedule *s* is said to be valid if and only if the operations terminate before their results are needed. In this case, we say that data dependencies are satisfied, which is equivalent to the following mathematical inequality:

$$\forall n \in N,\ \forall e_{u,v} \in E : s_{n+w(e_{u,v})}(v) \geq s_n(u) + d(u)\,. \tag{3}$$

2.2.3- Maximum throughput of synchronous sequential circuit

The throughput, *T*, of synchronous sequential circuit is bounded by the inverse of the length, *P*, of the critical paths in the circuit. Based on data dependencies constraints only, the maximum throughput is [3]:

$$T = Min_{c \in C}\left(\left(\sum_{e_{u,v} \in c} w(e_{u,v})\right) \Big/ \left(\sum_{\forall v \in V \text{ and } e_{u,v} \in c} d(u)\right)\right), \tag{4}$$

where *C* is the set of directed cycles in the directed cyclic graph modeling the circuit. Determining the maximum throughput is a Minimal Cost-to-Time Ratio Cycle Problem [43, 66], which can be solved in the general case in $O(|V| \cdot |E| \cdot \log(|V| \cdot d_{max}))$ [66], where $d_{max} = Max_{v \in V}(d(v))$. A possible method to solve this problem is to iteratively apply Bellman-Ford's algorithm for longest paths on the graph $G_P = (V, E, d, w_P)$ derived from *G* by letting:

$$w_P(e_{u,v}) = d(u) - P \cdot w(e_{u,v}), \tag{5}$$

where $e_{u,v} \in E$ and $P = 1/T$. A binary search may be used to find the minimal value of *P* for which there is no positive cycle in G_P [3].

For the example in Figure 2.1, we have that $P = 4$. This value corresponds to the cycle defined by vertices v_1, v_2, and v_4.

2.2.4- Schedule for a given throughput

From equation (1) and inequality (3), we have that:

$$\forall e_{u,v} \in E,\ s_0(v) - s_0(u) \geq d(u) - P \cdot w(e_{u,v}). \quad (6)$$

In the case of periodic schedules, determining a valid schedule of all the instances of each vertex v in V is equivalent to determining $s_0(v)$ for each v in V, which is also equivalent to determining solutions to the system of inequalities described by (6). To resolve this system, the graph G_P, previously described, may be used. To find an ASAP schedule, Bellman-Ford's algorithm for longest paths, from a chosen vertex v_x to the others, may be applied on the graph G_P. Finding an ALAP schedule may be done as follows. Step 1, a graph G' has to be derived from G_P by inverting the direction of each edge in G_P. Step 2, Bellman-Ford's algorithm for longest paths, from the vertex v_x to the others, has to be applied on the graph G', where the weights of its edges are defined by equation (5). Finally, step 3, the ALAP schedule is obtained by multiplying each result in step 2 by -1. Table 2.1 gives the ASAP and ALAP schedules, relatively to $v_x = v_1$, of vertices of the circuit in Figure 2.1.

Table 2.1 : ASAP and ALAP schedules.

	Vertices				
	v_1	v_2	v_3	v_4	v_5
ASAP	0	-2	-4	-4	-4
ALAP	0	-2	1	-4	-1

2.2.5- Schedule graph

As in [8], a periodic schedule, with period P, of vertices of directed cyclic graph modeling a circuit is presented by a schedule graph $G_s = (V, E, d, w_s, P)$. Here V, E and d have the same definition given for the case of the graph G previously defined, and $w_s : E \rightarrow Q$ is a weight function, which associates to each edge $e_{u,v}$ in E the time distance between the schedule times of u and v. Mathematically, $w_s(e_{u,v})$ is defined as follows:

$$\forall e_{u,v} \in E,\ w_s(e_{u,v}) = s_{w(e_{u,v})}(v) - s_0(u). \quad (7)$$

Because s is periodic with period P, equation (7) may be written as follows:

$$\forall e_{u,v} \in E,\ w_s(e_{u,v}) = s_0(v) - s_0(u) + P \cdot w(e_{u,v}). \quad (8)$$

The graph G_s is consistent if and only if for each edge $e_{u,v}$ in E, $w_s(e_{u,v}) \geq d(u)$. This is derived from equation (3). Figure 2.2 shows a consistent schedule graph, where edges are labeled with w_s values, for the circuit in Figure 2.1 using the ALAP schedule from Table 2.1.

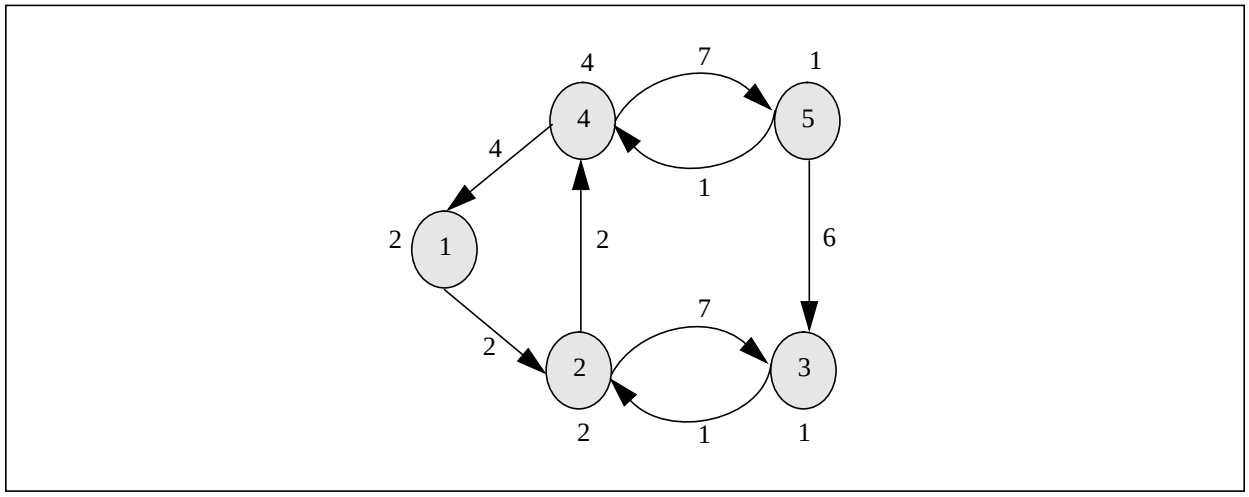


Figure 2.2 : Schedule graph.

2.2.6- Incidence, unimodular and eulerian matrices

In this paper, the incidence matrix, *M*, of a directed graph, *G*, is a matrix whose lines are indexed by the edges of *G* while the columns are indexed by its vertices. The entries of *M* are defined as follows: $M_{e,i}$ is equal to 1 if *i* is the tail of the edge *e*, to -1 if *i* is the head of *e*, and to 0 otherwise. For instance, the incidence matrix of the graph in Figure 2.1 is:

$$M = \begin{bmatrix} 1 & -1 & 0 & 0 & 0 \\ 0 & 1 & -1 & 0 & 0 \\ 0 & 1 & 0 & -1 & 0 \\ 0 & -1 & 1 & 0 & 0 \\ -1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & -1 \\ 0 & 0 & -1 & 0 & 1 \\ 0 & 0 & 0 & -1 & 1 \end{bmatrix}.$$

A matrix *M* is *totally unimodular* if every one of its square submatrices has a determinant equal to

-1 , 0 or 1 [15]. A submatrix of a matrix is said to be *Eulerian* [15, 6] if the sum of the entries of its lines and the sum of the entries of its columns are both even. This will be used to prove Theorem 2.1 in Section 2.4.8.

2.3- Register placement

In the method proposed in [8], which was outlined in Section 2.1, a register placement step is needed in order to preserve the behavior of the original circuit. The placement of registers is derived from a schedule graph G_s, by breaking every path in G_s that is longer than the optimal clock period P. For paths having a length less than P, no register is required because operations chaining is assumed.

For the circuit in Figure 2.1, applying the algorithm in [8] for register placement on the schedule graph G_s in Figure 2.2, starting from v_1, gives the placement of registers and their schedules as it is summarized by Table 2.2. Using the results of this table, we present the placement of registers in the circuit as depicted by Figure 2.3. The number of registers that are placed is 8 and the number of phases is 4.

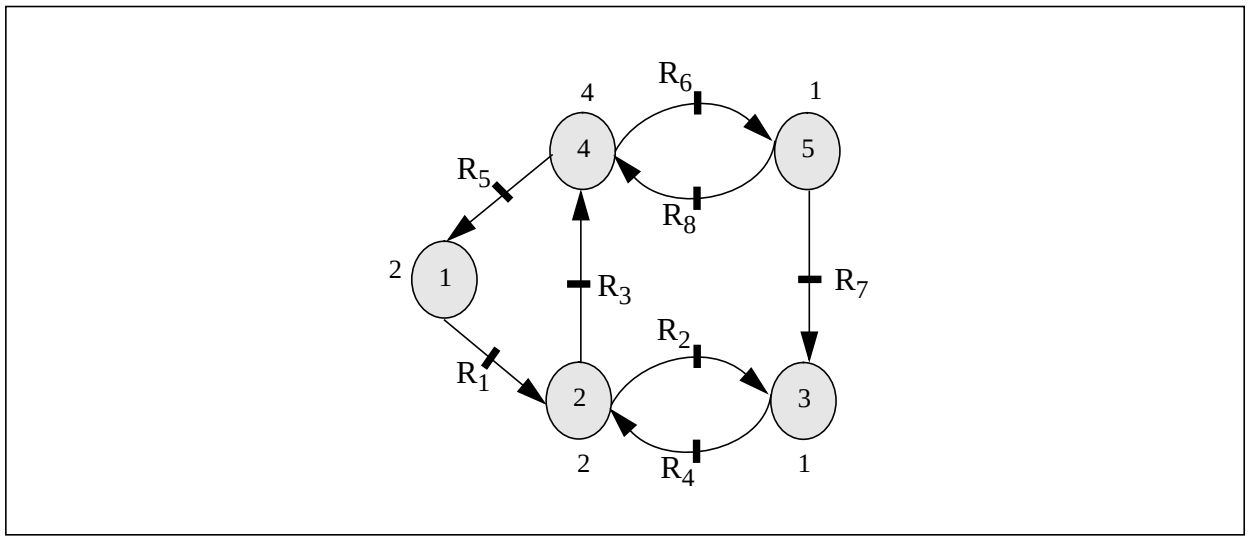


Figure 2.3 : Registers placement using ALAP.

Table 2.2 : Registers placement and their schedules.

Registers	Registers Placement	Schedule
R_1	$e_{1, 2}$	2
R_2	$e_{2, 3}$	1
R_3	$e_{2, 4}$	0
R_4	$e_{3, 2}$	2
R_5	$e_{4, 1}$	0
R_6	$e_{4, 5}$	3
R_7	$e_{5, 3}$	1
R_8	$e_{5, 4}$	0

2.4- Reducing register requirements

As explained in Section 2.3, the algorithm to place registers breaks only paths that are longer than the optimal clock period *P*, because operations chaining is assumed. Recall that this algorithm works on a schedule graph, G_s, where its edges are weighted as defined by (8). By examining equation (8), no register is required between *u* and *v* if $w_s(e_{u, v})$ is less than *P*. Hence, having $w_s(e_{u, v})$ as small as possible for each edge $e_{u, v}$ in G_s may allow to decrease the number of registers that will be placed by this algorithm. Having edges with this characteristic depends on the schedule chosen to construct the graph G_s. The problem of reducing the number of registers thus transforms to determining a schedule that allow $w_s(e_{u, v})$ to be as small as possible for each $e_{u, v}$ in *G*. To achieve that new goal, we develop a mathematical formulation summarized in Figure 2.4.

Let us define:

$$\varepsilon_{u, v} \geq w_s(e_{u, v}), \tag{9}$$

for each edge $e_{u, v}$. Minimizing $\sum \varepsilon_{u, v}$ over all edges tends to minimize the number of edges where a register will be necessary. The combination of (8) and (9) gives the inequality (10) in Figure 2.4. In this figure, inequality (11) represents the data dependency constraints that every valid schedule must satisfy. In the mathematical formulation in Figure 2.4, the variables are the schedule time of the first instance of each vertex *v* in *V* (i.e., $s_0(v)$) and the $\varepsilon_{u, v}$, which are defined for all vertices *u* and *v* such that the edge $e_{u, v}$ is in *E*.

For the circuit in Figure 2.1, 0, -2, -4, -4, and -1 is a valid schedule for the functional

elements 1, 2, 3, 4 and 5, respectively. This schedule is an optimal solution for the formulation in Figure 2.4. Using this schedule, the optimized circuit presented in Figure 2.5 has the same functionality as the original one. But, only 7 registers and 3 phases are needed instead of 8 registers and 4 phases that are required for the circuit in Figure 2.3; recall that the circuit in Figure 2.3 is derived using an ALAP schedule. The schedule time of registers 1, 2, 3, 4, 5, 6, and 7 in the circuit in Figure 2.5 is 0, 0, 2, 0, 3, 0, and 0, respectively. Registers 1 and 2 are in the output of the same functional element and they have the same schedule time. Hence, only one of them is needed. We have the same thing for the case of registers 6 and 7. Hence, only 5 registers are required in this circuit

To solve the formulation in Figure 2.4, we examine three cases. Case 1: when there is no restriction on the type of each variable in this formulation. Case 2: when the variables have to be integers, and P is integer. Case 3: like case 2 but P is rational. But before examining these cases, let us first check if this formulation has a solution, and if it is possible to prune the solution space.

Formulation F_1:

$$Minimize \sum_{\forall e_{u,v} \in E} \varepsilon_{u,v}$$

Subject to:

$$\forall e_{u,v} \in E,\ \varepsilon_{u,v} \geq s_0(v) - s_0(u) + P \cdot w(e_{u,v}) \quad (10)$$

$$\forall e_{u,v} \in E,\ s_0(v) - s_0(u) \geq d(u) - P \cdot w(e_{u,v}) \quad (11)$$

Figure 2.4 : Scheduling for reducing register requirements.

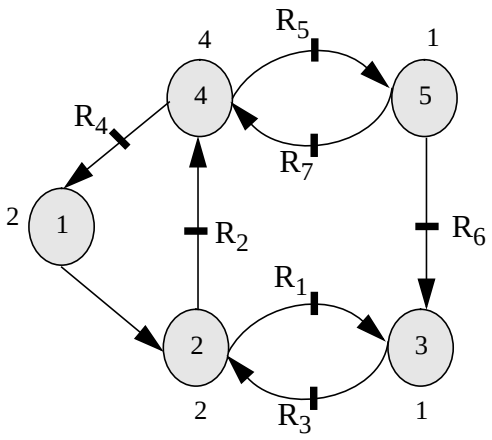


Figure 2.5 : Final circuit using Figure 2.4.

Lemma 2.1 : *Given that P is a feasible clock period, the formulation in Figure 2.4 has always a solution.*

Proof: an ASAP (or an ALAP) schedule satisfies (10) and (11). Hence, this schedule is a solution to the mathematical formulation in Figure 2.4.

❑

Lemma 2.2 : *If P and d(v) are integers, then for each v in V, ASAP(v) and ALAP(v) are integers.*

Proof: The procedure to find the ASAP and ALAP schedules for each vertex *v* in *V* is given in Section 2.2.4. Also, we have seen that this can be done by applying the Bellman-Ford's algorithm for longest paths on the graph G_P or *G'*, where each $w_P(e_{u,v})$ is defined as in (5). Because *P* and, for each *v* in *V*, *d*(*v*) are integers, then $w_P(e_{u,v})$ is integer. Hence, this algorithm produces integer values for ASAP or ALAP schedule.

❑

The ASAP and ALAP schedules may be used to prune the solution space of the mathematical formulation in Figure 2.4. This can be done by letting: $ASAP(v) \le s_0(v) \le ALAP(v)$ for each *v* in *V*.

The following subsections examine how to solve the mathematical formulation in Figure 2.4 for the three cases of interest.

2.4.7- Case 1: No restriction on the type of variables

In this case, the mathematical formulation in Figure 2.4 is a linear program. Hence, it can be solved efficiently by using the simplex method [28] or by using one of the two polynomial-time methods such that the ellipsoid method [62] or the interior point method [59]. In this paper, we have used the LP_Solve tool [74] (which is in the public domain) to obtain the experimental results.

2.4.8- Case 2: All variables and *P* are integers

In this case, the mathematical formulation in Figure 2.4 is an integer linear program which is NP-hard in the general case. But, in this mathematical formulation, the right hand sides of the inequalities are integers. From linear programming theory [100], we know that if the constraint matrix of an integer linear program is totally unimodular, and if the right hand sides of the inequalities (as in Figure 2.4) are integers, then the integer linear program and its relaxation, obtained by ignoring the constraint *integer*, have the same optimal solution. The relaxed formulation is a linear program and hence it can be solved as it was discussed in Section 2.4.7.

Now, to solve our integer linear program as a linear program, we must prove that the constraint matrix, *A*, of the formulation in Figure 2.4 is totally unimodular.

Theorem 2.1 : *The constraint matrix, A, of the formulation in Figure 2.4 is totally unimodular.*

To prove Theorem 2.1, let us first recall the following theorem which is proved in [15].

Theorem 2.2 : *A matrix, X, is totally unimodular if and only if for every (square) Eulerian submatrix, Y, of X, we have that the sum of the entries of X divides by 4.*

Proof of Theorem 2.1: Let *m* be the number of edges and *n* be the number of vertices in the directed cyclic graph, *G*, modeling a circuit. The constraint matrix, *A*, of the mathematical formulation in Figure 2.4 is:

$$A = \begin{bmatrix} A_1 & A_2 \\ A_3 & A_4 \end{bmatrix}$$

where A_1, A_2, A_3, and A_4 are matrices of size $m \times m$, $m \times n$, $m \times m$ and $m \times n$, respectively. The matrix A_1 is the identity matrix. A_3 is the zero matrix. A_2 is the incidence matrix of *G* and $A_4 = -A_2$. Each line of A_2 contains exactly an 1 and an -1; the other entries of this line are equal to zero.

Let *B* be an arbitrary square eulerian submatrix of *A*. The sum of the entries of each line of *B* is even; all columns of *B* have this propriety too. Consequently, *B* and A_1 do not have any common element. Now, let us discuss the rest of the cases. First, it is well known that every incidence matrix is totally unimodular [6]. It is evident that the zero matrix is totally unimodular too. Hence, A_2, A_3, and A_4 are totally unimodular. Now, if *B* is a submatrix of A_2, or A_3, or A_4, then it is totally unimodular; hence, Theorem 2.2 is satisfied and *A* is totally unimodular. For the case where *B* is formed by elements in A_2 and by elements in A_4, or by elements in A_3 and by elements in A_4, then by the eulerian condition, each line of *B* has exactly one entry equal to 1 and one entry equal to -1 (the other entries are equal to zero); the sum of the entries of this line is equal to zero; consequently, the sum of the all entries of *B* is equal to zero and then it divides by 4; in this case, the condition of Theorem 2.2 is satisfied; hence, we have that *A* is totally unimodular.

❑

As a summary, we have proved that the constraint matrix of the mathematical formulation in Figure 2.4 is totally unimodular. The right hand sides of the inequalities in this formulation are integers. Consequently, solving the corresponding linear program, obtained by ignoring the *integer* constraint, gives the optimal solution of our integer linear program; this solution is guaranteed to

be integer [100].

2.4.9- Case 3: Variables are integers, and *P* is rational

In this case, the mathematical formulation in Figure 2.4 is an integer linear program. But, because the right hand side of the inequalities of this formulation are not integers, we cannot solve this formulation as it was done in Section 2.4.8. In [46, 3], a theorem says that given a valid periodic schedule *s*, the schedule *s** defined by $s^*_n(v) = \lfloor s_n(v) \rfloor$, where $n \in N$ and $v \in V$, is a *k*-periodic schedule with the same throughput. Hence, to solve this formulation, we can ignore the constraint *integer* and solve the resulting linear program as it was done in Section 2.4.7. Then, the *k*-periodic schedule, *s**, can be determined.

2.5- Reducing the number of phases

In the method based on software pipelining [8], outlined in Section 2.1, once registers are placed and their schedule is fixed, the phases are determined. If a register, *R*, is placed on edge $e_{u,v}$, then its schedule time is:

$$s_v(R) = s_0(v) \bmod P, \tag{12}$$

where *P* is the optimal clock period. The number of phases is the number of the different $s_v(R)$ for all the required registers. Let us now analyze how to determine a schedule that gives a small number of phases. By equation (12), we have that:

$$\exists k_v \in N : s_v(R) = s_0(v) - k_v \cdot P, \tag{13}$$

and

$$s_v(R) < P. \tag{14}$$

By letting,

$$\delta_{u,v} \geq \left| s_v(R) - s_u(R) \right|, \tag{15}$$

for each two pair of different vertex *u* and *v* in *V*, and by minimizing $\sum \delta_{u,v}$, then the number of the phases tends to be reduced. Because the schedule we want to determine for producing a small number of phases must be valid, the inequalities described by (6) must not be violated. Using (13) and (15), we have that:

$$\delta_{u,v} \geq \left| s_0(v) - s_0(u) + (k_u - k_v) \cdot P \right|. \tag{16}$$

Putting together the equalities and inequalities: (16), (14) and (6), we have the mathematical formulation, for determining a valid schedule with a small number of phases, which is presented in Figure 2.6. In this figure, inequalities (17) and (18) are equivalent to (16); by examining inequalities (17) and (18), we conclude that only one of them is required. Inequality (19) is derived from (13) and (14). Inequality (19) is not in the standard form of linear programs, but if P is integer, then it may be replaced by $s_0(v) - k_v \cdot P \leq P - 1$. Inequality (20) is equivalent to (6). If P is integer, then this formulation is a mixed integer linear program. But, relaxing this formulation by ignoring the term $(\pm(k_u - k_v) \cdot P)$ in (17) and (18) and restricting the inequalities to the vertices having edges between them, and by removing (19), we have the mathematical formulation in Figure 2.7. In this formulation, the variables are the schedule time of the first instance of each vertex v in V (i.e., $s_0(v)$) and the $\delta_{u,v}$, which are defined for all vertices u and v such that the edge $e_{u,v}$ is in E. The resolution of the linear program in Figure 2.7 may be done as we have done for the case of the formulation in Figure 2.4.

Formulation F_2:

$$Minimize \sum_{\forall u \in V,\ \forall v \in V,\ u \neq v} \delta_{u,v}$$

Subject to:

$$\forall u \in V,\ \forall v \in V,\ \forall k_u \in N,\ \forall k_v \in N \text{ and } u \neq v:\ \delta_{u,v} \geq s_0(v) - s_0(u) + (k_u - k_v) \cdot P \quad (17)$$

$$\forall u \in V,\ \forall v \in V,\ \forall k_u \in N,\ \forall k_v \in N \text{ and } u \neq v:\ \delta_{u,v} \geq s_0(u) - s_0(v) + (k_v - k_u) \cdot P \quad (18)$$

$$\forall v \in V,\ \forall k_v \in N: s_0(v) - k_v \cdot P < P \quad (19)$$

$$\forall e_{u,v} \in E,\ s_0(v) - s_0(u) \geq d(u) - P \cdot w(e_{u,v}) \quad (20)$$

Figure 2.6 : Scheduling for reducing the number of phases.

