Générateur de ROMs basse consommation

Arnaud Turier, Lotfi Ben Ammar : ATMEL (aturier,lbenammar)@es2.fr
Amara Amara : ISEP (Amara.Amara@isep.fr)

Résumé

L'importance des mémoires dans un système intégré fait que sa consommation dépend fortement de celle des mémoires. Il est donc nécessaire de disposer de larges mémoires avec une faible consommation. Dans cet article, nous présentons un générateur de ROMs pour des applications embarquées, faible tension, faible consommation. Plusieurs techniques de réduction de la consommation statique et dynamique sont mises en œuvre telles que le partitionnement en sous blocs, l'utilisation, de la précharge sélective, d'un pipe-line et d'une structure auto-contrôlée des signaux. La méthodologie employée pour développer ce générateur ainsi que le flot de données dans lequel il peut être utilisé sont détaillés. En comparaison avec un ancien générateur, nous montrons que les résultats obtenus en terme de consommation dynamique et de densité sont très satisfaisants.

1 Introduction

Dans la conception d'un ASIC, la mémoire devient un élément incontournable, que ce soit des ROMs pour stocker du code pour un micro-contrôleur ou des SRAMs pour créer un cache [1] [2]. Elle prend une place importante en terme de surface et de puissance consommée. Réduire la consommation des mémoires revient à diminuer celle du circuit entier. C'est