For the circuit in Figure 2.1, 0, -2, -2, -4, and -4 is a valid schedule for the functional elements 1, 2, 3, 4 and 5, respectively. This schedule is an optimal solution for the formulation in Figure 2.7. Using this schedule, the optimized circuit presented in Figure 2.8 has the same functionality as the original one. It operates using 8 registers and 2 phases instead of 8 registers and 4 phases that are needed for the circuit in Figure 2.3; note that the circuit in Figure 2.3 is obtained using an ALAP schedule. The schedule time of registers 1, 2, 3, 4, 5, 6, 7 and 8 in the circuit in

Figure 2.8 is 2, 2, 0, 2, 0, 0, 2, and 0, respectively. In this circuit, registers 5 and 6 are in the output of the same functional element, and they have the same schedule time; hence, only one of them is needed; consequently, the circuit can operate with 7 registers instead of 8.

Formulation F_3:

$$Minimize \sum_{\forall e_{u,v} \in E} \delta_{u,v}$$

Subject to:

$$\forall e_{u,v} \in E,\ \delta_{u,v} \geq s_0(v) - s_0(u) \quad (21)$$

$$\forall e_{u,v} \in E,\ \delta_{u,v} \geq s_0(u) - s_0(v) \quad (22)$$

$$\forall e_{u,v} \in E,\ s_0(v) - s_0(u) \geq d(u) - P \cdot w(e_{u,v}) \quad (23)$$

Figure 2.7 : The relaxed formulation of Figure 2.6.

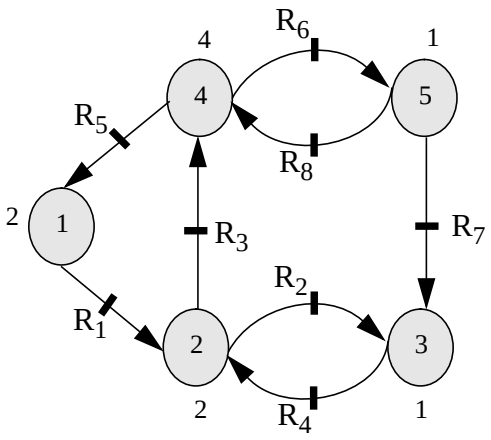


Figure 2.8 : Final circuit using Figure 2.7.

2.6- Experimental results

We have implemented the two methods presented in Sections 2.4 and 2.5 in a tool coded in C++, called CircuitOptimizer, which has the architecture described by Figure 2.9. Recall that the first method determines schedules for reducing register requirements, while the second one gives schedules for reducing the number of required phases. Starting from a given directed cyclic graph specification of a synchronous sequential circuit, the tool determines the optimal clock period and the ASAP and ALAP schedules, which could be used by the other components of the tool. Then, depending on the choice of the type of schedule required, automatic generation of the constraints of the mathematical formulation of this schedule is done. The LP_Solve tool [74] (in the public domain) is used to solve the generated mathematical formulation. Finally, the schedule found by the LP_Solve is parsed and used to place registers. The phases and their number are then determined.

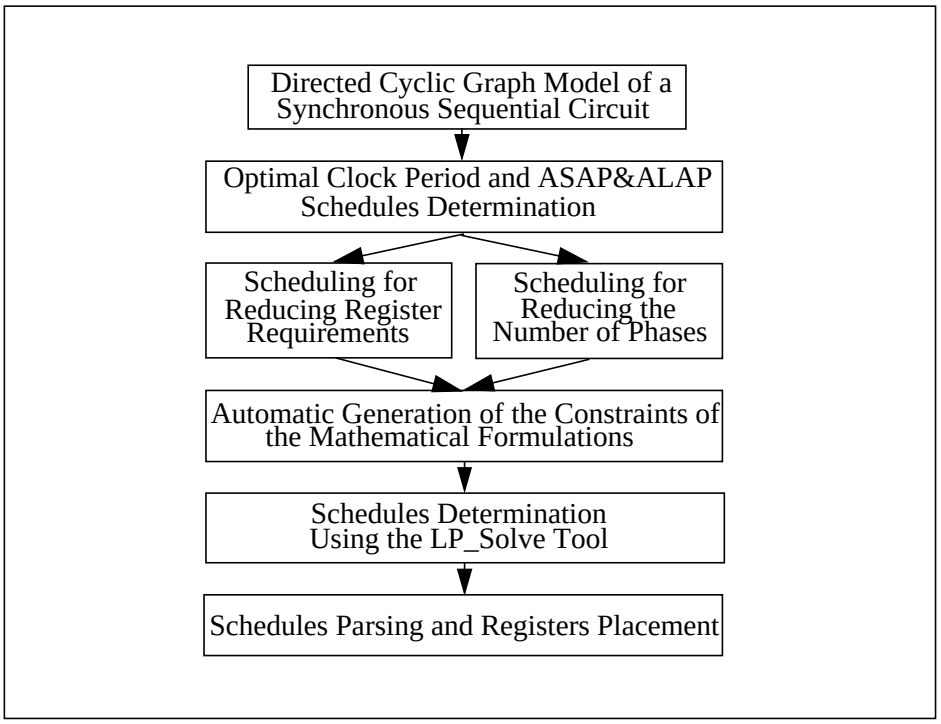


Figure 2.9 : The architecture of the CircuitOptimizer Tool.

To test the effectiveness of the methods that we have developed to determine schedules for reducing register and phase requirements, we have experimented the CircuitOptimizer tool on benchmarks selected from the ISCAS89 set. Table 2.3 reports results when the tool have been used to place a small number of registers. The results of the case when the target is to place registers for reducing the number of required phases are presented in Table 2.4. To determine the gain obtained by using our methods, we compare the results of these methods with the results of the original method [8] that uses ALAP. In Table 2.3, the number of the registers is reduced by a factor ranging from 24% to 38%. Note that even though it was not the objective of the method to reduce the number of phases, we nevertheless obtained reductions ranging from 19% to 67%. In Table 2.4, a substantial reduction of the number of phases has been obtained; the gain factor is between 12% and 70%; the number of registers is also reduced, except for the case of the circuit S5378 where it has increased as a result of reducing the number of phases. All the results in Tables 2.3 and 2.4 have been obtained in less than 20 minutes on an UltraSPARC-10 with 1GB RAM.

Table 2.3 : Reducing register requirements.

	Arbitrary schedule: ALAP		**Schedule produced by our method**		**Gain**	
	# registers	**# phases**	**# registers**	**# phases**	**# registers**	**# phases**
S344	131	15	96	7	27%	53%
S641	142	16	90	13	37%	19%
S1423	422	65	320	47	24%	28%
S5378	1033	53	692	27	33%	49%
S9234	1042	48	643	16	38%	67%

Table 2.4 : Reducing phase requirements.

	Arbitrary schedule: ALAP		**Schedule produced by our method**		**Gain**	
	# registers	**# phases**	**# registers**	**# phases**	**# registers**	**# phases**
S344	131	15	79	6	40%	60%
S641	142	16	90	14	37%	12%
S1423	422	65	366	31	13%	52%
S5378	1033	53	1168	16	-13%	70%
S9234	1042	48	849	16	18%	67%

2.7- Conclusion

In this paper, we showed that choosing a good schedule has an impact on the number of registers that must be placed in the circuit derived using software pipelining techniques, and on the required number of phases. Reducing the number of registers contributes to the minimization of the area occupied by the circuit and reduces its power consumption, while reducing the number of phases reduces the complexity of the clock generation and distribution tasks.

We have developed two polynomial-time-solvable methods for determining schedules to reduce register and phase requirements. As demonstrated by the experimental results using a subset of the ISCAS89 benchmarks, the methods have proved to be very efficient for reducing the number of registers that must be inserted in the final circuit, and its number of phases.

Chapitre 3

Méthodes efficaces pour la réduction du nombre de registres et du nombre de phases pour des circuits synchrones à débit maximal

Pour le design des circuits numériques de grande taille, des algorithmes ayant une complexité temporelle polynômiale ne sont pas nécessairement efficaces. À titre d'exemple, prenons le cas des algorithmes du retiming [71] où différents efforts (comme nous l'avons présenté à la Section 1.6) ont été faits pour améliorer leur efficacité.

Au Chapitre 2, nous avons présenté deux formulations mathématiques pour résoudre les problèmes P_1 et P_2 présentés dans la Section 1.1 du Chapitre 1. Dans ce chapitre, nous nous concentrons sur la transformation de ces formulations en une formulation d'un problème de flot à coût minimum, puisque ce dernier possède des algorithmes de résolution efficaces. Nous comparons expérimentalement le temps d'exécution pour résoudre les formulations originales et les formulations transformées. Pour résoudre les programmes linéaires obtenus, nous utilisons un outil du domaine publique qui est une implantation de l'algorithme du *simplex*. Même si nous n'avons pas utilisé des algorithmes spécialisés pour résoudre des formulations du problème de flot à coût minimum, les résultats expérimentaux montrent que ces dernières peuvent être résolues plus rapidement que les formulations originales. L'usage d'un algorithme du *simplex* peut influencer le gain découlant de l'utilisation de l'approche proposée.

Nous poursuivons ce chapitre par la présentation de l'article [20].

Efficient Methods for Reducing Register and Phase Requirements for Synchronous Circuits Derived Using Software Pipelining Techniques

Noureddine Chabini[1], El Mostapha Aboulhamid[1], and Yvon Savaria[2]

1: *LASSO, DIRO, Université de Montréal, C.P. 6128, Centre ville, Montréal, Qc, Canada, H3C 3J7. Email:{chabinin, aboulham}@iro.umontreal.ca*

2: *GRM, DGEGI, École Polytechnique de Montréal, C.P. 6079, Succ. Centre-ville, Montréal, Qc, Canada, H3C 3A7. Email: savaria@vlsi.polymtl.ca*

Abstract

Recently, we have presented two mathematical formulations and procedures to solve them that apply to the problem of determining schedules, to reduce register and phase requirements for multi-phase synchronous circuits derived using software pipelining techniques. In this paper, we show how to transform these formulations to minimum cost network flow problems, which can be solved efficiently. We show that the resulting formulations can be solved by algorithms of time complexity $O(n^3 log(n))$ for a network of n nodes. Although we have not used a specialized algorithm to solve the new formulations, experimental results on a subset of the ISCAS89 benchmarks show that these formulations can be solved much faster than the original formulations, where the same algorithm based on the simplex method is used.

3.1- Introduction

In order to minimize the clock period of a synchronous sequential circuit, this latter is modeled (as in [88, 11, 71]) as a directed cyclic graph $G = (V, E, d, w)$, where V is the set of functional elements in the circuit, and E is the set of edges that represent interconnections between vertices. Each vertex v in V has a non-negative integer propagation delay $d(v) \in N$. Each edge $e_{u, v}$, from u to v, in E is weighted with a register count $w(e_{u, v}) \in N$, representing the number of registers on the wire between u and v.

Figure 3.1 presents an example of a circuit and its directed cyclic graph model. In this figure, large rectangles represent functional elements, and small rectangles represent registers.

Wires are oriented to show the propagation direction of the signals. The propagation delay of each functional element of this circuit is specified as a label on the left of each large rectangle.

Software pipelining has been proved to be a powerful technique for increasing the instruction-level parallelism for parallel processors. It has recently been used for optimizing clocked circuits [8, 11], where the input circuit is a synchronous circuit with a single-phase clock period, like the circuit in Figure 3.1, which has a period of $6 = d(v_4) + d(v_1)$. The resulting circuit is a multi-phase clocked circuit, where all clocks have the same period. The method may be described as follows. First, the optimal clock period, *P*, is determined, and a schedule of all the functional elements of the circuit is computed. Second, in order to preserve the behavior of the original circuit, registers are placed independently of their initial placement. The placement of registers is done using the computed schedule. Finally, once the registers are placed, the phases are determined. Only the schedule time of the first instance of each functional element has to be determined, since the schedule is supposed to be periodic. This latter has been defined [8, 11, 3] as a periodic function $s : N \times V \rightarrow Q$ of period *P*, where $s_n(v) \equiv s(n, v)$ denotes the schedule time of the n^{th} iteration of operation *v*. In multi-phase flip-flop based circuits, the schedule time is the start time of the operation. To be a valid schedule, this latter must satisfy the data dependency constraints, which can be expressed mathematically as follows:

$$\forall n \in N, \forall e_{u, v} \in E : s_{n + w(e_{u, v})}(v) \geq s_n(u) + d(u). \qquad (1)$$

For the method described above, more details as well as an illustrative example can be found in [18, 8]. A comparison of that method and some methods based on retiming [71] is provided in [8].

In that method [8, 11], it was question of how to determine a schedule that allows to reduce the number of registers, and the number of clock phases in the final design. Decreasing the number of registers contributes to minimizing the area occupied by the circuit and reduces its power consumption, while decreasing the number of phases reduces the complexity of the clock generation and distribution tasks.

To determine the required schedule, we have recently proposed in [18] two mathematical formulations, and presented procedures to solve them. In this paper, we show how we can transform these formulations to a formulation of the minimum cost network flow problem. Since this problem can be solved efficiently, this implies that our original formulations can also be solved

efficiently. We show that they can be solved by algorithms of time complexity $O(n^3\log(n))$ for a circuit of n computational elements, which is the time complexity of the original method based on the software pipelining technique presented in [8, 11]. Since the time to market is a serious constraint during the design of digital systems, designing algorithms with low time complexity is very important. Although we have not used a specialized algorithm to solve the new formulations, experimental results on a subset of the ISCAS89 benchmarks show that these formulations can be solved much faster than the original formulations, where the same algorithm based on the simplex method is used.

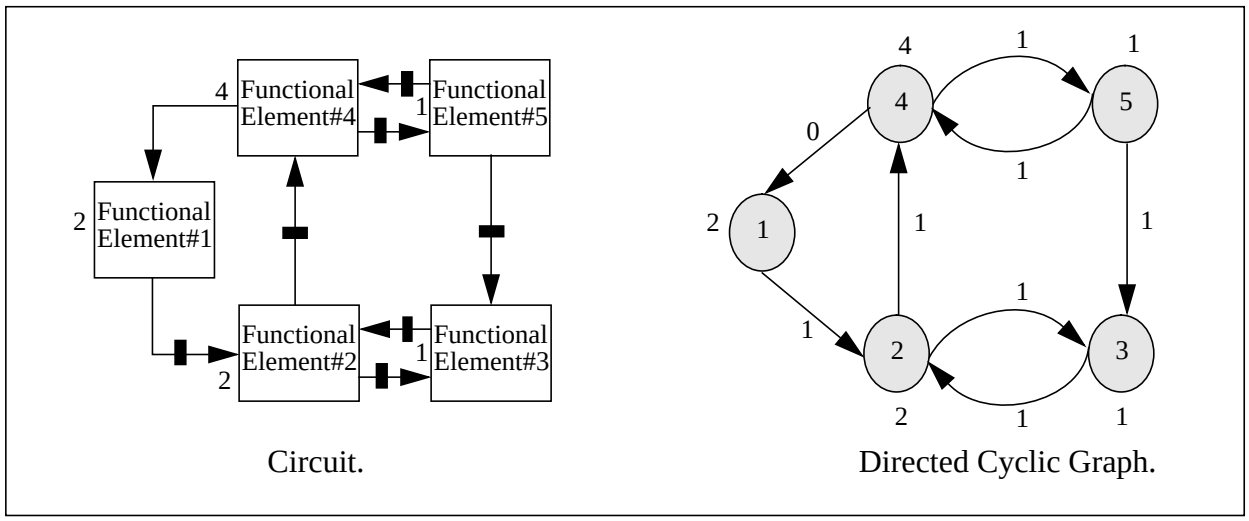


Figure 3.1 : Sample circuit and its directed cyclic graph model.

The rest of this paper is organized as follows. Sections 2 and 3 present the transformation, to a minimum cost network flow problem, of the two mathematical formulations for determining schedules to reduce register and phase requirements. Experimental results are presented in Section 4. Section 5 concludes the paper.

3.2- A Minimum Cost Network Flow Formulation for the Problem of Determining Schedules for Reducing Register Requirements

As mentioned in Section 3.1, to determine schedules for reducing register requirements for circuits derived using software pipelining techniques, we have developed in [18] the mathematical formulation presented in Figure 3.2. In this formulation, the variables are the $\varepsilon_{u,v}$'s and the

schedule time $s_0(k)$ of each computational element *k* of the circuit. *P* is the optimal clock period of the circuit. The other parameters are as defined in Section 3.1. Equation (2) is used to reduce the required number of registers. Equation (3) is equivalent to equation (1). Due to space limitation, the reader is referred to [18] for details.

$$\text{Minimize} \sum_{\forall e_{u,v} \in E} \varepsilon_{u,v}$$

Subject to:

$$\forall e_{u,v} \in E,\ \varepsilon_{u,v} + s_0(u) - s_0(v) \geq P \cdot w(e_{u,v}) \quad (2)$$

$$\forall e_{u,v} \in E,\ s_0(v) - s_0(u) \geq d(u) - P \cdot w(e_{u,v}) \quad (3)$$

Figure 3.2 : Scheduling for reducing register requirements.

Let $\delta^+(u)$ and $\delta^-(u)$ be the set of successors and the set of predecessors of *u*, respectively. The mathematical formulation of the dual of the formulation in Figure 3.2 can be written as presented in Figure 3.3.

$$\text{Maximize} \left(\left(\sum_{\forall e_{u,v} \in E} P \cdot w(e_{u,v}) \cdot \Psi_{u,v} \right) + \left(\sum_{\forall e_{u,v} \in E} (d(u) - P \cdot w(e_{u,v})) \cdot \Phi_{u,v} \right) \right)$$

Subject to:

$$\forall e_{u,v} \in E,\ \Psi_{u,v} = 1 \quad (4)$$

$$\forall u \in V,\ \sum_{v \in \delta^+(u)} \Psi_{u,v} - \sum_{v \in \delta^-(u)} \Psi_{v,u} - \sum_{v \in \delta^+(u)} \Phi_{u,v} + \sum_{v \in \delta^-(u)} \Phi_{v,u} = 0 \quad (5)$$

$$\forall e_{u,v} \in E,\ \Psi_{u,v} \geq 0,\ \Phi_{u,v} \geq 0 \quad (6)$$

Figure 3.3 : The dual formulation of the formulation in Figure 3.2.

From equation (4), we have that:

$$\sum_{\forall e_{u,v} \in E} P \cdot w(e_{u,v}) \cdot \Psi_{u,v}$$

is a constant. Hence, this term can be removed from the objective function. Consequently, we have that:

$$Maximize \left(\left(\sum_{\forall e_{u,v} \in E} P \cdot w(e_{u,v}) \cdot \Psi_{u,v} \right) + \left(\sum_{\forall e_{u,v} \in E} (d(u) - P \cdot w(e_{u,v})) \cdot \Phi_{u,v} \right) \right)$$

is equivalent to:

$$Maximize \left(\sum_{\forall e_{u,v} \in E} (d(u) - P \cdot w(e_{u,v})) \cdot \Phi_{u,v} \right)$$

which is also equivalent to:

$$Minimize \left(\sum_{\forall e_{u,v} \in E} (P \cdot w(e_{u,v}) - d(u)) \cdot \Phi_{u,v} \right)$$

Let $b_u = \left|\delta^+(u)\right| - \left|\delta^-(u)\right|$. By definition of b_u and by equations (4) and (5), we have that: $\forall u \in V$,

$$\sum_{v \in \delta^+(u)} \Phi_{u,v} - \sum_{v \in \delta^-(u)} \Phi_{v,u} = \sum_{v \in \delta^+(u)} \Psi_{u,v} - \sum_{v \in \delta^-(u)} \Psi_{v,u}$$
$$= \left|\delta^+(u)\right| - \left|\delta^-(u)\right| = b_u$$

With the all previous modifications, the formulation in Figure 3.3 can simplified to the formulation presented in Figure 3.4, which is a formulation of the minimum cost network flow problem [65].

Theorem 3.1 : *The formulation in Figure 3.2 can be solved by algorithms of time complexity $O(n^3 log(n))$ for a circuit of n computational elements.*

Proof: The formulation in Figure 3.4 is a transformed dual of the formulation in Figure 3.2. Hence, solving one of them provides the solution to the other. The former formulation is a formulation of the minimum cost network flow problem, which can be solved efficiently by using

one of the methods in [1]. An algorithm of time complexity $O(n^3 log(n))$ for a network of n nodes, to solve that problem, can be found in [45]. ❑

$$Minimize \left(\sum_{\forall e_{u,v} \in E} (P \cdot w(e_{u,v}) - d(u)) \cdot \Phi_{u,v} \right)$$

Subject to:

$$\forall u \in V, \quad \sum_{v \in \delta^+(u)} \Phi_{u,v} - \sum_{v \in \delta^-(u)} \Phi_{v,u} = b_u \qquad (7)$$

$$\forall e_{u,v} \in E, \Phi_{u,v} \geq 0 \qquad (8)$$

Figure 3.4 : The simplified dual formulation of the formulation in Figure 3.2.

3.3- A Minimum Cost Network Flow Formulation for the Problem of Determining Schedules for Reducing the Number of Phases

To determine schedules for reducing the required number of phases for circuits derived using software pipelining techniques, we have developed in [18] the mathematical formulation presented in Figure 3.5. In this formulation, the variables are the $\varepsilon_{u,v}$'s and the schedule time $s_0(k)$ of each computational element k of the circuit. P is the optimal clock period of the circuit. The other parameters are as defined in Section 3.1. Equations (9) and (10) are used to reduce the required number of phases. Equation (11) is equivalent to equation (1). Due to space limitation, the reader is referred to [18] for details.

Figure 3.6 presents the mathematical formulation of the dual of the problem in Figure 3.5, where $\delta^+(u)$ and $\delta^-(u)$ are as defined in Section 3.2. In this formulation, since $\forall e_{u,v} \in E, \varphi_{u,v} = \Psi_{u,v}$, then equation (14) can be simplified to:

$$\forall u \in V, \quad \sum_{v \in \delta^+(u)} \Phi_{u,v} - \sum_{v \in \delta^-(u)} \Phi_{v,u} = 0$$

Since $\varphi_{u,v}$ and $\Psi_{u,v}$ do not appear in the objective function, then equations (12) and (13) can be removed from the formulation. Also, the maximization of the objective function can be

transformed to:

$$Minimize \left(\sum_{\forall e_{u,v} \in E} (P \cdot w(e_{u,v}) - d(u)) \cdot \Phi_{u,v} \right)$$

With the all above modifications, the final dual of the formulation in Figure 3.5 is presented in Figure 3.7, which is a formulation of the minimum cost network flow problem .

$$Minimize \sum_{\forall e_{u,v} \in E} \varepsilon_{u,v}$$

Subject to:

$$\forall e_{u,v} \in E, \varepsilon_{u,v} + s_0(u) - s_0(v) \geq 0 \quad (9)$$

$$\forall e_{u,v} \in E, \varepsilon_{u,v} + s_0(v) - s_0(u) \geq 0 \quad (10)$$

$$\forall e_{u,v} \in E, s_0(v) - s_0(u) \geq d(u) - P \cdot w(e_{u,v}) \quad (11)$$

Figure 3.5 : Scheduling for reducing the required number of phases.

$$Maximize \left(\sum_{\forall e_{u,v} \in E} (d(u) - P \cdot w(e_{u,v})) \cdot \Phi_{u,v} \right)$$

Subject to:

$$\forall e_{u,v} \in E, \varphi_{u,v} = 1 \quad (12)$$

$$\forall e_{u,v} \in E, \Psi_{u,v} = 1 \quad (13)$$

$$\forall u \in V, \left(\sum_{v \in \delta^+(u)} (\varphi_{u,v} - \Psi_{u,v}) - \sum_{v \in \delta^-(u)} (\varphi_{v,u} - \Psi_{v,u}) \right) - \sum_{v \in \delta^+(u)} \Phi_{u,v} + \sum_{v \in \delta^-(u)} \Phi_{v,u} = 0 \quad (14)$$

$$\forall e_{u,v} \in E, \varphi_{u,v} \geq 0, \Psi_{u,v} \geq 0, \Phi_{u,v} \geq 0 \quad (15)$$

Figure 3.6 : The formulation dual of the formulation in Figure 3.5.

$$Minimize \left(\sum_{\forall e_{u,v} \in E} (P \cdot w(e_{u,v}) - d(u)) \cdot \Phi_{u,v} \right)$$

Subject to:

$$\forall u \in V, \quad \sum_{v \in \delta^{+}(u)} \Phi_{u,v} - \sum_{v \in \delta^{-}(u)} \Phi_{v,u} = 0 \qquad (16)$$

$$\forall e_{u,v} \in E, \Phi_{u,v} \geq 0 \qquad (17)$$

Figure 3.7 : The simplified dual formulation of the formulation in Figure 3.5.

Theorem 3.2 : *The formulation in Figure 3.5 can be solved by algorithms of time complexity $O(n^3 log(n))$ for a circuit of n computational elements.*

Proof: The same as for Theorem 3.1.

❑

3.4- Experimental Results

To test the effectiveness of our approach for transforming the two mathematical formulations to a formulation of the minimum cost network flow problem, we have experimented these two formulations and the new ones on a subset of the ISCAS89 benchmarks. Results are reported in Tables 3.1 and 3.2. Columns of these tables are as follows, the first column gives the name of the circuit. The CPU times (in second), for solving the primal and the simplified dual formulations, are given in the second and the third columns, respectively. The fourth column gives the speedup obtained, which is defined as the CPU time for solving the primal divided by the CPU time for solving the dual. The LP_Solve tool [74] (in the public domain) is used to solve the mathematical formulations, which are automatically generated by a tool we have developed in [74].

Although we have not used specialized algorithms to solve the two dual formulations, experimental results show that these formulations can be solved much faster than the original formulations, where the same algorithm based on the simplex method is used. Indeed, for the case of determining schedules for reducing register requirements, a speedup ranging from 8 to 10.63 has

been obtained as reported in Table 3.1. A significant acceleration has been obtained in the case of determining schedules for decreasing the required number of phases. As Table 3.2 reports, this acceleration ranges from 19.88 to 135.21.

Table 3.1 : Comparison of the time to solve the primal and the dual problems of determining schedules for reducing register requirements.

	CPU time for solving the primal (sec.)	**CPU time for solving the dual (sec.)**	**Speedup**
S344	0.93	0.1	9.3
S641	1.28	0.16	8
S1423	12.14	1.39	8.73
S5378	102.5	9.64	10.63
S9234	54.95	5.49	10.00
S13207	319.35	30.41	10.50

Table 3.2 : Comparison of the time to solve the primal and the dual problems of determining schedules for reducing the required number of phases.

	CPU time for solving the primal (sec.)	**CPU time for solving the dual (sec.)**	**Speedup**
S344	1.32	0.06	22
S641	1.79	0.09	19.88
S1423	21.38	0.72	29.69
S5378	176.13	3.59	49.06
S9234	313.7	2.32	135.21
S13207	1031.16	12.02	85.78

3.5- Conclusions

Recently, we have proposed two mathematical formulations for the problem of determining schedules for reducing register and phase requirements for circuits derived using software pipelining techniques. In this paper, we have shown that these formulations can be transformed to a formulation of the minimum cost network flow problem. Since this problem can be solved

efficiently, this implies that the two formulation can also be solved efficiently. We have proved that they can be solved with time complexity $O(n^3 log(n))$, which is the time complexity of the original method that optimizes circuits using software pipelining techniques. Experimental results have shown that the new formulations can be solved much faster than the original ones, although we have not used specialized algorithms for the minimum cost network flow problem.

Chapitre 4

Ordonnancement et placement optimal de registres pour des circuits synchrones à débit maximal

Dans ce chapitre, nous nous concentrons sur la résolution du problème P_3 présenté dans la Section 1.1 du Chapitre 1. Plus précisément, notre objectif est de résoudre simultanément les deux problèmes: (1) détermination d'ordonnancement des éléments de calcul du circuit à optimiser et (2) placement d'un nombre minimal de registres dans celui-ci. Pour atteindre cet objectif, nous développons une formulation mathématique et nous l'utilisons pour développer deux méthodes de résolution. La première méthode est un programme linéaire mixte permettant d'avoir la solution optimale. La deuxième méthode est une heuristique formulée sous forme d'un programme linéaire. Nous transformons ce programme linéaire en une formulation de flot à coût minimum puisque ce dernier possède des algorithmes de résolution efficaces. Notre approche est utile pour la résolution des problèmes rencontrés dans [8], mais elle est capable de résoudre des problèmes liés à l'augmentation de la tolérance au "clock skew" rencontrés dans [9]. Nous expérimentons l'efficacité de notre approche sur un ensemble de cas de test largement utilisés dans notre domaine.

Nous poursuivons ce chapitre par la présentation de l'article [22] qui est une version augmentée de l'article [21].

Scheduling and Optimal Register Placement for Synchronous Circuits Derived Using Software Pipelining Techniques

Noureddine Chabini[1], El Mostapha Aboulhamid[1], Yvon Savaria[2]
1: *LASSO, DIRO, Université de Montréal C.P.6128, Suc. Centre ville, Montréal, Qc, Canada, H3C 3J7. Email:{chabinin, aboulham}@iro.umontreal.ca*
2*: GRM, DGEGI, École Polytechnique de Montréal, C.P. 6079, Suc. Centre-ville, Montréal, Qc, Canada, H3C 3A7. Email: savaria@vlsi.polymtl.ca*

Abstract

A method based on software pipelining has been recently proposed to optimize mono-phase clocked sequential circuits. The resulting circuits are multi-phase clocked sequential circuits, where all clocks have the same period. To preserve functionality of the original circuit, registers must be placed according to a correct schedule. This schedule also ensures the maximum throughput. In that method, it is question of (1) how to determine a schedule that requires the minimum number of registers, and (2) how to place these registers optimally. In this paper, problems (1) and (2) are tackled simultaneously. More precisely, we deal with the problem of determining schedules with the minimum register requirements, where the optimal register placement is done during the schedule determination. To optimally solve that problem, we provide a mixed integer linear program that we use to derive a linear program, which is polynomial-time solvable. We show that the dual of this linear program can be transformed to a minimum cost network flow problem, which can be solved more efficiently. The proposed approach can handle clock skew. Experimental results confirm the effectiveness of the approach, and show that significant reductions of the number of registers can be obtained. Also, they confirm that the obtained dual formulation can be solved much faster than its primal.

4.1- Introduction

Basic Retiming has been proposed as an optimization technique for synchronous circuits [71]. This technique changes the location of registers in the circuit in order to achieve one of the following goals: (i) minimizing the clock period, (ii) minimizing the number of registers, or (iii) minimizing the number of registers for a target clock period. It can guarantee the determination of the optimal clock period only for single-phase edge-trigged clocked circuits. To deal with the

multi-phase case, methods based on basic retiming have been proposed [54, 70, 72]. In [70], retiming has ben extended to deal with circuits whose registers are not enabled at the same time. The main idea is that registers controlled by the same phase can be moved across combinational blocks.

It is known that with level-sensitive storage elements (latches), clocked circuits can be made faster and smaller [54, 72] than with edge triggered flip-flops. In [54], methods to minimize the clock period of multi-phase level-sensitive clocked circuits are provided. Also, procedures to derive that kind of circuits from edge-triggered ones are presented. In [72], retiming with multi-phase clocks was proposed. With this method, the phases are fixed before retiming, which can give an optimal clock period if good phases are chosen.

Clock skew is defined as the maximum difference of the delays from the clock source to the clock-pins on storage elements [108]. Clock skew can cause malfunction of clocked circuits. Methods to ensure zero-skew in the design are reported in [68, 108]. However, skews are sometimes used as a tool to improve the performance of clocked circuits [37, 40, 99]. In [40], two linear programs were presented to solve the problem of finding skews to minimize the clock period, and the problem of maximizing skews for a target clock period. The equivalence between clock skews and retiming has been first reported in [40], and a formal proof is provided in [37]. For the work in [37], a clock skew optimization problem is first solved with the objective of minimizing the clock period. Then, obtained skews are transformed to retiming by moving some flip-flops across combinational blocks. For single-phase clocked circuits, a mixed integer linear program to combine retiming and clock skew is devised in [41]. The authors of [9] reported that the tolerance to the clock skew for clocked circuits can be improved by using latches instead of flip-flops. They show that, for multi-phase clocked circuits operating at the optimal clock period P, the maximum tolerance to clock skew is $(P - Dmax)/4$, where $Dmax$ is the delay of the slowest computational element in the circuit.

Software pipelining is a powerful technique for increasing the instruction-level parallelism for parallel processors. This method overlaps the execution of successive iterations. It has recently been used to develop a method for optimizing mono-phase clocked sequential circuits [8]. The resulting circuit is a multi-phase clocked circuit, where all clocks have the same period. That method may be described as follows. First, the optimal clock period is determined, and a schedule

of all the functional elements of the circuit is computed. Second, in order to preserve the behavior of the original circuit, registers are placed, independently of their initial placement, according to that schedule. Finally, once the registers are placed, the phases are determined.

With this method, it is question of (1) how to determine a schedule that produces the minimum number of required registers, and (2) how to place the minimum number of registers even if that schedule is already determined. Solving (1) and (2) is of great interest, since reducing the number of registers allows to reduce the number of control signals, the area of the circuit, and the power consumption.

In [18], the authors have provided two polynomial-time solvable methods to determine schedules for reducing register requirements, and the number of required phases. Compared to the original method [8], these methods proved very efficient in reducing the number of registers and the number of the required phases. Nevertheless, the problem of how to efficiently place registers in the circuit is not addressed.

Scheduling under register constraints has been examined in the literature to generate the code for parallel processors. But, it was assumed that processors are single-phase clocked. This is not the case of the method [8] outlined above.

In this paper, we focus on solving simultaneously problems (1) and (2) that are outlined above. More precisely, we tackle the problem of determining schedules that yield the minimum number of registers, where the optimal register placement is done during the schedule determination. To optimally solve that problem, we provide a mixed integer linear program (MILP), which we use to derive a linear program (LP) that is polynomial-time solvable. Also, we transform the dual of this linear program to a formulation of a minimum cost network flow problem that can be solved more efficiently. Furthermore, we present how the proposed approach can handle clock skew. To test the effectiveness of the approach, we experiment the MILP and the LP on well known benchmarks, and we show the superiority of that approach over the original method [8]. The comparison of the run-time of both the LP and its transformed dual is also done.

This paper is organized as follows. The next section gives some notations and definitions used in this article. Section 3 briefly reviews the registers placement step in the method based on software pipelining, which was outlined above. Also, it shows that the algorithm used to place registers is greedy. The problem we address and its optimal solution are presented in Section 4, and

a linear program for that problem is given in Section 5. The transformation of the dual of this linear program to a formulation of a minimum cost network flow problem is presented in Section 6. Section 7 presents how the proposed approach can handle clock skew and gives some theoretical results. Experimental results are provided in Section 8 and Section 9 concludes the paper.

4.2- Preliminaries

4.2.1- The Cyclic Graph Model

In order to minimize the clock period of a synchronous sequential circuit, it is modeled (as in [8, 71]) as a directed cyclic graph $G = (V, E, d, w)$, where V is the set of functional elements in the circuit, and E is the set of edges which represent interconnections between vertices. Each vertex v in V has a non-negative integer propagation delay $d(v) \in N$, which is assumed to be fixed. Each edge $e_{u, v}$, from u to v, in E is weighted with a register count $w(e_{u, v}) \in N$, representing the number of registers on the wire between u and v.

Figure 4.1 presents an example of a circuit and its directed cyclic graph model. In this figure, large rectangles represent functional elements, and small rectangles represent registers. Wires are oriented to show the propagation direction of the signals. The propagation delay of each functional element of this circuit is specified as a label on the left of each large rectangle. This example will be used through this paper, and will serve to illustrate the initial specification of the problem to solve. The initial specification is in general a synchronous circuit with a single-phase clock. The minimum clock period of the circuit in Figure 4.1 as specified is 6, which is equal to $d(v_4) + d(v_1)$.

4.2.2- Periodic Schedules

We define a schedule s [2, 8] as a function $s : N \times V \rightarrow Q$, where $s_n(v) \equiv s(n, v)$ denotes the schedule time of the n^{th} iteration of operation v. In multi-phase flip-flop based circuits, the schedule time is the start time of the operation. A schedule s is called periodic with period P, if:

$$\forall n \in N, \forall v \in V: s_{n+1}(v) = s_n(v) + P. \quad (1)$$

When there is no resource constraint, a schedule s is said to be valid if and only if the operations terminate before their results are needed. In this case, we say that data dependencies are satisfied, which is equivalent to the following mathematical inequality:

$$\forall n \in N,\ \forall e_{u,v} \in E : s_{n + w(e_{u,v})}(v) \geq s_n(u) + d(u). \tag{2}$$

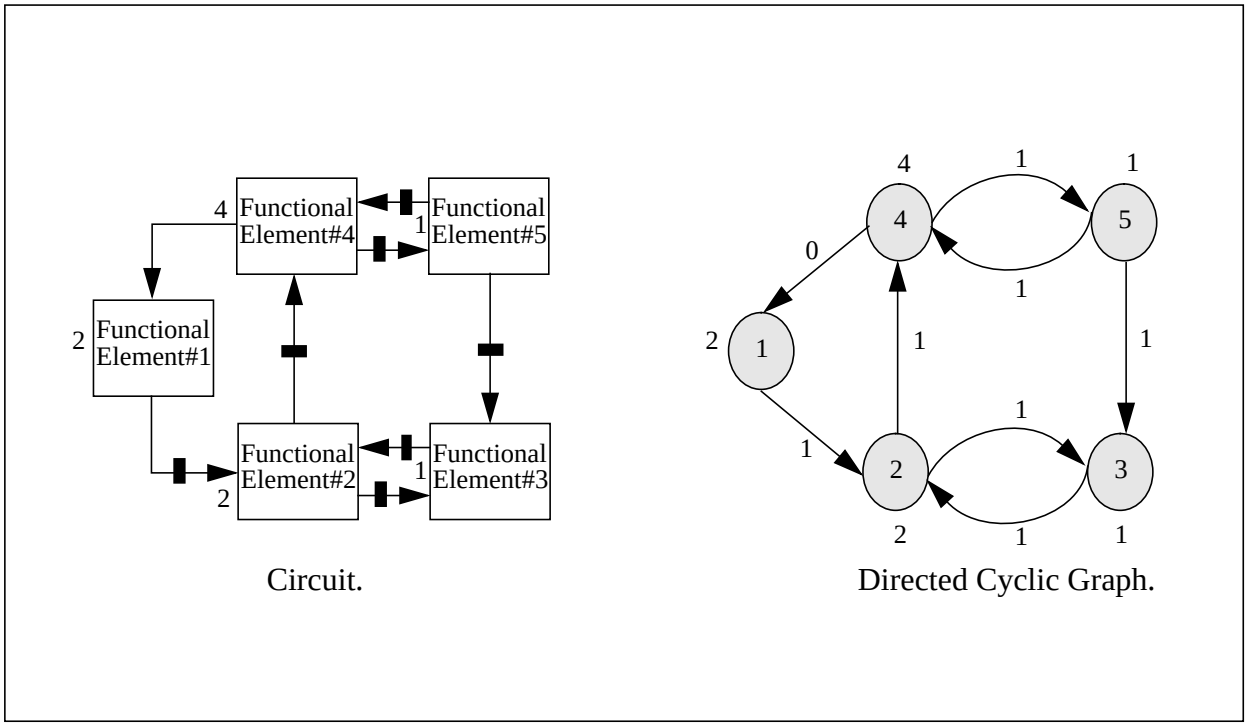


Figure 4.1 : Sample circuit and its directed cyclic graph model.

4.2.3- Maximum Throughput of Synchronous Sequential Circuits

The throughput, *T*, of a synchronous sequential circuit is bounded by the inverse of the length, *P*, of the critical paths in the circuit. Based on data dependencies constraints only, the maximum throughput is [2]:

$$T = Min_{c \in C}\left(\left(\sum_{e_{u,v} \in c} w(e_{u,v})\right) \Big/ \left(\sum_{\forall v \in V \text{ and } e_{u,v} \in c} d(u)\right)\right). \tag{3}$$

where *C* is the set of directed cycles in the directed cyclic graph modeling the circuit. Determining the maximum throughput is a Minimal Cost-to-Time Ratio Cycle Problem [43, 66], which can be solved in the general case in $O(|V| \cdot |E| \cdot \log(|V| \cdot d_{max}))$ [32, 66], where $d_{max} = Max_{v \in V}(d(v))$. A possible method to solve this problem is to iteratively apply Bellman-Ford's algorithm [30] for longest paths on the graph $G_P = (V, E, d, w_P)$ derived from *G* by

letting:

$$w_P(e_{u,v}) = d(u) - P \cdot w(e_{u,v}). \quad (4)$$

where $e_{u,v} \in E$ and $P = 1/T$. A binary search may be used to find the minimal value of P for which there is no positive cycle in G_P [2]. Without loss of generality, for circuits that do not attempt to perform wave pipelining, we assume that P is greater than or equal to the execution delay of each computational element in the circuit.

For the example in Figure 4.1, we have that P = 4. This value corresponds to the cycle defined by vertices v_1, v_2, and v_4.

4.2.4- Schedule for a Given Throughput

From equation (1) and inequality (2), we have that:

$$\forall e_{u,v} \in E,\ s_0(v) - s_0(u) \geq d(u) - P \cdot w(e_{u,v}). \quad (5)$$

In the case of periodic schedules, determining a valid schedule of all the instances of each vertex v in V is equivalent to determining $s_0(v)$ for each v in V, which is also equivalent to determining solutions to the system of inequalities described by (5). To solve this system, the graph G_P, previously described, may be used. As Soon As Possible (ASAP) and As Late As Possible (ALAP) schedules are possible solutions to that system. To find an ASAP schedule, Bellman-Ford's algorithm [30] for longest paths, from a chosen vertex v_x to the others, may be applied on the graph G_P. Finding an ALAP schedule may be done as follows. Step 1, a graph G' has to be derived from G_P by inverting the direction of each edge in G_P. Step 2, Bellman-Ford's algorithm for longest paths, from the vertex v_x to the others, has to be applied on the graph G', where the weights of its edges are defined by equation (4). Finally, step 3, the ALAP schedule is obtained by multiplying each result in step 2 by -1. Relatively to $v_x = v_1$, the ASAP schedules of vertices v_1, v_2, v_3, v_4, and v_5 of the circuit in Figure 4.1 are 0, -2, -4, -4 and -4, respectively. Their ALAP schedules are 0, -2, 1, -4, and -1, respectively.

4.2.5- Schedule Graph

As in [8], a periodic schedule, with period P, is expressed by a schedule graph $G_s = (V, E, d, w_s, P)$. Here V, E and d have the same definition given for the case of the graph G previously defined, and $w_s : E \rightarrow Q$ is a weight function, which associates to each edge $e_{u,v}$ in E the time distance between the schedule times of u and v. Mathematically, $w_s(e_{u,v})$ is defined as follows:

$$\forall e_{u,v} \in E,\ w_s(e_{u,v}) = s_{w(e_{u,v})}(v) - s_0(u). \tag{6}$$

Because s is periodic with period P, equation (6) may be written as follows:

$$\forall e_{u,v} \in E,\ w_s(e_{u,v}) = s_0(v) - s_0(u) + P \cdot w(e_{u,v}). \tag{7}$$

The graph G_s is consistent if and only if for each edge $e_{u,v}$ in E, $w_s(e_{u,v}) \geq d(u)$. This is derived from equation (2). Figure 4.2 shows a consistent schedule graph, where edges are labeled with w_s values, for the circuit in Figure 4.1, using 2, 2, 3, 0, and 0 as a schedule for nodes 1, 2, 3, 4, and 5, respectively.

4.3- Register Placement

In the method proposed in [8], which was outlined in Section 4.1, a register placement step is needed in order to preserve the behavior of the original circuit. The placement of registers is derived from a schedule graph G_s, by breaking every path in G_s that is longer than the optimal clock period P. For paths having a length less than P, no register is required because operations chaining is assumed.

For the circuit in Figure 4.1, applying the algorithm in [8] for register placement on the schedule graph G_s in Figure 4.2, starting from v_1, gives the placement of registers and their schedules as depicted by Figure 4.3. The number of registers that are placed is 8 and the number of phases is 3.

The algorithm for register placement in [8] is not optimal. Indeed, for Figure 4.3, register R_1 can be omitted.

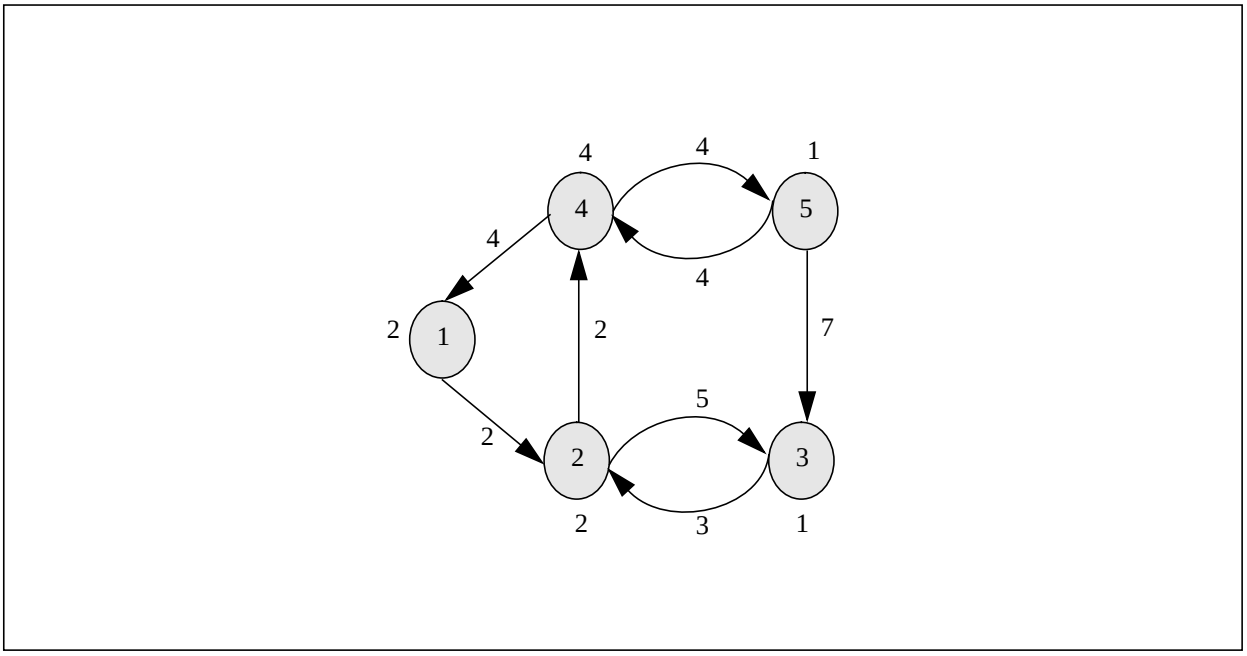
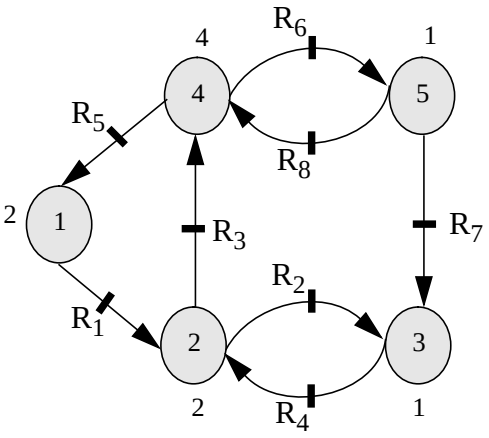


Figure 4.2 : Schedule graph.



Register	Schedule
R_1	2
R_2	3
R_3	0
R_4	2
R_5	0
R_6	0
R_7	3
R_8	0

Figure 4.3 : Registers' placement an their schedules.

4.4- Problem Formulation and Optimal Solution

As mentioned in Section 4.1, two problems arise in the method based on software pipelining proposed in [8]: the first one is how to determine a schedule that yields the minimum number of required registers, and the second one is how to place the minimum number of required registers even if that schedule is already determined. Our focus in this paper is to simultaneously solve these two problems. More precisely, the problem (*Prob*) we tackle is to determine a schedule with the minimum register requirements, where the register placement is done during the schedule determination. To optimally solve *Prob*, we provide a mixed integer linear program (MILP), and use it to derive a linear program which is polynomial-time solvable.

Before presenting that MILP, let us first give some requirements. Figure 4.4 gives a portion of the cyclic graph modeling the circuit, where *i* and *j* are two computational elements. $x_{i,j}$ denotes the number of registers that must be placed on the arc $e_{i,j}$ to guarantee that the length, $l_{i,j}$, of every path that goes to *j* via *i*, is less than or equal to the optimal clock period *P*. $l_{i,j}$ will be defined in the following. Note that as in [8], operation chaining is assumed, and hence no register is required if $l_{i,j} \leq P$. Suppose that paths that go to *j* via *i* are already examined in order to determine if some registers must be placed on them or not. Let m_i be a no-negative real greater than or equal to each remainder obtained by dividing the length of each one of those paths by *P*. The length $l_{i,j}$ of every path that goes to *j* via *i* is the sum of m_i and $w_s(e_{i,j})$, where $w_s(e_{i,j})$ is defined by equation (7). $y_{i,j}$ is the remainder of the division of $l_{i,j}$ by *P*. We require that $m_i \leq (P - d(i))$, which guarantee that if a register *R* is putted after the computational element *i*, then *R* will be scheduled after *i* finishes computing.

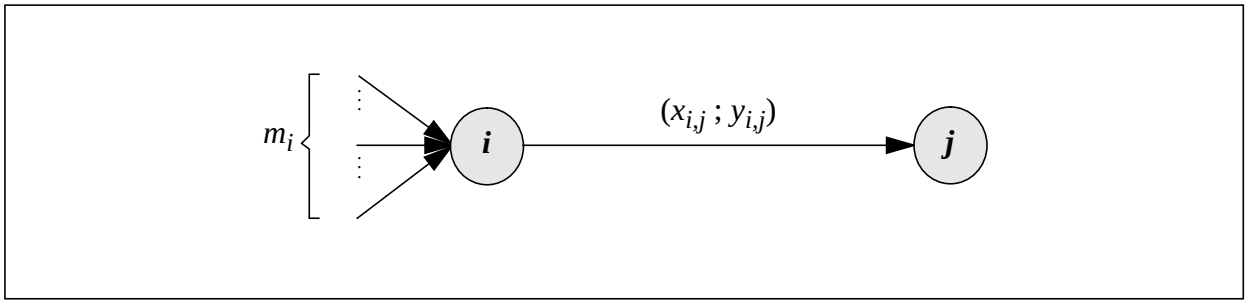


Figure 4.4 : Illustration of the variables of the MILP.

Figure 4.5 presents a mathematical formulation to *Prob*. The objective function expresses the number of registers to be placed in the circuit. Equations (8), (9), and (10) are equivalent to the

definition of $x_{i,j}$, $y_{i,j}$, and m_i, respectively. Inequality (11) is equivalent to (5). (13) is required, since the number of registers must be an integer. In this formulation, the variables are $x_{i,j}$, $y_{i,j}$, m_i and the schedule $s_0(u)$ for each computational element u.

The formulation in Figure 4.5 can be linearized as follows. Using the fact that $\lfloor x \rfloor \le x < \lfloor x \rfloor + 1$, and that no register is required if the length of a path is less than or equal to P, equation (8) can be replaced by:

$$\forall e_{i,j} \in E,\ x_{i,j} \le (s_0(j) - s_0(i) + P \cdot w(e_{i,j}) + m_i)/P \le x_{i,j} + 1. \tag{14}$$

Equations (9) and (10) together can be replaced by:

$$\forall e_{k,i} \in E,\ m_i \ge (s_0(i) - s_0(k) + P \cdot w(e_{k,i}) + m_k) - P \cdot x_{k,i}. \tag{15}$$

After linearizing the formulation in Figure 4.5, we obtain the MILP to optimally solve *Prob* as presented in Figure 4.6. In this figure, equations (16) and (17) are equivalent to (14). (18) is equivalent to (15). (19), (20) and (21) are equivalent to (12), (11) and (13), respectively.

$$Minimize\left(\sum_{\forall e_{i,j} \in E} x_{i,j}\right)$$

Subject to:

$$\forall e_{i,j} \in E,\ x_{i,j} = \lfloor (s_0(j) - s_0(i) + P \cdot w(e_{i,j}) + m_i)/P \rfloor \tag{8}$$

$$\forall e_{i,j} \in E,\ y_{i,j} = (s_0(j) - s_0(i) + P \cdot w(e_{i,j}) + m_i) - P \cdot x_{i,j} \tag{9}$$

$$\forall i \in V,\ m_i \ge (y_{k,i})_{k \in Fanin(i)} \tag{10}$$

$$\forall e_{i,j} \in E,\ s_0(j) - s_0(i) \ge d(i) - P \cdot w(e_{i,j}) \tag{11}$$

$$\forall i \in V,\ m_i \le P - d(i) \tag{12}$$

$$\forall e_{i,j} \in E,\ x_{i,j} \text{ is integer} \tag{13}$$

Figure 4.5 : A mathematical formulation to solve *Prob*.

4.5- A Linear Program for Solving *Prob*

Linear programs are polynomial-time solvable [59, 62]. A linear program for solving *Prob* can be obtained by deleting the constraint that $x_{i,j}$ is integer in Figure 4.6. In this case, once the linear program is solved, the number of registers to be placed on the arc $e_{i,j}$ is $\lceil x_{i,j} \rceil$. Details on why it is possible to place $\lceil x_{i,j} \rceil$ registers on the arc $e_{i,j}$ can be found in [10].

$$Minimize \left(\sum_{\forall e_{i,j} \in E} x_{i,j} \right)$$

Subject to:

$$\forall e_{i,j} \in E, -P \cdot x_{i,j} - s_0(i) + s_0(j) + m_i \geq -P \cdot w(e_{i,j}) \quad (16)$$

$$\forall e_{i,j} \in E, P \cdot x_{i,j} + s_0(i) - s_0(j) - m_i \geq P \cdot w(e_{i,j}) - P \quad (17)$$

$$\forall e_{k,i} \in E, P \cdot x_{k,i} + s_0(k) - s_0(i) + m_i - m_k \geq P \cdot w(e_{k,i}) \quad (18)$$

$$\forall i \in V, m_i \leq P - d(i) \quad (19)$$

$$\forall e_{i,j} \in E, s_0(j) - s_0(i) \geq d(i) - P \cdot w(e_{i,j}) \quad (20)$$

$$\forall e_{i,j} \in E, x_{i,j} \text{ is integer} \quad (21)$$

Figure 4.6 : A MILP to optimally solve *Prob*.

4.6- A Minimum Cost Network Flow Formulation

In this section, we focus on transforming the dual of the linear program (LP) in Section 4.5 to a formulation of a minimum cost network flow problem, since this latter can be solved more efficiently [1, 45, 66].

By replacing $P \cdot x_{i,j}$ with $X_{i,j}$. The LP can be written as in Figure 4.7.

For each i in V, let $\delta^+(i)$ be the set of immediate successors of i and $\delta^-(i)$ be its set of immediate predecessors. The dual of the linear program in Figure 4.7 can be written as in Figure 4.8. In this figure, the variables are: $A_{i,j}$'s, $B_{i,j}$'s, $C_{i,j}$'s, $D_{i,j}$'s, and $E_{i,j}$'s.

$$Minimize\left(\sum_{\forall e_{i,j} \in E} X_{i,j}\right)$$

Subject to:

$$\forall e_{i,j} \in E, -X_{i,j} - s_0(i) + s_0(j) + m_i \geq -P \cdot w(e_{i,j}) \quad (22)$$

$$\forall e_{i,j} \in E, X_{i,j} + s_0(i) - s_0(j) - m_i \geq P \cdot w(e_{i,j}) - P \quad (23)$$

$$\forall e_{k,i} \in E, X_{k,i} + s_0(k) - s_0(i) + m_i - m_k \geq P \cdot w(e_{k,i}) \quad (24)$$

$$\forall i \in V, -m_i \geq -(P - d(i)) \quad (25)$$

$$\forall e_{i,j} \in E, s_0(j) - s_0(i) \geq d(i) - P \cdot w(e_{i,j}) \quad (26)$$

Figure 4.7 : A linear program to solve *Prob*.

$$Maximize \left(\begin{array}{l} \sum_{\forall e_{i,j} \in E} -P \cdot w(e_{i,j}) \cdot A_{i,j} \\ + \sum_{\forall e_{i,j} \in E} (P \cdot w(e_{i,j}) - P) \cdot B_{i,j} \\ + \sum_{\forall e_{i,j} \in E} P \cdot w(e_{i,j}) \cdot C_{i,j} \\ + \sum_{\forall e_{i,j} \in E} (d(i) - P \cdot w(e_{i,j})) \cdot D_{i,j} \\ + \sum_{\forall e_{i,j} \in E} - (P - d(i)) \cdot E_{i,j} \end{array} \right) \quad (27)$$

Subject to:

$$\forall e_{i,j} \in E, -A_{i,j} + B_{i,j} + C_{i,j} = 1 \quad (28)$$

$$\forall i \in V, \begin{array}{l} \sum_{j \in \delta^+(i)} (-A_{i,j} + B_{i,j} + C_{i,j} - D_{i,j}) \\ - \sum_{j \in \delta^-(i)} (-A_{j,i} + B_{j,i} + C_{j,i} - D_{j,i}) = 0 \end{array} \quad (29)$$

$$\forall i \in V, \left(\sum_{j \in \delta^+(i)} (A_{i,j} - B_{i,j} - C_{i,j}) \right) + \left(\sum_{j \in \delta^-(i)} C_{j,i} + E_{j,i} \right) = 0 \quad (30)$$

$$\forall e_{i,j} \in E, A_{i,j} \geq 0, B_{i,j} \geq 0, C_{i,j} \geq 0, D_{i,j} \geq 0, E_{i,j} \geq 0 \quad (31)$$

Figure 4.8 : The dual of the formulation in Figure 4.7.

Equation (29) can be written as follows:

$$\forall i \in V, \sum_{j \in \delta^+(i)} D_{i,j} - \sum_{j \in \delta^-(i)} D_{j,i} = \sum_{j \in \delta^+(i)} (-A_{i,j} + B_{i,j} + C_{i,j}) - \sum_{j \in \delta^-(i)} (-A_{j,i} + B_{j,i} + C_{j,i}) \cdot \tag{33}$$

For each i in V, let $f_i = |\delta^+(i)| - |\delta^-(i)|$. Using equations (28) and (33), equation (29) can be replaced by:

$$\forall i \in V, \sum_{j \in \delta^+(i)} D_{i,j} - \sum_{j \in \delta^-(i)} D_{j,i} = |\delta^+(i)| - |\delta^-(i)| = f_i. \tag{34}$$

Using equation (28), equation (30) can be transformed to:

$$\forall i \in V, \sum_{j \in \delta^-(i)} C_{j,i} + E_{j,i} = \sum_{j \in \delta^+(i)} (-A_{i,j} + B_{i,j} + C_{i,j}) = |\delta^+(i)|. \tag{35}$$

Using equation (28), we can transform formula (27) to:

$$Maximize \left(\begin{array}{l} \sum_{\forall e_{i,j} \in E} - (P - d(i)) \cdot E_{i,j} - (P \cdot B_{i,j}) \\ + \sum_{\forall e_{i,j} \in E} (d(i) - P \cdot w(e_{i,j})) \cdot D_{i,j} \end{array} \right). \tag{36}$$

Formula (36) can be rewritten as follows:

$$Minimize \left(\begin{array}{l} \sum_{\forall e_{i,j} \in E} (P - d(i)) \cdot E_{i,j} + P \cdot B_{i,j} \\ + \sum_{\forall e_{i,j} \in E} (P \cdot w(e_{i,j}) - d(i)) \cdot D_{i,j} \end{array} \right). \tag{37}$$

Now, let us put all the above modifications together. Using (37) instead of (27), (34) instead of (29), and (35) instead of (30), we obtain the formulation in Figure 4.9.

$$Minimize \left(\sum_{\forall e_{i,j} \in E} (P - d(i)) \cdot E_{i,j} + P \cdot B_{i,j} + \sum_{\forall e_{i,j} \in E} (P \cdot w(e_{i,j}) - d(i)) \cdot D_{i,j} \right) \quad (39)$$

Subject to:

$$\forall e_{i,j} \in E, -A_{i,j} + B_{i,j} + C_{i,j} = 1 \quad (40)$$

$$\forall i \in V, \sum_{j \in \delta^+(i)} D_{i,j} - \sum_{j \in \delta^-(i)} D_{j,i} = f_i \quad (41)$$

$$\forall i \in V, \sum_{j \in \delta^-(i)} C_{j,i} + E_{j,i} = \left|\delta^+(i)\right| \quad (42)$$

$$\forall e_{i,j} \in E, A_{i,j} \geq 0, B_{i,j} \geq 0, C_{i,j} \geq 0, D_{i,j} \geq 0, E_{i,j} \geq 0 \quad (43)$$

Figure 4.9 : Formulation after modifying Figure 4.8.

Since, $\forall e_{i,j} \in E, B_{i,j} \geq 0$, $E_{i,j} \geq 0$ and considering that constraints on $D_{i,j}$'s are independent of constraints on $B_{i,j}$'s and on $E_{i,j}$'s, then we can always select all $B_{i,j} = 0$ and $E_{i,j} = 0$. Hence, formula (39) can be reduced to:

$$Minimize \left(\sum_{\forall e_{i,j} \in E} (P \cdot w(e_{i,j}) - d(i)) \cdot D_{i,j} \right). \quad (44)$$

Due to the fact that the variables $A_{i,j}$'s, $B_{i,j}$'s, $C_{i,j}$'s and $E_{i,j}$'s do not appear in formula (44), then they can be removed from the constraints. The final form of the transformed dual of the linear program in Figure 4.7 is presented in Figure 4.10, which is a formulation of a minimum cost network flow problem, and hence it can be solved more efficiently using, for instance, algorithms provided in [1, 45, 66].

$$Minimize \left(\sum_{\forall e_{i,j} \in E} (P \cdot w(e_{i,j}) - d(i)) \cdot D_{i,j} \right)$$

Subject to:

$$\forall i \in V, \sum_{j \in \delta^{+}(i)} D_{i,j} - \sum_{j \in \delta^{-}(i)} D_{j,i} = f_i \quad (45)$$

$$\forall e_{i,j} \in E, D_{i,j} \geq 0 \quad (46)$$

Figure 4.10 : The transformed dual of Figure 4.7.

4.7- Multi-Phase Clocked Circuits with Clock Skew

Clock skew generally limits the performance of synchronous systems. However, intentional clock skew can be introduced to improve the performance of clocked circuits. The authors of [9] have established a relation between the tolerable skew Δ, the optimal clock period P at which the circuit operates, and the maximal value of the distance $Dist$ between the schedule of any two adjacent registers. They showed that, when minimum delays are considered as zero:

$$\Delta = (P - Dist)/4, \quad (47)$$

and that the maximum tolerable skew can be obtained when $Dist = Dmax$, where $Dmax$ is the delay of the slowest computational element in the circuit.

To produce a multi-phase clocked circuit operating at the optimal clock period and having a given tolerance to clock skew Δ, the authors of [9] used the method in [8], which is outlined in Section 4.1. Also, during register placement, outlined in Section 4.3, paths are broken whenever their lengths become greater than $Dist$ instead of when they are greater than P. As reported in [9], the authors do not have a method to reduce the number of registers that must be placed in the circuit, and they do not know how to exploit non-critical paths to reduce the number of required registers. The main limitations come from the use of ASAP or ALAP schedules, and from the lack

of a powerful algorithm for register placement. In other words, they faced problems (1) and (2) exposed in Section 4.1.

The approach we proposed in the previous sections can be used to overcome the obstacles presented above. Indeed, from equation (47), we have that:

$$Dist = P - 4 \cdot \Delta, \tag{48}$$

for a given tolerance to clock skew Δ. By using equation (48), the mathematical formulation in Figure 4.5 transforms to the one presented in Figure 4.11.

$$Minimize \left(\sum_{\forall e_{i,j} \in E} x_{i,j} \right)$$

Subject to:

$$\forall e_{i,j} \in E,\ x_{i,j} = \left\lfloor (s_0(j) - s_0(i) + P \cdot w(e_{i,j}) + m_i) / (P - 4 \cdot \Delta) \right\rfloor \tag{49}$$

$$\forall e_{i,j} \in E,\ y_{i,j} = (s_0(j) - s_0(i) + P \cdot w(e_{i,j}) + m_i) - (P - 4 \cdot \Delta) \cdot x_{i,j} \tag{50}$$

$$\forall i \in V,\ m_i \geq (y_{k,i})_{k \in Fanin(i)} \tag{51}$$

$$\forall i \in V,\ m_i \leq Dist - d(i) \tag{52}$$

$$\forall e_{i,j} \in E,\ s_0(j) - s_0(i) \geq d(i) - P \cdot w(e_{i,j}) \tag{53}$$

$$\forall e_{i,j} \in E,\ x_{i,j} \text{ is integer} \tag{54}$$

Figure 4.11 : A mathematical formulation to solve *Prob* in the case of no-zero clock skew.

A mixed integer linear program ($MILP_{skew}$) and a linear program (LP_{skew}) can be derived from the formulation in Figure 4.11 as we did for obtaining Figures 4.6 and 4.7.

From inequality (49), we can deduce the following theorem.

Theorem 4.1 : *For a clocked circuit operating at the optimal clock period P, a tolerance to clock skew $\Delta = P/4$ requires an infinite number of registers.*

Proof: From inequality (5), we have that:

$$\forall e_{i,j} \in E,\ s_0(j) - s_0(i) + P \cdot w(e_{i,j}) \geq d(i) > 0\,. \tag{55}$$

By definition, we have that:

$$\forall i \in V,\ m_i \geq 0\,. \tag{56}$$

By (55) and (56), we can deduce that:

$$\forall e_{i,j} \in E,\ s_0(j) - s_0(i) + P \cdot w(e_{i,j}) + m_i > 0\,. \tag{57}$$

By (58), the numerator of (49) is always positive. Hence, when Δ tends to $P/4$, we have that $P - 4 \cdot \Delta$ tends to 0. Consequently, $x_{i,j}$ tends to the infinity when Δ tends to $P/4$. In this case, the number of required registers is the infinity.

4.8- Experimental Results

To test the effectiveness of our approach, the MILP in Figure 4.6 and the corresponding linear program (LP), obtained by ignoring the constraint *integer* in (21), are experimented on well known benchmarks. Circuits from the ISCAS89 benchmark suite are used to test the efficiency of the LP in terms of the run-time and of the reduction of the number of registers inserted in the circuit. The mathematical formulations for each circuit are automatically generated by a module we coded in C++ and integrated in a tool we developed in [18]. We did not implement the cited polynomial-time algorithms for linear programs, but the Lp_Solve tool [74] (in the public domain) is used to solve the generated mathematical formulations. Obtained results are given in Tables 4.1 and 4.2, where the first column gives the name of the circuit and the second column presents the number, N_1, of registers placed using the algorithm in [8] that uses ALAP as a schedule. The number, N_2, of registers placed by MILP or by LP are presented in the third column. The fourth column gives the relative gain defined as $((N_1 - N_2)/N_1) \times 100\%$. For Table 4.2, the fifth column gives the run-time in seconds on an UltraSparc 10 with 1GB RAM. As Table 4.1 reports, significant reductions of the number of required registers are obtained. Substantial reductions are also obtained using the LP. Indeed, as summarized by Table 4.2, reductions as high as 77.46% are obtained in less than 181.4s.

To test the efficiency of the transformed dual of the LP, we used the two formulations to solve a few benchmarks as given in Table 4.3. In this table, we present the run time (RT_1) of the LP, and the run time (RT_2) of its transformed dual, in the second and third columns respectively. The fourth

column presents the speedup defined as RT_1/RT_2. As Table 4.3 reports, speedup factors ranging from 1 to 16.47 have been achieved, although we have not used specialized algorithms for minimum cost network flow problems to solve the transformed dual.

Table 4.1 : Register placement by [8] vs. by MILP.

	#registers placed by [8]	**#registers placed using Figure 4.6**	**Relative gain**
Figure 4.1	8	7	12.5%
SOIIR Filter [2]	3	2	23.33%
Polynomial Div. [2]	9	4	55.55
Correlator [8]	6	6	0%
FOWDEF [63]	14	8	42.85%

Table 4.2 : Register placement by [8] vs. by LP.

	#registers placed by [8]	**#registers placed using the LP**	**Relative gain**	**Run-time (s)**
Figure 4.1	8	8	0%	0.01
SOIIR Filter [2]	3	3	0%	0.02
Polynomial Div. [2]	9	4	55.55%	0.02
Correlator [8]	6	6	0%	0.01
FOWDEF [63]	14	12	14.28%	0.06
S344	131	53	59.54%	1.63
S641	142	32	77.46	1.25
S1423	422	216	48.81%	17.31
S5378	1033	581	43.75%	181.4
S9234	1042	466	55.27%	93.29

Table 4.3 : Run times of the LP and Figure 4.10.

	Run-time of the LP (s)	**Run-time using Figure 4.10 (s)**	**Speed up**
Figure 4.1	0.01	0.01	1
SOIIR Filter [2]	0.02	0.01	2
Polynomial Div. [2]	0.02	0.01	2
Correlator [8]	0.01	0.01	1
FOWDEF [63]	0.06	0.01	6
S344	1.63	0.11	14.81
S641	1.25	0.19	6.57
S1423	17.31	1.55	11.16
S5378	181.4	11.01	16.47
S9234	93.29	6.4	14.57

The efficiency of the approach with clock skew is also tested. Indeed, we experimented the linear program LP_{skew} obtained as presented in Section 4.7, and compared results with the method proposed in [9]. As a target skew, we used $\Delta = (P - Dist)/4$, where P is the minimal clock period, $Dist = Max(P/5, Max_{v \in V}(d(v)))$, and $d(v)$ is the delay of the computational element v. Recall that, as stated in [9], $Dist = Max_{v \in V}(d(v))$ gives the maximum tolerance to clock skew. We used $P/5$ since if $Dist = P/5$ we have that $\Delta = P/5$, which is close to $P/4$ that leads to an infinite number of registers based on Theorem 4.1. Table 4.4 reports obtained results using some circuits from the ISCAS89 benchmark suite. The first column gives the name of the used circuits. Columns 2 presents the number, Nb_1, of registers placed by the method in [9] using ALAP as a schedule. The number, Nb_2, of registers placed by LP_{skew} is given in column 3. The fourth column gives the relative gain defined as $((Nb_1 - Nb_2)/Nb_1) \times 100\%$. The execution time for solving LP_{skew} is presented in the fifth column. As summarized by Table 4.4, reductions of the number of required registers as high as 57.68% are obtained. Execution times for solving LP_{skew} range from 1.19 s to 163.08 s.

Table 4.4 : Register placement with clock skew: results by [9] vs. by LP_{skew}.

	#registers placed by [9]	**#registers placed using the LP_{skew}**	**Relative gain**	**Run-time (s)**
S344	456	221	51.53%	1.19
S641	487	388	20.32%	1.35
S1423	2252	953	57.68%	17.24
S5378	1449	1373	5.24%	163.08
S9234	3167	1431	54.81	82.28

4.9- Conclusions

A method based on software pipelining has recently been proposed to optimize mono-phase clocked sequential circuit. The resulting circuit is a multi-phase clocked circuit, where all clocks have the same period. To preserve the behavior of the original circuit, registers are placed according to a schedule, which has the maximum throughput.

In that method, two problems arise: how to determine schedules that lead to a minimal register requirements, and how to place the minimum number of required registers even if these schedules are already determined.

In this paper, we have simultaneously tackled these two problems. We have provided a mixed integer linear program and used it to derive a linear program, which is polynomial-time solvable. Also, we have transformed the dual of this linear program to a formulation of minimum cost network flow problem, which can be solved more efficiently. Furthermore, we have presented how the proposed approach can handle clock skew. Experimental results on well known benchmarks confirmed the effectiveness of the approach. Indeed, significant reductions of the number of required registers have been obtained in very short run-time. The run time to solve the transformed dual of the linear program was very small compared to the one of the primal.

Chapitre 5

Réduction de la consommation de puissance pour des circuits synchrones

Le besoin de réduire la consommation de puissance pour des circuits numériques est motivé principalement par le besoin de prolonger la durée de vie des batteries pour les systèmes mobiles et portables, ainsi que par le besoin de réduire l'échauffement pour les systèmes à haute performance. Une réduction de puissance permet de réduire la taille des batteries ou de prolonger l'autonomie d'un système. D'autre part, la chaleur engendrée par la consommation de puissance, peut endommager le matériel, ralentir sa vitesse de traitement, et même modifier la fonctionnalité du système.

Dans ce chapitre, nous nous concentrons sur la réduction de la consommation de la puissance dynamique de circuits synchrones en technologie CMOS et qui fonctionnent à leur vitesse maximale. La puissance dynamique est une fonction quadratique de la tension d'alimentation. La vitesse de traitement d'un élément de calcul décroît quand sa tension d'alimentation décroît. Afin de réduire la consommation de la puissance dynamique de circuits synchrones fonctionnant à vitesse maximale, déterminée en supposant que tous les éléments de calcul sont alimentés par des tensions les plus élevées possible, nous proposons l'utilisation de plusieurs tensions d'alimentation. Les éléments de calcul sur les chemins critiques sont à alimenter par des tensions les plus hautes possible. Des tensions les plus basses possible sont à assigner aux autres éléments. Le problème de minimiser la consommation de la puissance en utilisant plusieurs tensions d'alimentation est un problème NP-complet en général. Nous formulons mathématiquement ce problème et nous proposons deux méthodes de résolution. La première méthode donne la solution optimale et est basée sur une technique de type "branch-and-bound". La

deuxième méthode est une heuristique formulée sous forme d'un programme linéaire. Nous testons l'efficacité de notre approche sur un ensemble de cas de test. Les résultats expérimentaux montrent que (1) des facteurs de réduction de la consommation de puissance de l'ordre de 53,84% peuvent être obtenus et que (2) dû aux règles proposées pour l'élimination des solutions inutiles, la méthode exacte est capable de produire la solution optimale en un petit nombre d'essais dans un espace de solutions très grand.

Nous poursuivons ce chapitre par la présentation de l'article [19].

Determining Schedules for Reducing Power Consumption Using Multiple Supply Voltages

Noureddine Chabini[1], El Mostapha Aboulhamid[1], Yvon Savaria[2]

1: *LASSO, DIRO, Université de Montréal C.P.6128, Suc. Centre ville, Montréal, Qc, Canada, H3C 3J7. Email:{chabinin, aboulham}@iro.umontreal.ca*

2: *GRM, DGEGI, École Polytechnique de Montréal, C.P. 6079, Suc. Centre-ville, Montréal, Qc, Canada, H3C 3A7. Email: savaria@vlsi.polymtl.ca*

Abstract

Dynamic power is the main source of power consumption in CMOS circuits. It depends on the square of the supply voltage. It may significantly be reduced by scaling down the supply voltage of some computational elements in the circuit, with the penalty of an increase of their execution delay. To reduce the dynamic power consumption, without degrading the performance determined assuming that the circuit operates at the highest available supply voltage, the supply voltage of computational elements off critical paths can be scaled down. Defined here as $MinP_{dyn}$, the problem of minimizing the dynamic power consumption, under performance constraints, by scaling down the supply voltage of computational elements on non-critical paths is NP-hard in general. Solving $MinP_{dyn}$ for multi-phase clocked sequential circuits may allow to reduce their power consumption and the required number of registers. Reducing the number of registers also allows to reduce the power consumption, the number of control signals, and the area of the circuit. In this paper, we focus on devising methods to efficiently solve $MinP_{dyn}$ for designs modeled as cyclic or acyclic graphs. More precisely, once the circuit is optimized for timing constraints, then we look for schedules that allow the computational elements of the circuit to operate at the lowest possible supply voltage. We present an integer linear program for that problem, which we use to devise a polynomial time solvable method and an exact algorithm based on a branch-and-bound technique. Experimental results confirm the effectiveness of the method and power reduction factors as high as 53,84% were obtained. Also, they show that the exact algorithm produces optimal results in a small number of tries, which is due to the rules used to prune useless solutions.

5.1- Introduction

Design for low power has several motivations, such as prolonging battery life in wearable electronic devices, and reducing the cooling cost for high performance systems. Battery life is becoming a product differentiator in many portable electronic markets [42]. High performance systems are characterized by large power dissipation. This transforms to heat that can lead to system malfunction or that may force reducing system performance.

Dynamic power, P_{dyn}, is the main source of power dissipation in CMOS circuits. P_{dyn} depends on the square of the supply voltage, V_{dd}, as expressed by the following equation [69, 42]:

$$P_{dyn} = KC_{lc} f V_{dd}^2, \quad (1)$$

where K is the switching activity factor, C_{lc} is the loading capacitance, and f is the clock frequency.

Due to the quadratic term in equation (1), dynamic power may significantly be reduced by scaling down the supply voltage. Supply voltage reductions increase execution delays. To avoid decreasing performance, we can use performance determined assuming that the circuit operates at the highest acceptable supply voltage as a target. Then, the supply voltage of computational elements on non-critical paths can be reduced. Defined in this paper as $MinP_{dyn}$, the problem of minimizing the dynamic power consumption, under performance constraints, by scaling down the supply voltage of computational elements on non-critical paths is NP-hard in general [25].

Methods to obtain solutions to $MinP_{dyn}$ have been proposed. In [94], a polynomial-time algorithm has been proposed for performance-constrained non-pipelined designs. Given a predetermined set of supply voltages, the authors [94] schedule the acyclic datapath and assign voltages to the computational elements in order to minimize power. In their approach, it was assumed that the voltage versus delay curve is identical for all computational elements in the circuit. Under this assumption, the problem transforms to a problem where identical computational elements are used. Without this assumption, there is no guarantee that the algorithm produces optimal results.

A technique to reduce power consumption has been proposed in [110], where only two supply voltages are allowed. A depth-first search is used to determine computational elements

which may operate at low supply voltage, without violating the timing constraints of the circuit. In this method, it is only after examining all its successors that a computational element can be allowed to operate at low supply voltage. Nevertheless, significant reductions of power consumption can be obtained by selecting a computational element without first examining all its successors. Also, it is a question of how to choose the node to start the depth-first search in cyclic designs.

To optimally solve $MinP_{dyn}$ under resource constraints, an integer linear program approach for non-pipelined acyclic designs has been provided in [69]. The authors [69] also present a heuristic to solve the problem, but general designs, such as pipelined and/or cyclic datapaths, are not examined.

A dynamic programming technique to solve $MinP_{dyn}$ is proposed in [25]. The authors [25] reported that their method can produce optimal results for designs modeled as tree-like data flow graphs. Suboptimal results are obtained in the general case, such as pipelined design.

Methods based on software pipelining techniques have been recently proposed to derive multi-phase clocked sequential circuits operating at the optimal throughput [18, 8]. Solving $MinP_{dyn}$ for this class of circuits may allow to reduce the power consumption, and to reduce the number of registers in the circuit. Although operation chaining is assumed to reduce the required number of registers and the number of control signals in this class of circuits, further reductions may be obtained by a supply voltage scaling approach. Indeed, registers connected to computational elements on non-critical paths may be omitted, since the execution delay of these elements, after applying supply voltage scaling techniques, may be increased, which may make operation chaining possible. Control signals for these registers will then be saved, and the circuit area will be reduced too.

The approach we propose here is general and can be used at any level of the design hierarchy, but we focus in this paper on solving $MinP_{dyn}$ for the class of circuits mentioned above. These circuits contain feedbacks and can be modeled as cyclic graphs for optimization purposes. To the best of our knowledge, no work has been explicitly proposed yet for solving $MinP_{dyn}$, when the design is modeled as a cyclic graph. In this paper, we present an integer linear programming formulation to $MinP_{dyn}$ that we use to devise polynomial-time solvable method and an exact algorithm based on the branch-and-bound technique. Experimental results confirm the

effectiveness of the method and power reduction factors as high as 53,84% were obtained. Also, they show that the exact algorithm produces optimal results in a small number of tries, which is due to the rules used to prune useless solutions.

The rest of the paper is organized as follows. In Section 2, we introduce the notations and definitions used in this work. Section 3 presents the formulation of the problem we tackle. We present a polynomial time solvable method to determine solutions to that problem in section 4, and an exact algorithm based on a branch-and-bound technique in section 5. Section 6 provides experimental results and Section 7 concludes the paper.

5.2- Preliminaries

5.2.1- The Cyclic Graph Model

A synchronous sequential circuit is modeled (as in [18, 8]) as a directed cyclic graph $G = (V, E, d, w)$, where V is the set of functional elements in the circuit, and E is the set of edges which represent interconnections between vertices. Each vertex v in V has a non-negative integer execution delay $d(v) \in N$. Each edge $e_{u, v}$, from u to v, in E is weighted with a register count $w(e_{u, v}) \in N$, representing the number of registers on the wire between u and v.

Figure 5.1 presents an example of a circuit and its directed cyclic graph model. In this figure, large rectangles represent functional elements, and small rectangles represent registers. Wires are oriented to show the propagation direction of the signals. The execution delay of each functional element of this circuit is specified as a label on the left of each large rectangle. This example will be used through this paper, and will serve as an example of initial specification for the problem to optimize. The initial specification is in general a synchronous circuit with a single-phase clock period, and it is assumed to operate at the highest acceptable supply voltage. As an example, the clock period of the circuit in Figure 5.1 is 6, which is equal to $d(v_4) + d(v_1)$.

5.2.2- Periodic Schedules

We define a schedule s [2] as a function $s : N \times V \rightarrow Q$, where $s_n(v) \equiv s(n, v)$ denotes the schedule time of the n^{th} iteration of operation v. In multi-phase flip-flop based circuits, the schedule time is the start time of the operation. A schedule s is called periodic with period P, if:

$$\forall n \in N, \forall v \in V: s_{n+1}(v) = s_n(v) + P. \quad (2)$$

When there is no resource constraint, a schedule s is said to be valid if and only if the

operations terminate before their results are needed. In this case, we say that data dependencies are satisfied, which is equivalent to the following mathematical inequality:

$$\forall n \in N,\ \forall e_{u,v} \in E : s_{n + w(e_{u,v})}(v) \geq s_n(u) + d(u). \tag{3}$$

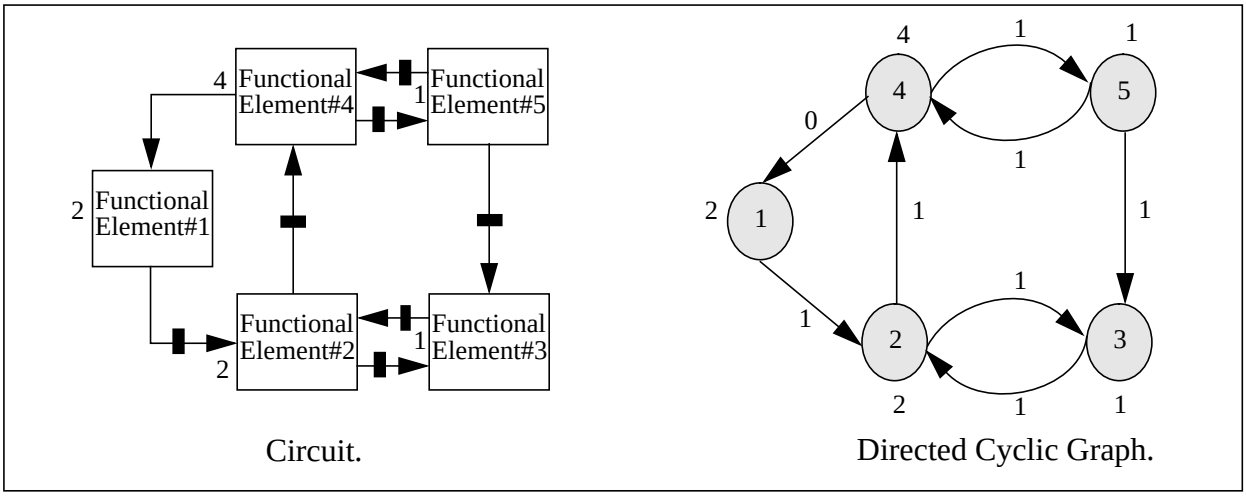


Figure 5.1 : Sample circuit and its directed cyclic graph model.

5.2.3- The Maximum Throughput of Synchronous Sequential Circuits

The throughput, *T*, of a synchronous sequential circuit is bounded by the inverse of the length, *P*, of the critical paths in the circuit. Based on data dependencies constraints only, the maximum throughput is [2]:

$$T = Min_{c \in C}\left(\left(\sum_{e_{u,v} \in c} w(e_{u,v})\right) \Big/ \left(\sum_{\forall v \in V \text{ and } e_{u,v} \in c} d(u)\right)\right), \tag{4}$$

where *C* is the set of directed cycles in the directed cyclic graph modeling the circuit. Determining the maximum throughput is a Minimal Cost-to-Time Ratio Cycle Problem [17, 43, 66], which can be solved in the general case in $O(|V| \cdot |E| \cdot \log(|V| \cdot d_{max}))$ [17, 66], where $d_{max} = Max_{v \in V}(d(v))$. A possible method to solve this problem is to iteratively apply Bellman-Ford's algorithm for longest paths on the graph $G_P = (V, E, d, w_P)$ derived from *G* by letting:

$$w_P(e_{u,v}) = d(u) - P \cdot w(e_{u,v}), \tag{5}$$

where $e_{u,v} \in E$ and $P = 1/T$. A binary search may be used to find the minimal value of *P* for which there is no positive cycle in G_P [66].

For the example in Figure 5.1, we have that $P = 4$. This value corresponds to the cycle

defined by vertices v_1, v_2, and v_4.

5.2.4- Schedule for a Given Throughput

From equation (2) and inequality (3), we have that:

$$\forall e_{u,v} \in E,\ s_0(v) - s_0(u) \geq d(u) - P \cdot w(e_{u,v}). \tag{6}$$

In the case of periodic schedules, determining a valid schedule of all the instances of each vertex v in V is equivalent to determining $s_0(v)$ for each v in V, which is also equivalent to determining solutions to the system of inequalities described by (6). To solve this system, the graph G_P, previously described, may be used. To find an ASAP schedule, Bellman-Ford's algorithm for longest paths, from a chosen vertex v_x to the others, may be applied on the graph G_P. Finding an ALAP schedule may be done as follows: step 1, a graph G' has to be derived from G_P by inverting the direction of each edge in G_P; step 2, Bellman-Ford's algorithm for longest paths, from the vertex v_x to the others, has to be applied on the graph G', where the weights of its edges are defined by equation (5); finally, step 3, the ALAP schedule is obtained by multiplying each result in step 2 by -1. Relatively to $v_x = v_1$, the ASAP schedules of vertices v_1, v_2, v_3, and v_4 of the circuit in Figure 5.1 are 0, -2, -4, -4 and -4, respectively. The ALAP schedules of these vertices are 0, -2, 1, -4 and -1, respectively.

5.2.5- Schedule Graph

As in [8], a periodic schedule, with period P, of vertices of directed cyclic graph modeling a circuit is presented by a schedule graph $G_s = (V, E, d, w_s, P)$. Here V, E and d have the same definition given for the case of the graph G previously defined, and $w_s : E \rightarrow Q$ is a weight function, which associates to each edge $e_{u,v}$ in E the time distance between the schedule times of u and v. Mathematically, $w_s(e_{u,v})$ is defined as follows:

$$\forall e_{u,v} \in E,\ w_s(e_{u,v}) = s_{w(e_{u,v})}(v) - s_0(u). \tag{7}$$

Because s is periodic with period P, equation (7) may be written as follows:

$$\forall e_{u,v} \in E,\ w_s(e_{u,v}) = s_0(v) - s_0(u) + P \cdot w(e_{u,v}). \tag{8}$$

The graph G_s is consistent if and only if for each edge $e_{u,v}$ in E, $w_s(e_{u,v}) \geq d(u)$. This is derived from (6) and (8). Figure 5.2 shows a consistent schedule graph, where edges are labeled with w_s values, for the circuit in Figure 5.1 using the ALAP schedule presented in Section 5.2.4.

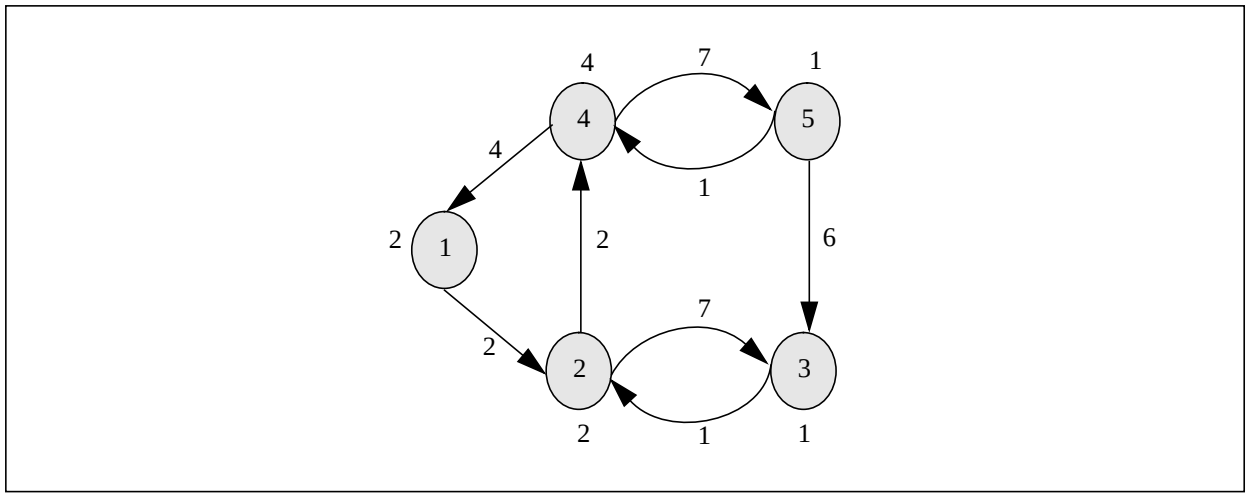
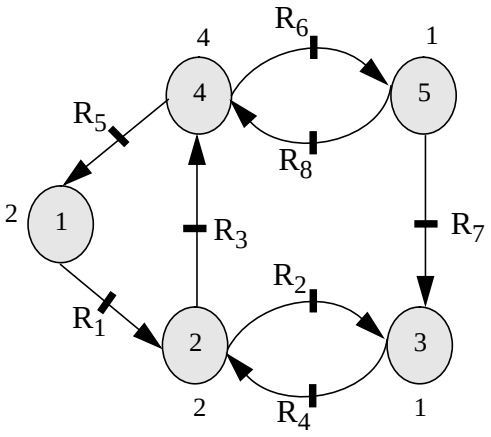


Figure 5.2 : Schedule graph.

5.2.6- Register Placement

In the method based on software pipelining proposed in [8], once the optimal clock period is determined and a schedule time of all the computational elements is computed, then a register placement step is needed in order to preserve the behavior of the original circuit. The placement of registers is derived from a schedule graph G_s, by breaking every path in G_s that is longer than the optimal clock period P. For paths having a length less than P, no register is required because operations chaining is assumed.

For the circuit in Figure 5.1, applying the algorithm in [8] for register placement on the schedule graph G_s in Figure 5.2, starting from v_1, gives the placement of registers and their schedules, as presented in Figure 5.3. The number of registers is 8 and the number of phases is 4.



Register	Schedule
R_1	2
R_2	1
R_3	0
R_4	2
R_5	0
R_6	3
R_7	1
R_8	0

Figure 5.3 : Register placement and their schedules.

5.3- Problem Formulation

Let P_h be the power consumed by the circuit, where each computational element operates at the highest supply voltage. The problem $MinP_{dyn}$ we tackle in this paper is defined as follows. Given that acceptable multiple different supply voltages are available, and that only one can be used to drive a computational element, then determine the schedule that leads to the maximum reduction of power consumption, relatively to P_h, under the requirements R_1 and R_2 defined in the following. For R_1, we are locking for a solution where each computational element operates at the lowest possible supply voltage. For R_2, the schedule has the same latency and throughput as when the circuit operates at the highest supply voltages only. Here, latency is defined as the time required to execute the first instance of all the operations.

Before presenting a mathematical formulation for that problem, let us first define some requirements. Based on the supply voltages used, we assume that we have n_i different implementations for each computational element i. If supply voltage V_{dd}^k, where $1 \le k \le n_i$, is used, then the computational element i has an execution delay $d(i) = d_{i,k}$ and consumes the power $p_{i,k}$. We suppose that supply voltages are sorted from the highest value to the smallest one, and numbered starting from 1. For each $i \in V$, and for each k such that $1 \le k \le n_i$, let us denote by $x_{i,k}$ an integer, which is equal to 1 if supply voltage k is used and to 0 otherwise. $MinP_{dyn}$ can be

formalized as a mixed integer linear programming problem, $MILPMinP_{dyn}$, as presented in Figure 5.4. In this formulation, the objective function expresses the power consumed by all the computational elements. The variables are $x_{i,k}$'s and the schedule time $s_0(u)$ for each $u \in V$. P is the optimal clock period, which is determined as mentioned in Section 5.2.3 assuming that the circuit operates at the highest supply voltages. The other parameters are as defined in Section 5.2. Equation (9) ensures that only one of all the available implementations of computational element i will be used. Equation (10) is equivalent to (6), where the execution delay of i is $(\sum_{k=1}^{n_i} x_{i,k} d_{i,k})$. Equation (11) is used to have a schedule that has a latency less than or equal to that of the case where the circuit operates at the highest supply voltages only. Equation (12) and (13) are used to prune the solution space and to guarantee that the schedule will have the optimal throughput, where ASAP and ALAP schedules are computed assuming that the circuit operates at the highest supply voltages. Equation (14) is equivalent to the definition of $x_{i,k}$'s.

Formulation F_1:

$$Minimize \sum_{\forall i \in V} \sum_{k=1}^{n_i} x_{i,k} p_{i,k}$$

Subject to:

$$\forall i \in V, \sum_{k=1}^{n_i} x_{i,k} = 1 \quad (9)$$

$$\forall e_{i,j} \in E, s_0(j) - s_0(i) \geq (\sum_{k=1}^{n_i} x_{i,k} d_{i,k}) - P \cdot w(e_{i,j}) \quad (10)$$

$$\forall i \in V, s_0(i) + \sum_{k=1}^{n_i} x_{i,k} d_{i,k} \leq ALAP_i + d_{i,1} \quad (11)$$

$$\forall i \in V, s_0(i) \geq ASAP_i \quad (12)$$

$$\forall i \in V, s_0(i) \leq ALAP_i \quad (13)$$

$$\forall i \in V, k = 1, ..., n_i : x_{i,k} \in \{0, 1\} \quad (14)$$

Figure 5.4 : A Mixed integer linear programming formulation to $MinP_{dyn}$.

Due to the space limitation, proofs of the following theorems are omitted.

Theorem 5.1 : *The problem in Figure 5.4 has always a solution that is not necessarily optimal.*

Theorem 5.2 : *The optimal solution to the problem in Figure 5.4 has the maximum throughput, P.*

Since no efficient method is reported yet, in the literature, for solving large and general mixed integer linear programming problems, then devising heuristics to solve them is of great interest. In the subsequent section, we will present how we can transform the $MILPMinP_{dyn}$ to obtain efficient methods for solving $MinP_{dyn}$.

5.4- A Linear Programming Formulation to Determine a Solution to $MinP_{dyn}$

Scaling down the supply voltage of a computational element reduces its power consumption, but increases its execution delay. Based on this fact, replacing

$$Minimize \sum_{\forall i \in V} \sum_{k=1}^{n_i} x_{i,k} p_{i,k} \tag{16}$$

in the formulation in Figure 5.4, by

$$Maximize \sum_{\forall i \in V} \sum_{k=1}^{n_i} x_{i,k} d_{i,k}, \tag{17}$$

may produce significant power saving over the power consumed when the circuit operates at the highest supply voltages. Let $NMILPMinP_{dyn}$ be the resulting formulation.

Let

$$d(i) = \sum_{k=1}^{n_i} x_{i,k} d_{i,k}. \tag{18}$$

A linear programming formulation, $LPMinP_{dyn}$, can be derived from $NMILPMinP_{dyn}$ as presented in Figure 5.5. In this formulation, (19) is derived using (17) and (18). Inequality (20) is obtained using (10) and (18). Inequality (21) is determined using (11) and (18). Inequalities (22) and (23) are equivalent to (12) and (13), respectively. Inequalities (24) and (25) are equivalent to the fact that the execution delay of any implementation of the computational element *i* is between $d_{i,1}$ and d_{i,n_i}, as defined in Section 5.3.

Once the problem in Figure 5.5 is solved, the supply voltage under which the computational element *i* must operate is V_{dd}^m, where *m* is an integer such that $d_{i,m} \le d(i) < d_{i,m+1}$. The schedule of computational elements may have to be recomputed using, for instance, methods provided in [18].

Formulation F_2:

$$Maximize \sum_{\forall i \in V} d(i) \quad (19)$$

Subject to:

$$\forall e_{i,j} \in E,\ d(i) + s_0(i) - s_0(j) \leq P \cdot w(e_{i,j}) \quad (20)$$

$$\forall i \in V,\ s_0(i) + d_i \leq ALAP_i + d_{i,1} \quad (21)$$

$$\forall i \in V,\ -s_0(i) \leq -ASAP_i \quad (22)$$

$$\forall i \in V,\ s_0(i) \leq ALAP_i \quad (23)$$

$$\forall i \in V,\ -d(i) \leq -d_{i,1} \quad (24)$$

$$\forall i \in V,\ d(i) \leq d_{i,n_i} \quad (25)$$

Figure 5.5 : A linear programming formulation to determine a solution to $MinP_{dyn}$.

Due to the space limitation, proofs of the following theorems are omitted.

Theorem 5.3 : *The problem in Figure 5.5 has always a solution.*

Theorem 5.4 : *The optimal solution to the problem in Figure 5.5 has the maximum throughput, P.*

Theorem 5.5 : *The problem in Figure 5.5 can be solved in polynomial time.*

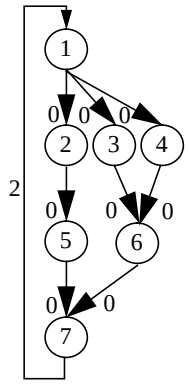
In the case where the computational elements exhibit significant differences in the power reduction to delay increase tradeoff, the objective function in Figure 5.5 may not provide the best power reduction. Let us first give an example and then show how we can improve the objective function to deal with such situation. For the example in Figure 5.6, the critical path is formed by nodes 1, 2, 5 and 7. The paths 1, 3, 6, 7, and 1, 4, 6, 7 are not critical. Indeed, the maximum throughput is 1/4 and the ASAP schedules for nodes 1, 2, 3, 4, 5, 6 and 7 are 0, 2, 2, 2, 4, 3 and 6, respectively. Their ALAP schedules are 0, 2, 3, 3, 4, 4 and 6, respectively. Critical nodes have the ASAP schedules equal to their ALAP schedules. Now, to scale down the supply voltage of the nodes on non-critical paths, we have two possibilities: scaling down the supply voltage of nodes 3 and 4 or of node 6. If the first case is chosen, the power consumed will be reduced by 20 units; whereas in the second case, it will be reduced by 50 units. However, by examining the average power consumed per unit of time, c_i, for nodes 3, 4, and 6, we remark that for nodes 3 and 4, we

have that $c_4 = c_3 = (p_{3,1} + p_{3,2})/(d_{3,1} + d_{3,2}) = 10$, and for node 6, we have that $c_6 = (p_{6,1} + p_{6,2})/(d_{6,1} + d_{6,2}) = 30$. Hence, injecting these coefficients into the objective function would favor slowing node 6 over nodes 3 and 4. Based on that fact, formula (19) can be replaced by:

$$Maximize \sum_{\forall i \in V} c_i \cdot d(i), \tag{26}$$

where c_i's are defined as follows:

$$c_i = (\sum_{k=1}^{n_i} p_{i,k})/(\sum_{k=1}^{n_i} d_{i,k}). \tag{27}$$



	V^1_{dd}		V^2_{dd}	
Node#	$d_{i,1}$	$p_{i,1}$	$d_{i,2}$	$p_{i,2}$
1	2	100	3	50
2	2	100	3	50
3	1	20	2	10
4	1	20	2	10
5	2	100	3	50
6	2	100	3	50
7	2	100	3	50

Figure 5.6 : Example to explain how Figure 5.5 can be improved.

5.5- An Exact Algorithm to Solve $MinP_{dyn}$

To solve optimally $MinP_{dyn}$, we provide an exact algorithm, *ExactAlgo*, based on a branch-and-bound technique as presented in Figure 5.8. Let $config_G$ be the set of supply voltages at which the computational elements of *G* operate. Initially, $config_G$ holds the highest supply voltages only. *ExactAlgo* has as input a graph *G*, an integer *i* that controls the termination of the algorithm, an upper bound, *UpperBound*, on the power consumed by the circuit, and the optimal clock period, *P*, of the circuit. Initially, the circuit operates at the highest supply voltages, which allows to determine *UpperBound*, *P* and ASAP and ALAP schedules for computational elements. These

schedules are saved as *ASAP_save* and *ALAP_save*. An example of tree to be explored by *ExactAlgo* is presented in Figure 5.7. *ExactAlgo* has to be called with $i = 1$. At each time *ExcatAlgo* is called, one and only one computational element is switched to the next low supply voltage, which leads to the creation of a new node in the searching tree. In order to be accepted, the configuration of the graph at that node must satisfy the following conditions: (1) the graph is still working at the maximum throughput *P*, (2) ASAP and ALAP schedules are still bounded by the values of *ASAP_save* and *ALAP_save*, and (3) each computational element *i* terminates its execution no later than $d_{i,1} + ALAP\_\text{save}_i$. If one of these constraints is not satisfied, then the computational element is switched back to the last admissible supply voltage; in this case, the process of trying another supply voltage for *i* is stopped, and we then examine the next computational element of the node where we are; if all the computational elements of that node are examined, then *ExactAlgo* backtracks or it terminates if it passed all the computational elements of the first node at which the algorithm started to execute. If a node is accepted, then a new node is created in the searching tree in order to examine the next computational element. This new node has as parent the computational element whose supply voltage was just updated at the accepted node. In *ExactAlgo*, an early check of the acceptance of switching the supply voltage to the next one is done by verifying if $d_{i,k} \leq d_{i,1} + ALAP\_\text{save}_i - ASAP\_\text{save}_i$, which is derived from inequality (11) assuming that $s_0(i) = ASAP\_\text{save}_i$.

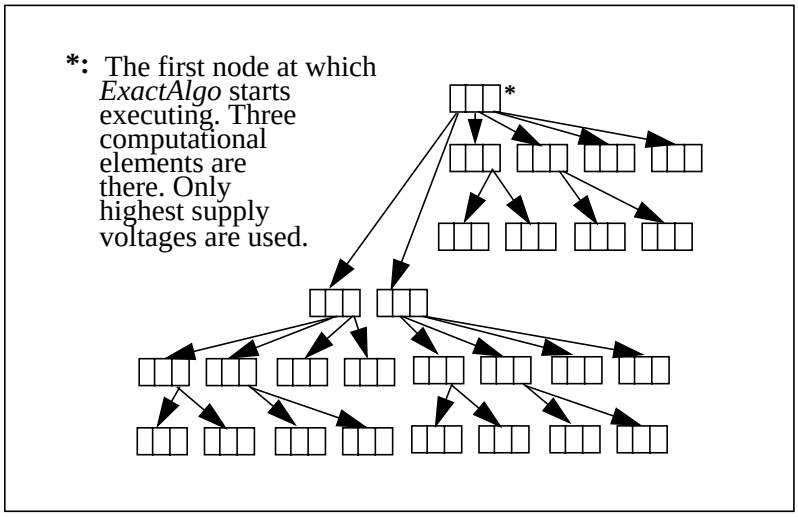


Figure 5.7 : Example of implicit tree to be explored by *ExactAlgo*. Three supply voltages are assumed.

```
Function ExactAlgo (Graph: G = (V, E, d, w), integer i, float UpperBound, rational P) {
  if (i > |V|) then return;
  else {
    integer j;
    for (j=i; j<= |V|; j++) {
      for (k=2; k<=NumberOfVdd; k++) {
        if (d_{i,k} >d_{i,1} + (ALAP_save_i - ASAP_save_i))
        then { k = NumberOfVdd + 1; break;}
        else {
          save_d_i = d_i;
          d_i = d_{i,k};
          determine the optimal period, P_opt, for the updated graph G;
          determine the ASAP and ALAP schedules for the updated graph G;
          if (P_opt > P or
           ∃x ∈ V : [ASAP_x,ALAP_x] ⊄ [ASAP_save_x,ALAP_save_x] or
           ∃x ∈ V : (ALAP_x + d_x > ALAP_save_x + d_{x, 1})
          ) then
          {k = NumberOfVdd + 1; break;}
          else {
            determine the power, power, consumed by the updated graph G;
            ExactAlgo (G = (V, E, d, w), i+1, UpperBound, P);
            if (power < UpperBound) then {
              UpperBound = power;
              save the present configuration of G into config_G ;
              // config_G initialy holds the highest supply voltages
            }
          }
          d_i = save_d_i;
        }
      }
    }
  }
}
```

Figure 5.8 : An algorithm based on a branch-and-bound technique.

5.6- Experimental Results

To test the effectiveness of the approach we propose, we experimented the linear programming formulation in Figure 5.5 and the exact algorithm in Figure 5.8 on some benchmarks used in the literature and on Figures 5.1 and 5.6. Circuits from the ISCAS89 benchmark suite have been used to test the efficiency, in terms of reducing power consumption and the run-time, of the linear programming formulation. We use four supply voltages: 5V, 3.3V, 2.4V, and 1.5V. To determine the delay $d_{i,k}$ and the power consumed $p_{i,k}$ for each computational element *i*, when the supply voltage V_{dd}^{k} is used, we proceed as follows. We use the rule [110]

$$d_{i,k} = ({V^{k}}_{dd}/({V^{k}}_{dd} - V_{th})^{2}) \cdot (({V^{1}}_{dd} - V_{th})^{2}/{V^{1}}_{dd}) \cdot d_{i,1},$$

where V_{th} is the threshold voltage. V_{dd}^{1} is assumed known and V_{th} is fixed at 0.7V, which is a typical value used in the literature. $p_{i,k}$'s are determined assuming that the fanout of the computational element *i* has the same loading capacitance. By using equation (1), we have that:

$$p_{i,k} = KC_0 f.(Fanout_i \cdot ({V^{k}}_{dd})^{2}). \qquad (28)$$

For Figure 5.5 with formula (26), we use the LP_Solve tool [74] to solve the linear program for each circuit. The linear program is automatically generated by a module we coded in C++ and integrated in the tool we developed in [18]. Results are summarized in Table 5.1, where the first column gives the circuit name, the second column gives the power consumed (divided by KC_0f) in the case of using the highest supply voltage only (P_h) and in the case of using multiple supply voltages (P_m), the third column presents the power saving factor defined as $((P_h - P_m)/P_h) \times 100$, and the last column gives the run-time in second.

ExactAlgo is coded in C++ and integrated in the tool we developed in [18]. Table 5.2 reports the obtained results. Column 1, 2 and 3 are the same as for Table 5.1. Column 4 presents the number of tries done divided by the size, k^n, of the searching space, where *k* is the number of the different available supply voltages and *n* is the number of the computational elements in the circuit.

As Table 5.1 reports, significant reductions of power consumption are obtained in a short

run-time using the linear programming formulation. Indeed, saving factors ranging from 5.3% to 53.84% are obtained in less than 166.81s using an UltraSparc 10 with 1GB RAM. As summarized by Table 5.2, optimal power consumptions using multiple supply voltages are obtained in a small number of tries in a very large searching space. For instance, for the circuit FOWDEF, we have that $n = 34$ and $k = 4$; nevertheless, only 22399 possibilities are examined (in 44.01 s) to find the schedule with the minimal power consumption.

5.7- Conclusions

The problem, $MinP_{dyn}$, of minimizing the power consumption, under performance constraints, by supply voltage scaling is NP-hard in general. To the best of our knowledge, no work in the literature has explicitly tackled this problem in the case of cyclic designs.

Software pipelining has recently been used to derive multi-phase clocked sequential circuits operating at the maximum throughput. To preserve the functionality of the original circuit, registers are placed after the optimal clock period P and a schedule of the computational elements of the circuit are determined. During registers placement, only paths that have a length greater than P are broken since operation chaining is assumed.

Solving $MinP_{dyn}$ for the class of multi-phase clocked circuits is very interesting, since it may allow to reduce the number of registers and hence to save the power they consume, to reduce the number of the control signals, and to reduce the area of the circuit.

We have provided a mixed integer linear formulation to $MinP_{dyn}$, and used it to devise a polynomial time solvable method and an exact algorithm based on a branch-and-bound technique. Experimental results confirmed the effectiveness of the method and showed that power reduction factors as high as 53.84% can be obtained in short run-times. They also confirmed the efficiency of the exact algorithm. Indeed, optimal results are obtained in a small number of tries in a very large searching space.

Table 5.1 : Results using Figure 5.5.

	(Power Consumed)/(KC_0f)		**Gain = (Column#2-Column#3)/ Column#2**	**Run-time (sec)**
	Using the highest supply voltages	**Using multiple supply voltages**		
Figure 5.1	205	176	14.14%	0.01
Figure 5.6	232	202	12.93%	0.01
SOIIR Filter [2]	129	114	11.62%	0.01
Polynomial Div. [2]	309	251	18.77%	0.01
Correlator [8]	283	268	5.3%	0.01
FOWDEF [63]	1306	1181	9.57%	0.01
S344	6225	4146	33.39%	0.2
S641	8457	5980	29.28%	0.17
S731	9758	6549	32.88%	0.22
S5378	64847	47234	27.16%	14.37
S9243	46905	22712	51.57%	12.38
S1238	25017	17896	28.46%	1.81
S1423	24533	11323	53.84%	1.87
S1488	33360	28144	15.63%	3.64
S1494	33868	28431	16.05%	5.93
S13207	116500	62255	46.56%	79.76
S15850	139403	69383	50.22	166.81

Table 5.2 : Results using *ExactAlgo*.

	(Power Consumed)/(KC_0f)		**Gain = (Column#2-Column#3)/ Column#2**	**(Number of tries)/(The size of the searching space)**
	Using only the highest supply voltages	**Using multiple supply voltages**		
Figure 5.1	205	176	14.14%	9/1024
Figure 5.6	232	202	12.93%	20/16384
SOIIR Filter [2]	129	114	11.62%	8/256
Polynomial Div. [2]	309	209	32.36%	741/262144
Correlator [8]	283	268	5.3%	13/65536
FOWDEF [63]	1306	1144	12.4%	$22399/(4^{34})$

Chapitre 6

Accélération d'algorithmes de résolution du problème rapport extrême coût/temps

Le problème rapport extrême coût/temps consiste à trouver la valeur optimale de ce rapport parmi ses valeurs possibles définies par les cycles dans un graphe. Comme nous allons présenter dans ce chapitre, ce problème est important en pratique. Les algorithmes les plus recommandés pour la résolution de ce problème requièrent une borne sur la solution optimale pour amorcer leur exécution. La détermination d'une bonne borne permet de réduire significativement le temps d'exécution de ces algorithmes, surtout pour le cas de graphes de grande taille. Une telle réduction est d'une grande importance, surtout quand ces algorithmes sont à invoquer à plusieurs reprises. Dans ce chapitre, nous présentons une méthode d'une faible complexité temporelle pour déterminer une borne sur la solution optimale de ce problème. Cette méthode est basée sur des algorithmes de détermination d'arbres de recouvrement optimal et son efficacité a été testée sur un ensemble de cas de test.

Nous poursuivons ce chapitre par la présentation de l'article [17].

A Fast Method for Determining an Efficient Bound on the Optimal Solution of the Cost-to-Time Ratio Problem

Noureddine Chabini
LASSO, DIRO, Université de Montréal C.P.6128, Suc. Centre ville, Montréal, Qc, Canada, H3C 3J7. Email: chabinin@iro.umontreal.ca
and
El Mostapha Aboulhamid
LASSO, DIRO, Université de Montréal C.P.6128, Suc. Centre ville, Montréal, Qc, Canada, H3C 3J7. Email: aboulham@iro.umontreal.ca
and
Yvon Savaria
GRM, DGEGI, École Polytechnique de Montréal, C.P. 6079, Suc. Centre-ville, Montréal, Qc, Canada, H3C 3A7. Email: savaria@vlsi.polymtl.ca

Abstract

The cost-to-time ratio problem has numerous applications. For instance, algorithms for solving this problem have been used to compute optimal clock period of synchronous circuits and the iteration period bound of cyclic data flow graphs. Some excellent algorithms for solving this problem require a bound to start their execution. Choosing a good bound may significantly decrease their execution time. This paper presents a fast method for determining an efficient bound on the optimal solution of this problem. The method has been implemented and tested on ISCAS89 benchmark circuits. By using the obtained bounds, experimental results show that the best algorithms we have implemented for solving this problem have been substantially accelerated.

Keywords: Cost-to-Time Ratio, Graph, CAD, Optimal Clock Period, Bound, Spanning Tree.

6.1- Introduction

Let us first provide some notations and definitions. Let $G = (V, E)$ be a directed graph with n nodes and m arcs. Associate to each arc e in E a cost $w(e)$ and a transit time $t(e)$. Denote by $W(C)$ and $T(C)$ the weight and the transit time of a cycle C in G. $W(C)$ and $T(C)$ are, respectively, equal

to the sum of the weights and to the sum of transit times of the arcs on the cycle C.

The cost-to-time ratio, $q(C)$, of a cycle C is defined as:

$$q(C) = \frac{W(C)}{T(C)}, \; T(C) > 0 . \tag{1}$$

The cost-to-time ratio problem is to find the cycle C in G for which $q^* = q(C)$ is minimal or maximal. That is,

$$q^* = Min_{\forall C \in G}(q(C)),$$

or

$$q^* = Max_{\forall C \in G}(q(C)).$$

When all arcs in the graph have a transit time equal to unity, the cost-to-time ratio of a cycle C is called the cycle mean, which we denote here by $\lambda(C)$. It is defined as:

$$\lambda(C) = \frac{W(C)}{T(C)} = \frac{W(C)}{|C|}, \tag{2}$$

where $|C|$ denotes the number of arcs in the cycle C. In this case, the cost-to-time ratio problem is called the mean cycle problem. It corresponds to finding the cycle C in G for which $\lambda^* = \lambda(C)$ is minimal or maximal. That is,

$$\lambda^* = Min_{\forall C \in G}(\lambda(C)),$$

or

$$\lambda^* = Max_{\forall C \in G}(\lambda(C)).$$

Numerous applications of the cost-to-time ratio problem are found in Computer Aided Design (CAD), graph theory, discrete event system theory and manufacturing systems [34, 32, 33]. For instance, algorithms for solving this problem have been used to compute optimal clock period of synchronous circuits [11] and the iteration period bound of cyclic data flow graphs [43, 55, 67, 111]. These algorithms have also been used for a ship routing problem [31] and to the cyclic stuffing problem [47, 48]. Other applications of the cost-to-time ratio problem may be found in [44, 53].

As we will discuss in the next section, some of the best algorithms for solving the cost-to-time ratio problem require a bound on the optimal solution to start their execution. Choosing a good bound may significantly speedup the convergence of these algorithms. Decreasing their execution time is very important, since, these algorithms may have to be executed many times.

This paper presents a fast method for determining an efficient bound on the optimal solution of the cost-to-time ratio problem. Since the mean cycle problem is just a particular case of the former, our method is applicable to both. The method has been implemented and tested on graphs corresponding to the *ISCAS89* benchmark circuits. As it will be discussed in the section of experimental results, by using the obtained bounds, some of the best algorithms that we have implemented have been substantially accelerated.

The rest of this paper is organized as follows. The next section presents some algorithms proposed for solving the cost-to-time ratio problem. Section 3 presents our method for determining a bound on the optimal solution for this problem. Experimental results are presented in section 4, and section 5 summarizes our conclusions, ongoing work, and suggestions for future work.

6.2- Proposed Methods for Solving the Cost-to-Time Ratio Problem

Many algorithms have been proposed for solving the cost-to-time ratio problem. Algorithms to find the minimum value of $q(C)$ are also applicable to find its maximum, and vice-versa; this is possible because passing from one problem to the other can be done by multiplying the cost of all arcs by -1. Hence, in the sequel, we will only discuss some algorithms proposed to find the minimum value of $q(C)$ as they are presented in their respective source papers. Also, since the mean cycle problem is just a particular case of the cost-to-time ratio problem, algorithms proposed for the latter are applicable to the former. It is also possible that algorithms for the former may be used to solve the latter [34, 32].

Algorithms for solving the minimum cost-to-time ratio problem have been proposed by Burns [12], Dantzig et *al*. [31], Dasdan and Gupta [33], Golischek [44], Hartmann [47], Karp [60], Karp and Orlin [61], Lawler [67, 66], and Megiddo [86]. See [34, 32] for other algorithms.

A classification of the fastest algorithms for the minimum cost-to-time and the minimum mean cycle problems has been done by Dasdan et *al*. [34, 32]. Also, these authors presented an empirical study comparing the ten best algorithms for the minimum mean cycle problem. Based

on this study, we will focus on the Burns algorithm [12, 34, 32], Karp-and-Orlin (KO)'s algorithm [34, 32, 61], Young-Tarjan-and-Orlin (YTO)'s algorithm [34, 32, 114], Lawler's algorithm [67, 66] and the Howard's algorithm [29, 34, 32].

Let G_λ be a weighted directed graph derived from the graph G, where each arc $a_{i,j}$ in G_λ has a weight equal to $w(e) - \lambda$ for each arc e in G, such that $e = a_{i,j}$. Shortest paths in G_λ are well defined if and only if there is no negative cycle in G_λ. The minimum cycle mean λ^* is the largest λ such that the graph G_λ does not have any negative cycle. Some algorithms for determining λ^*, such as KO's algorithm and YTO's algorithm, fit into the class of parametric shortest path algorithms. These algorithms start with $\lambda = -\infty$ and change it incrementally until a cycle of weight zero is detected. When a cycle of weight zero is detected in G_λ, that cycle is the one with the minimum mean.

Burn's algorithm [12, 34, 32] behaves like parametric shortest path algorithms. It is based on a linear programming technique.

Lawler's algorithm [67, 66] does a binary search over the possible values of λ^* and checks for a negative cycle in G_λ every iteration. If one is found, the chosen λ must be decreased because it is too large; otherwise it is increased because it is too small. To do this search, lower and upper bounds, L and H, on λ^* must first be determined. This algorithm terminates when the size of the interval for the possible values of λ^* becomes too small. The performance of Lawler's algorithm depends on H and L.

Howard's algorithm [29, 34, 32] behaves like a parametric shortest path algorithm. It starts with a large value of λ and decreases it until the shortest paths in G_λ are well defined. For a given λ, if no negative cycle has been detected in G_λ, then the correct λ has been found. Else, λ has to be updated to the mean of the negative cycle since its mean is smaller then λ. Howard's algorithm has been ranked first in the empirical study done in [34, 32].

As discussed, KO's algorithm, YTO's algorithm, Lawler's algorithm and Howard's algorithm require a bound on the optimal solution to start their execution. Hence, their convergence may be accelerated by choosing a good bound on this solution.

Little work has been done for determining bounds on the optimal solution of the cost-to-time ratio problem. Gerez et *al*. [43] have presented a $O(Tm + T^4)$ algorithm based on the longest path matrices and their multiplications, where T is the sum of the transit times in the input graph

and *m* is its number of arcs. An algorithm to find bounds on the separation time of events in an arbitrary process graph without conditional behavior has been presented by Hulgaard et *al*. [53].

In the next section, we present a fast method for determining an efficient bound on the optimal solution of the cost-to-time ratio problem.

6.3- A Method for Determining a Bound on the Optimal Solution of the Cost-to-Time Ratio Problem

Considering the equivalence between finding the minimum and maximum cost-to-time ratio (one can be derived from the other by multiplying the cost of all arcs by -1), we will focus on methods that search for a minimum. Thus, in the rest of the paper, we look for a cycle C in G for which $q^* = q(C)$ is minimal, where $q(C)$ is defined by equation (1).

By examining Eq. (1), $q(C)$ may be minimized if

$$W(C) = \sum_{e \in C} w(e) \tag{3}$$

is minimized. The minimization of $W(C)$ in Eq. (3) may be regarded as a minimum spanning tree problem. Hence, an upper bound on the optimal solution of the minimum cost-to-time ratio problem of a graph G may be obtained as follows. First, a minimum spanning tree, Tr, of G has to be determined. Second, arcs not in this tree must be examined to detect some cycles among which the cycle that minimize Eq. (1) may exist. Each arc must be examined exactly one time. If the arc being examined generates a cycle C in the tree Tr, then $q(C)$ must be calculated and compared to a temporary value q_t that is initially fixed to infinity or to the best known bound on the value of q^*. If $q(C)$ is less than q_t, then q_t is updated to $q(C)$. This process continues until all arcs not in Tr have been examined. The final value of q_t is an upper bound on q^*. Figure 6.1 presents the corresponding algorithm. An example to illustrate its execution is given in Figure 6.2. Figure 6.2 (a) presents an input graph G. Figure 6.2 (b) gives a minimum spanning tree, Tr, of G. The set of arcs not in Tr are: $a_{3,1}$, $a_{4,1}$, $a_{4,5}$ and $a_{5,1}$. The arc $a_{4,5}$ does not generate a cycle in Tr. Cycles generated by the arcs $a_{3,1}$, $a_{4,1}$and $a_{5,1}$ have $q(C)$ equal to 1/3, 1/4 and 1/8 respectively. Thus, the upper bound returned by the algorithm is 1/8, which is the optimal solution in this case.

Algorithm:

Input: A strongly connected digraph $G = (V, E)$ with n nodes and m arcs. Each arc e in E has a weight equal to $[w(e), t(e)]$.

Output: q_u, an upper bound on the solution, q^*, of the minimum cost-to-time ratio problem.

Begin:

1. Let $q_u = Max_{e \in E}\{w(e)\}$, if $t(e) \geq 1$ for all e in E. Else, q_u is to be fixed to an upper bound on q^*, for example the infinity.
2. Determine a minimum spanning tree, *Tr*, of G. To do so, only the weight $w(e)$ of each arc has to be considered.
3. Traverse *Tr* in preorder and in postorder. Let $prenum_N$ and $postnum_N$ the numbers associated to each node N in *Tr* during these traversals, respectively.
4. For each node N in *Tr*, determine $AccW_N$ and $AccT_N$ as follows:
   If N is the root of *Tr*, then $AccW_n = 0$ and $AccT_n = 0$.
   Else, $AccW_N = AccW_P + w(e)$ and $AccT_N = AccT_P + t(e)$, for each arc, in *Tr*, from P to N.
5. Let $\bar{E}$ be the set of arcs in G but not in *Tr*.
   While $\bar{E}$ is not empty do
   5.1 Delete an arc $e = a_{i,j}$ from $\bar{E}$.
   5.2 If $prenum_j \leq prenum_i$ and $postnum_j \geq postnum_i$ then $a_{i,j}$ generates a cycle C in *Tr*; in this case, go to 5.3. Else, go to 5.1.
   5.3 $q(C) = (AccW_i - AccW_j + w(e))/(AccT_i - AccT_j + t(e))$.
   5.4 If $q_u > q(C)$, then $q_u = q(C)$.

End.

Figure 6.1 : Algorithm to determine an upper bound on the optimal solution of the minimum cost-to-time ratio problem.

The complexity of this algorithm is $O(m\lg(n))$. Indeed, in Figure 6.1, step 1 takes a time in $O(m)$. Step 2 may be done in $O(m\lg(n))$ by using Kruskal's algorithm. Steps 3 and 4 take $\Theta(n)$. In step 5, we have at most $(m-(n-1))$ arcs to analyze. So, step 5 takes a time in $O(m-(n-1))$ since the time required for steps 5.1 and 5.4 is bounded by a constant.

The input of our algorithm is a strongly connected digraph *G*. If *G* is not strongly connected, then it must be partitioned into strongly connected components as discussed in [34] for the case of cost-to-time ratio algorithms. An upper bound for each component is then determined. The upper bound for *G* is the maximum bound among those obtained.

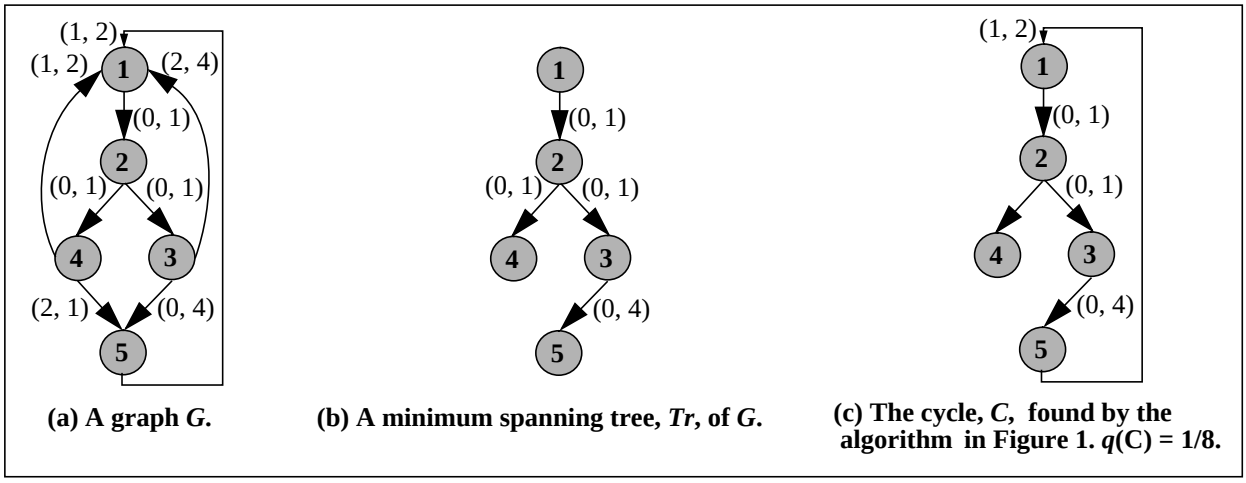


Figure 6.2 : Illustration of the execution of the algorithm in Figure 6.1.

6.4- Experimental Results

We have implemented the algorithm of Figure 6.1 in C++ and tested it on *ISCAS89* benchmark circuits. To test the effectiveness of the bounds computed by this algorithm, we have implemented two of the best algorithms for the minimum cost to time ratio problem: the algorithm described in [31], which is largely used in the CAD community [11, 111], and the Howard's algorithm [29, 34, 32] ranked first in the empirical study done in [34, 32]. We are also implementing other algorithms from the set of best algorithms for this problem to test the impact of the bound generated by our method on their convergence speed.

The implementation of the algorithm in [31] is tested for the case of the cost-to-time ratio problem, while Howard's algorithm is tested for the case of the mean cycle problem. Obtained results are summarized in Tables 6.1 and 6.2. The first column gives the name of the circuit. The second presents the execution time of our method. The third and fourth columns present the number of iterations and the execution time of each algorithm, which is implemented without considering the bound obtained by our method, and by considering it respectively; execution times

in the fourth columns include the execution time of our method. The difference between the number of iterations and the execution time of both columns 3 and 4 is given in column 5. Execution times are in milliseconds. Compared to the implementation which does not consider the proposed bound, results in Table 6.1 show that 1 to 2 iterations have been saved. For instance, in the case of s35932, 2 iterations have been saved and the execution time has been decreased from 1008650 ms to 137380 ms.

In the empirical study done in [34, 32], Howard's algorithm is the fastest, but as Table 6.2 shows, it can be accelerated. Indeed, 2 to 6 iterations have been saved by using the bound calculated by our method. For the case of s35932, 2 iterations have been saved and the execution time has been decreased from 163340 ms to 137380 ms.

In the fifth column of the tables 6.1 and 6.2, there is some cases where the difference between the execution times of columns 3 and 4 is negative. For these cases, no iteration has been saved, and the absolute value of each negative number corresponds to the execution time of our method, which is $O(m\lg(n))$.

Table 6.1 : Results with the algorithm described in [31].

	Our method	**Algorithm implemented without our method: impl#1**		**Algorithm implemented with our method: impl#2**		**Difference between the two implementations impl#1 and impl#2**	
	exec time in ms	**numb. of iterations**	**exec time in ms**	**numb. of iterations**	**exec time in ms**	**numb. of iterations**	**exec time in ms**
s344	5	4	50	3	35	1	15
s641	10	4	180	3	80	1	100
s713	5	4	210	3	135	1	75
s1238	15	3	1020	3	1035	0	-15
s5378	20	5	9830	5	9090	0	740
s13207	40	4	52060	3	24590	1	27470
s35932	220	4	1008650	2	25760	2	982890

Table 6.2 : Results with Howard's algorithm.

	Our method	**Algorithm implemented without our method: impl#1**		**Algorithm implemented with our method: impl#2**		**Difference between the two implementations impl#1 and impl#2**	
	exec time in ms	**numb. of iterations**	**exec time in ms**	**numb. of iterations**	**exec time in ms**	**numb. of iterations**	**exec time in ms**
s641	5	11	40	11	45	0	-5
s713	5	20	90	14	65	6	25
s838	10	6	240	4	220	2	20
s1238	5	20	260	20	265	0	-5
s5378	10	11	850	11	860	0	-10
s35932	200	10	163340	8	137380	2	25960

6.5- Conclusions and Future Works

Many problems in CAD, graph theory, discrete event system theory and manufacturing systems have been formulated as cost-to-time ratio problems. As we have reviewed, some of the best algorithms for this problem require a bound to start executing. By using a good bound on the optimal solution, these algorithms may be significantly accelerated. Decreasing their execution time is very important, since these algorithms may be executed many times.

This paper presented an $O(m\lg(n))$ algorithm for determining a bound on the optimal solution of the cost-to-time ratio problem of a graph with n nodes and m arcs. This algorithm has been implemented and interfaced with two of the best cost-to-time ratio algorithms: the algorithm described in [31] and Howard's algorithm [29, 34, 32]. Although this latter was found to be the fastest in the empirical study reported in [34, 32], results show that by using the bound obtained by our algorithm, it was possible to speed it up. The execution time of the former algorithm has also been decreased. We are implementing some of the other best algorithms to test the effectiveness of our method for accelerating them. Also, we plan to experiment the impact of the existence of several minimum (or maximum as desired) spanning trees in a graph on our algorithm.

Chapitre 7

Conclusion

Ce chapitre rappelle la problématique de cette thèse et résume ses contributions. Aussi, il présente des avenues de recherche.

7.1- Rappel de la problèmatique de la thèse et contributions

Dans cette thèse, nous avons traité des problèmes reliés à la conception en matériel de systèmes informatiques synchrones. Ces problèmes découlent du problème de l'amélioration de la performance sous contrainte de la réduction de la surface et/ou de la consommation de la puissance. Nous avons aussi abordé l'accélération des algorithmes de résolution du problème "Cost-to-Time Ratio".

La vitesse des circuits séquentiels synchrones dépend non seulement de la vitesse de leurs éléments de calcul mais aussi de la distance entre chaque paire de registres adjacents. Pour des circuits mono-phase, la technique de retiming est capable de minimiser cette distance en déplaçant les registres dans le circuit. Cependant, dans certains cas, cette distance peut être réduite davantage jusqu'à même atteindre la borne inférieure déterminée par la contrainte de dépendance de données. Pour ces cas, les circuits résultats sont des circuits multi-phases.

Une méthode basée sur des techniques de pipeline logiciel a été récemment proposée pour minimiser la période d'horloge de circuits séquentiels synchrones. Quand la période ainsi minimisée est différente de la période obtenue par la technique de retiming, le circuit résultat est un circuit séquentiel synchrone multi-phases. Comparée aux méthodes proposées dans la littérature pour le même problème, cette méthode est capable de produire des circuits séquentiels synchrones fonctionnant à vitesse maximale qui n'est limitée que par la contrainte de dépendance de données. La vitesse maximale est déterminée en résolvant une instance du problème "Cost-to-Time Ratio". Des registres sont à placer dans le circuit afin de conserver la fonctionnalité de sa version originale.

Le placement de registres requiert d'abord la détermination d'un ordonnancement des éléments de calcul du circuit. Une fois que les registres sont placés, on dérive les phases contrôlant le circuit final. Dans cette méthode, les problèmes suivants s'imposent:

- **P_1**: Comment déterminer des ordonnancements afin de réduire le nombre de registres.
- **P_2**: Comment déterminer des ordonnancements afin de réduire le nombre de phases.
- **P_3**: Comment placer le plus petit nombre de registres, même si un bon ordonnancement est déterminé.

Réduire le nombre de registres permet de réduire la surface du circuit et sa consommation de puissance, alors que réduire le nombre de phases permet de réduire la complexité des tâches de génération et de distribution de l'horloge.

Dans cette thèse, nous avons élaboré deux formulations mathématiques pour résoudre les problèmes P_1 et P_2 présentés ci-dessus. Nous avons présenté comment nous pouvons dériver des programmes linéaires à partir de ces formulations. Nous avons donné des procédures pour résoudre les programmes linéaires ainsi obtenus. Nous avons prouvé expérimentalement l'efficacité de notre approche.

Pour résoudre le problème P_3 présenté ci-dessus, nous avons développé une formulation mathématique qui permet de déterminer simultanément un ordonnancement des éléments de calcul du circuit à optimiser, ainsi que le placement des registres, tout en minimisant leur nombre total. À partir de cette formulation, nous avons dérivé un programme linéaire mixte ainsi qu'un programme linéaire facile à résoudre. Nous avons transformé ce programme linéaire en une formulation de flot à coût minimum, puisque ce dernier problème possède des algorithmes de résolution très efficaces. Nous avons démontré expérimentalement l'efficacité de notre approche.

Pour les circuits en technologie CMOS, la puissance dynamique est une des composantes la plus importante dans l'équation de la consommation de puissance d'un circuit. La puissance dynamique est une fonction quadratique de la tension d'alimentation. En conséquence, réduire la tension d'alimentation peut réduire de beaucoup la puissance consommée par le circuit.

Pour réduire la consommation de puissance pour des circuits en technologie CMOS, nous avons proposé une approche basée sur l'utilisation de plusieurs tensions d'alimentation. Pour une

période d'horloge donnée, qui peut être minimale, les tensions d'alimentation les plus hautes sont à affecter aux éléments de calcul qui sont sur les chemins critiques et des tensions les plus basses possible sont à assigner aux autres éléments. Nous avons formulé mathématiquement notre approche et nous avons proposé deux méthodes de résolution. La première méthode est basée sur une technique de type "branch-and-bound" et permet de déterminer la solution optimale. La deuxième méthode est une heuristique formulée sous forme d'un programme linéaire. Nous avons testé l'efficacité des deux méthodes sur un ensemble de cas de test. Notons qu'obtenir la consommation minimale de la puissance, en utilisant plusieurs tensions d'alimentation, est un problème prouvé NP-Complet en général.

Aussi, nous avons proposé une méthode efficace pour déterminer des bornes sur la solution optimale du problème "Cost-to-Time Ratio". Cette méthode se caractérise par une faible complexité temporelle et elle est basée sur des algorithmes de détermination d'un arbre de recouvrement minimal ("minimal spanning tree"). Les résultats expérimentaux ont démontré qu'en utilisant les bornes obtenues par cette méthode, les algorithmes que nous avons implantés ont pu être accélérés.

7.2- Avenues de recherche

Nous pensons que les méthodes que nous avons proposées, pour résoudre les problèmes P_1 et P_2 exposés dans la Section 1, peuvent donner de meilleurs résultats si le circuit sur lequel elles sont à appliquer contient plus de connexions possédant au moins un registre. Il serait alors intéressant de développer un algorithme de retiming qui permet de maximiser le nombre de connexions possédant au moins un registre, et de tester nos méthodes sur le circuit ainsi obtenu.

Durant la phase de calcul de la période optimale, nous pouvons aussi déterminer des ordonnancements de type au plutôt et de type au plus tard. Un élément de calcul est dit critique si son ordonnancement au plutôt est égale à celui au plus tard. En conséquence, nous pouvons localiser des chemins critiques dans le circuit, puisque ces derniers ne sont constitués que par des éléments de calcul critiques. Il serait intéressant d'optimiser ces chemins critiques en utilisant des techniques de resynthèse, des techniques d'insertion de registres additionnels comme ce qui a été fait dans l'approche de [49, 50, 51] discutée dans la Section 1.3 du Chapitre 1, et des transformations algébriques (e.g., distributivité et commutativité).

Nous pensons qu'il serait intéressant d'étendre la méthode [8] que nous avons présentée dans la Section 1 et les méthodes que nous avons proposées dans cette thèse pour traiter des circuits de contrôle. Aussi, nous croyons que pour ce type de circuits faisant des traitements itératifs irréguliers, dériver des circuits à vitesse variable est très important pour atteindre de meilleures performances. En plus, pour ces circuits, l'utilisation de plusieurs tensions d'alimentation et qui peuvent changer durant l'exécution a un grand potentiel pour réduire la consommation de puissance dynamique.

Le partage de ressources peut réduire la surface et la consommation de la puissance. Nous croyons qu'en cas de l'existence de chemins non-critiques, il est possible que certaines ressources (éléments de calcul ou registres) ne soient pas utilisées en même temps et donc qu'elles peuvent être partagées. Pour la méthode proposée dans [8], nous pensons qu'il est intéressant de développer des méthodes de détermination d'ordonnancement permettant le partage de ressources.

Bien évidemment, pour faire d'un rêve une réalité, le développement d'un outil de synthèse de circuits synchrones multi-phases semble d'être une contribution d'une grande valeur. À la date de la rédaction de cette thèse (en Juin 2001), la littérature nous indique qu'il n'y a pas encore d'outils de ce type.

Bibliographie

[1] R.-K. Ahuja, T.-L. Magnanti, and J.-B. Orlin., *Network Flows: Theory, Algorithms, and Applications*, Prentice Hall, Englewood Cliffs, NJ, 1993.

[2] I.-E. Bennour, *Estimation de la performance et méthodes d'allocation dans la synthèse de systèmes numériques, Thèse de doctorat*, DIRO, Université de Montréal, 1996.

[3] I.-E. Bennour, and E-M. Aboulhamid, "Les problèmes d'ordonnancement cycliques dans la synthèse de systèmes numeriques", *Technical Report* 996 (Oct. 1995), DIRO, Université de Montréal. `http://www.iro.umontreal.ca/~aboulham/pipeline.pdf`.

[4] L. Benini, E. Macci, M. Poncino, and G. De Micheli, "Telescopic Units: A New Paradigm for Performance Optimization of VLSI Designs," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vo. 17, No. 3, March 1998.

[5] L. Benini, G. De Micheli, "System-Level Power Optimization: Techniques and Tools," *ISLPED'1999*, San Diego, CA, USA, 1999.

[6] C. Berge, *théorie des graphes et ses applications*, Dunod, Paris, 1958.

[7] S. Bommu, M. Ciesielski, P. Kalla, N. O'Neill, "Sequential Logic Optimization with Implicit Retiming and Resynthesis," *Proceedings of International Conference on VLSI'97*.

[8] F.-R. Boyer, E.-M. Aboulhamid, Y. Savaria and M. Boyer, "Optimal Design of Synchronous Circuits Using Software Pipelining Techniques", *ACM Transactions on Design Automation of Electronic Systems*, Vo. 7, Num. 2, 2002.

[9] F. R. Boyer, E. M. Aboulhamid, and Y. Savaria, "Minimizing Sensitivity to Clock Skew Variations Using Level Sensitive Latches ," *To appear in Proceedings of European Conference on Circuit Theory and Design*, Espoo, Finland, August 28-31, 2001.

[10] F.-R. Boyer, E.-M. Aboulhamid and Y. Savaria, "An Efficient Verification Method for a Class of Multi-phase Sequential Circuits", *The 7th IEEE International Conference on Electronics, Circuits & Systems*, Lebanon, December 17-20, 2000.

[11] F. R. Boyer, E. M. Aboulhamid, Y. Savaria, and I. E. Bennour, "Optimal Design of Synchronous Circuits Using Software Pipelining Techniques," *International Conference on Computer Design*, Austin, Texas, Oct. 5-7, 1998.

[12] S.-M. Burns, *Performance Analysis and Optimization of Asynchronous Circuits*, PhD thesis, California Institute of Technology, 1991.

[13] W. Burleson, M. Ciesielski, F. Klass, W. Liu, "Wave-pipelining: A tutorial and Survey of Recent Research," *IEEE Transactions on VLSI*, Sept. 1998.

[14] T. M. Burks, K. A. Sakallah, and T. N. Mudge, "Optimization of critical paths in circuits with level-sensitive latches," *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pp. 468-473, 1994.

[15] P. Camion, "Characterisation of totally unimodular matrices", *Proc. of the Amer. Math. Soc*., Vo. 16, 1965.

[16] N.Chabini, El M.Aboulhamid, and Y.Savaria, "Scheduling and Optimal Register Placement for Synchronous Circuits Derived Using Software Pipelining Techniques," *The 13th International Conference on Microelectronics (ICM'2001)*, October 29-31, 2001, Rabat, Morocco.

[17] N.Chabini, El M.Aboulhamid, and Y.Savaria, "Fast Method for Determining an Efficient Bound on the Optimal Solution of the Cost-to-Time Ratio Problem," *Proceedings of the 5th World Multiconference on Systemics, Cybernetics and Informatics (SCI'2001), and of the 7th International Conference on Information Systems Analysis and Synthesis (ISAS'2001),* July 22-25, 2001, Orlando, Florida, USA.

[18] N.Chabini, El M.Aboulhamid, and Y.Savaria, "Reducing Register and Phase Requirements for Synchronous Circuits Derived Using Software Pipelining Techniques," *Proceedings of IEEE Computer Society Workshop on VLSI (WVLSI'2001)*, 19-20 April 2001, Orlando, Florida, USA.

[19] N.Chabini, El M.Aboulhamid, and Y.Savaria, "Determining Schedules for Reducing Power Consumption Using Multiple Supply Voltages" *To Appear in Proceedings of the International Conference on Computer Design (ICCD'2001)*, September 23 - 26, 2001,

Austin, Texas, USA.

[20] N.Chabini, El M.Aboulhamid, and Y.Savaria, “Efficient Methods for Reducing Register and Phase Requirements for Synchronous Circuits Derived Using Software Pipelining Techniques,” *To Appear in Proceedings of the 15th European Conference on Circuit Theory and Design (ECCTD’2001)*, August 28 - 31, 2001, Espoo, Finland.

[21] N.Chabini, and Y.Savaria, “Methods for Optimizing Register Placement in Synchronous Circuits Derived Using Software Pipelining Techniques,” *To Appear in Proceedings of the 14th International Symposium on System Synthesis (ISSS’2001)*, October 1-3, 2001, Montréal, Québec, Canada.

[22] N.Chabini, El M.Aboulhamid, and Y.Savaria, “Scheduling and Optimal Register Placement for Synchronous Circuits Derived Using Software Pipelining Techniques,” *To be submitted to IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems.*

[23] C.-H. Chang, E.-S. Davidson, and K.-A. Sakallah, "Maximum Rate Single-Phase Clocking of Closed Pipeline including Wave Pipelining, Stoppability, and Startability," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vo.14, No.12, Dec. 1995.

[24] T. H. Chao, Y. C. Hsu, and J. M. Ho, "Zero-skew clock net routing," *In Proceedings of the ACM/IEEE Design Automation Conference*, pp. 518-523, 1992.

[25] J.-M.Chang and M.Pedram, “Energy Minimization Using Multiple Supply Voltages”, *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, Vol.5, No.4, Dec., 1997.

[26] A. P. Chandrakasan, M. Potkonjak, R. Mehra, J. Rabaey, and R.-W. Brodersen, "Optimizing Power Using Transformations," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vo. 14, No. 1, January 1995.

[27] W. Chuang, S. S. Sapatnekar, and I. N. Hajj, ``Timing and Area Optimization for Standard-Cell VLSI Circuit Design," *IEEE Transactions on Computer-Aided Design*, Vol. 14, No. 3, pp. 308 - 320, March 1995.

[28] V. Chvatal, *Linear programming*, W. H. Freeman and Company, 1983.

[29] J. Cochet-Terrasson, G. Cohen, S. Gaubert, M. McGettrick, and J.P. Quadrat, “Numerical Computation of Spectral Elements in Max-Plus Algebra,” *In Proc. IFAC Conf. on Syst. Structure and Control* (1998).

[30] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*, New York, NY: McGraw-Hill, 1990.

[31] G.-B. Dantzig, W. Blattner, and M.-R. Rao, “Finding a Cycle in a Graph with Minimum Cost-to-Time Ratio with Application to a Ship Routing Problem,” *Theory of Graphs, International Symposium*, Rome, 1968.

[32] A. Dasdan, S. S. Irani, and R. K. Gupta, “Efficient Algorithms for Optimum Cycle Mean and Optimum Cost to Time Ratio Problems,” *In Proc. 36th Design Automation Conf. (DAC)*, Jun. 1999.

[33] A. Dasdan and R. K. Gupta, “Faster Maximum and Minimum Mean Cycle Algorithms for System Performance Analysis,” *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, 17(10), Oct. 1998.

[34] A. Dasdan, S. S. Irani, and R. Gupta, *An Experimental Study of Minimum Mean Cycle Algorithms*, Tech. Rep. #98-32, Univ. of California, Irvine, 1998.

[35] G. De Micheli, *Synthesis and Optimization of Digital Circuits*, New York, NY: McGraw-Hill, 1994.

[36] G. De Micheli, "Synchronous logic synthesis: Algorithms for cycle time minimization," *IEEE Transac-tions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 10, pp. 63-73, Jan. 1991.

[37] R. B. Deokar and S. S. Sapatnekar, ``A Fresh Look at Retiming via Clock Skew Optimization," *Proceedings of the ACM/IEEE Design Automation Conference*, pp. 304-309, 1995.

[38] R. B. Deokar and S. S. Sapatnekar, ``A Graph-theoretic Approach to Clock Skew Optimization," *Proceedings of the IEEE International Symposium on Circuits and*

Systems, pp. 1.407-1.410, 1994.

[39] M. Edahiro, "An effecient zero-skew routing algorithm," *In Proceedings of the ACM/IEEE Design Automation Conference*, pp. 375-380, 1994.

[40] J. P. Fishburn, "Clock skew optimization," *IEEE Transactions on Computers*, Vol. 39, pp. 945-951, July 1990.

[41] E. G. Friedman, X. Liu, and M. C. Papaefthymiou, "Minimizing sensitivity to delay variations in high-performance synchronous circuits," *Proceedings of Design Aut. and Test in Europe*, 1999, pp. 643-649.

[42] Gary K. Yeap, *Practical Low Power Digital VLSI Design*, Kluwer Academic Publishers, ISBN: 0-7923-8009-6.

[43] S.-H. Gerez, S.-M.-H. de Groot, and O.-E. Herrmann, "A Polynomial-Time Algorithm for the Computation of the Iteration-Period Bound in Recursive Data- Flow Graphs", *IEEE Trans. on Circuits and Syst.-1*, No. 1, Vo. 39, Jan. 1992.

[44] M.-V. Golitschek, "The Cost-to-Time Ratio Problem for Large or Infinite Graphs," Discrete Applied Mathematics, 16, 1987.

[45] A.-V. Goldberg and R.-E. Tarjan, "Solving Minimum-Cost Flow Problems by Successive Approximation," *Proceedings of the 19th Annual ACM Symposium on Theory of Computing*, New York City, May 25-27, 1987.

[46] C. Hanen, "Study of NP-hard Cyclic Scheduling Problem: the Recurrent Job-Shop", *European Journal of Operation Research*, Vo. 72, 1994.

[47] M. Hartmann, *On Cycle Means and Cyclic Stuffing*, Tech. Report, No. UNC/OR/TR-90/14, University of North Carolina, Chapel Hill, 1990.

[48] M. Hartmann, and J.-B. Orlin, "Finding Minimum Cost to Time Ratio Cycles with Small Integral Transit Times," *Networks* 23 (1993).

[49] S. Hassoun, "Fine Grain Incremental Rescheduling via Architectural Retiming", *International Symposium on System Synthesis*. Dec., 1998.

[50] S. Hassoun, and C. Ebeling, "Using Precomputation in Architecture and Logic Synthesis", *Proceeding of the International Conference on Computer-Aided Design*, Nov., 1998.

[51] S. Hassoun, and C. Ebeling, "Architectural Retiming: Pipelining Latency-Constrained Circuits", *Proceedings of the ACM/IEEE Design Automation Conference*, Las Vegas, June, 1996.

[52] M. Heshami and B. A. Wooley, "A 250-MHz skewed-clock pipelined data buffer," *IEEE Journal of Solid-State Circuits*, Vol. 31, pp. 376-383, Mar. 1996.

[53] H. Hulgaard, S.-M. Burns, T. Amon, and G. Borriello, "An Algorithm for Exact Bounds on the Separation of Events in Concurent Systems," *IEEE Trans. on Computers*, v. 44, no. 11, Nov. 1995.

[54] A.-T. Ishii, C.-E. Leiserson, and M. C. Papaefthymiou, "Optimizing two-phase, level-clocked circuitry", *Journal of the ACM* 44, 1 , Jan. 1997.

[55] K. Ito, and K.-K. Parhi, "Determining the Minimum Iteration Period of an Algorithm," *Journal of VLSI Signal Processing* 11, 3 (Dec. 1995).

[56] D. Joy and M. Ciesielski, "Clock period minimization with wave pipelining," *IEEE Transactions on Computer-Aided Design*, Vo. 12, No.4, pp. 461-472, Apr. 1993.

[57] D.-A. Joy, and M.-J. Ciesielski, "Placment For Clock Period Minimization With Multiple Wave Propagation," *Proceedings of the 28th ACM/IEEE Design Automation Conference*, 1991.

[58] P. Kalla, M. Ciesielski, "Performance Driven Resynthesis by Exploiting Retiming-Induced State Register Equivalence," *Design & Test in Europe Conference, DATE'99*.

[59] N. Karmakar, "A New polynomial-time algorithm for linear programming", *Combinatorica*, Vo. 4, 1984.

[60] R.-M. Karp, "Characterization of the Minimum Cycle Mean in a Digraph," *Discrete Mathematics* 23, 1978.

[61] R.-M. Karp, and J.-B. Orlin,"Parametric Shortest Path Algorithms with an Application to Cyclic Stafing," *Discrete Applied Mathematics* 3, 1981.

[62] L.-G. Khachian, “A Polynomial algorithm in linear programming”, *Soviet Math. Doklady*, Vo. 20, 1979.

[63] S. Y. Kung, H. J. Whitehouse and T. Kailath, *VLSI and Modern Signal Processing*, Prentice-Hall, Inc., Englewood Cliffs, NJ, 1985, pp. 259-60.

[64] K.N. Lalgudi and M.C. Papaefthymiou, "Fixed-Phase Retiming for Low Power Design," *In 1996 International Symposium on Low Power Electronics and Design, August 1996*.

[65] Laurence A. Wolsey, *Integer Programming*, John Wiley & Sons, Inc., 1998.

[66] E.-L. Lawler, *Combinatorial Optimization: Networks and Matroids*, Holt, Reinhart, and Winston, New York, NY, USA, 1976.

[67] E.-L. Lawler, “Optimal Cycles in Doubly Weighted Directed Linear Graphs,” *Theory of Graphs, International Symposium*, Rome, 1968.

[68] Y.-M. Li, M.-A. Jabori, "A Zero-Skew Clock Routing Scheme for VLSI Circuits," *Proc. Int. Conf. on Computer-Aided Design*, 1992.

[69] Y.-R.Lin and A.-C.-H. Wu, “Scheduling Techniques for Variable Voltage Low Power Designs”, *ACM Transactions on Design Aut. of Elec. Sys.*, Vol.2, N0.2, April, 1997.

[70] C. Legl, P. Vanbekbergen, and A. Wang, "Retiming of edge-triggered circuits with multiple clocks and load enables," *IWLS'97*, 1997.

[71] C. E. Leiserson and J. B. Saxe, "Retiming synchronous circuitry," *Algorithmica*, Vol. 6, pp. 5-35, 1991.

[72] B. Lockyear and C. Ebeling, "Optimal retiming of level-clocked circuits using symmetric clock schedules," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 13, pp. 1097-1109, Sept. 1994.

[73] B. Lockyear and C. Ebeling, "The practical application of retiming to the design of high-performance systems," *Proceedings of the IEEE/ACM International Conference on Computer- Aided Design*, pp. 288-295, 1993.

[74] The LP_Solve Tool: `ftp://ftp.ics.ele.tue.nl/pub/lp_solve/`

[75] E. Macci, M. Pedram, and F. Somenzi, "High-Level Power Modeling, Estimation, and Optimization," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vo. 17, No. 11, Nov. 1998.

[76] N. Maheshwari and S. S. Sapatnekar, ``Optimizing Large Multiphase Level-Clocked Circuits," *IEEE Transactions on Computer-Aided Design*, Vol. 18, No. 9, pp. 1249 -1264, September 1999.

[77] N. Maheshwari and S. S. Sapatnekar, ``Efficient Retiming of Large Circuits," *IEEE Transactions on VLSI Systems*, Vol. 6, No. 1, pp. 74 - 83, March 1998.

[78] N. Maheshwari and S. S. Sapatnekar, ``Efficient Minarea Retiming of Large Level-clocked Circuits," *Proceedings of the Design Automation and Test in Europe (DATE) Conference*, pp. 840-845, 1998.

[79] N. Maheshwari and S. S. Sapatnekar, ``Retiming Level-clocked Circuits for Latch Count Minimization," *Proceedings of the ACM International Workshop on Timing Issues in the Specification and Synthesis of Digital Systems*, pp. 135-140, 1997.

[80] N. Maheshwari and S. S. Sapatnekar, ``Minimum Area Retiming with Equivalent Initial States," *Proceedings of the IEEE/ACM International Conference on Computer-aided Design*, pp. 216-219, 1997.

[81] N. Maheshwari and S. S. Sapatnekar, ``An Improved Algorithm for Minimum Area Retiming," *Proceedings of the ACM/IEEE Design Automation Conference*, pp. 2-6, 1997.

[82] N. Maheshwari and S. S. Sapatnekar, ``A Practical Algorithm for Retiming Level-Clocked Circuits," *Proceedings of the IEEE International Conference on Computer Design*, pp. 440-445, 1996.

[83] S. Malik, E. M. Sentovich, R. K. Brayton, A. Sangiovanni-Vincentelli, "Retiming and Resynthesis: Optimizing Sequential Networks with Combinational Techniques," *IEEE Transactions on CAD/ICAS*, Vo. 10, No. 1, January 1991.

[84] S. Malik, K. J. Singh, R. K. Brayton, and A. Sangiovanni-Vincentelli, "Performance optimization of pipelined circuits," *Proceedings of the IEEE International Conference on*

Computer- Aided Design, pp. 410-413, 1990.

[85] H.-G. Martin, "Retiming by combination of relocation and clock delay adjustment," *Proceedings of the European Design Automation Conference*, pp. 384-389, 1993.

[86] N. Megiddo, "Combinatorial Optimization with Rational Objective Functions," *Math. of Oper. Res.* 3, 1979.

[87] J. Monteiro, S. Devadas, and A. Ghosh, "Retiming sequential circuits for low power," *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pp. 398-402, 1993.

[88] J. L. Neves and E. G. Friedman, "Circuit synthesis of clock distribution networks based on nonzero clock skew," *Proceedings of the IEEE International Symposium on Circuits and Systems*, pp. 4.175-4.178, 1994.

[89] P. Pan, "Performance-Driven Integration of Retiming and Resynthesis," *Proceedings of the ACM/IEEE Design Automation Conference*, New Orleans, Louisiana, 1999.

[90] P. Pan, A.-K. Karandikar, and C.-L. Liu, "Optimal Clock Period Clustering for Sequential Circuits with Retiming," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vo. 17, No. 6, June 1998.

[91] C. Papachristou, M. Spinning, and M. Nourani, "An Effective Power Management Schem for RTL Design Based on Multiple Clocks," *Proceedings of the ACM/IEEE Design Automation Conference*, Las Vegas, NV, 1996.

[92] S. Pullela, N. Menezes, and L. T. Pillage, "Reliable nonzero clock skew trees using wire width optimization," *Proceedings of the ACM/IEEE Design Automation Conference*, pp. 165-170, 1993.

[93] V. Raghunathan, S, Ravi, and G. Lakshminarayana, "Integrating Variable-Latency Components into High-Level Synthesis," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol.19, No.10, Oct. 2000.

[94] S.Raje and M.Sarrafzadeh, "Variable Voltage Scheduling", *In Proc. of the Int. Sym. on Low Power Design*, Monterey, CA, Aug., 1995.

[95] R. Rudell, "Tutorial: Design of a Logic Synthesis System," *Proceedings of the ACM/IEEE Design Automation Conference*, Las Vegas, NV, 1996.

[96] K.-A. Sakallah, T.-N. Mudge, T.-M. Burks, and E.-S. Davidson, "Synchronization of Pipelines,"*IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vo.12, No. 8, Aug. 1993.

[97] K. A. Sakallah, T. N. Mudge, and O. A. Olukotun, "Analysis and design of latch-controlled synchronous digital circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 11, pp. 322-333, Mar. 1992.

[98] K.-A. Sakallah, T.-N. Mudge, and O.-A. Olukotun, "Analysis and Design of Latch-Controlled Synchronous Digital Circuits," *Proceedings of the 27th ACM/IEEE Design Automation Conference*, 1990.

[99] S. S. Sapatnekar and R. B. Deokar, ``Utilizing the Retiming-Skew Equivalence in a Practical Algorithm for Retiming Large Circuits," *IEEE Transactions on Computer-Aided Design*, Vol. 15, No. 10, pp. 1237 - 1248, October 1996.

[100] A. Schrijver, *theory of linear and integer programming*, John Wiley and Sons, 1986.

[101] N. Shenoy and R. Rudell, "Effecient implementation of retiming," *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pp. 226-233, 1994.

[102] N. V. Shenoy, R. K. Brayton, and A. L. Sangiovanni-Vincentelli, "Resynthesis of multi-phase pipelines," *Proceedings of the ACM/IEEE Design Automation Conference*, pp. 490-496, 1993.

[103] K.-J. Singh, and A. Sangiovanni-Vincentelli, "A Heuristic Algorithm for the Fanout Problem," *Proceedings of the Design Automation Conference*, pp. 357-60, 1990.

[104] K.-J. Singh, A. R. Wang, R. K. Brayton, and A. Sangiovanni-Vincentelli, "Timing Optimization of Combinational Logic," *Proceeding of the International Conference on Computer-Aided Design*, pp. 282-285, 1988.

[105] T. Soyata, E. G. Friedman, and J. H. Mulligan, Jr., "Incorporating internconnect, register and clock dis- tribution delays into the retiming process," *IEEE Transactions on Computer-*

Aided Design of Integrated Circuits and Systems, Vol. 16, pp. 165-120, Jan. 1997.

[106] H. J. Touati and R. K. Brayton, "Computing the initial states of retimed circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 12, pp. 157-162, Jan. 1993.

[107] H. J. Touati, H. Savoj, and R. K. Brayton, "Delay optimization of combinational logic circuits by clustering and partial collapsing," Proceedings of Int. Conf. on Computer-Aided Design, pp. 188-91, 1991.

[108] R.-S. Tsay, "An exact zero-skew clock routing algorithm," *IEEE Transactions on Computer- Aided Design,* Vol. 12, pp. 242-249, Feb. 1993.

[109] R.-S. Tsay, "Exact zero skew," *In Proceedings of the IEEE International Conference on Computer-Aided Design*, pp. 336-339, 1991.

[110] K.Usami and M.Horowitz, “Clustered Voltage Scaling Technique for Low-Power Design”, *In Proc. Int. Workshop on Low Power Design*, 1995.

[111] J. Wang, C. Einsenbeis, M. Jourdan, and B. Su, “Decomposed Software Pipelining,” *International Journal of Parallel Programming*, vol. 22, no. 3, pp. 351-373,1994.

[112] A. van der Werf, J. L. Van Meerbergen, E. H. L. Aarts, W. F. J. Verhaegh, and P. Lippens, "Effecient timing constraint derivation for optimally retiming high speed processing units," *International Symposium on High Level Synthesis*, pp. 48-53, 1994.

[113] D.-C. Wong, G.De Micheli and M.-J. Flynn, ``Designing High-Performance Digital Circuits Using Wave Pipelining: Algorithms and Practical Experiences," *IEEE Transactions on CAD/ICAS*, January 1993, pp. 25-46.

[114] N.-E. Young, R.-E. Tarjan, and J.-B. Orlin, “Faster Parametric Shortest Path and Minimum-Balance Algorithms,” *Networks* 21, 1991.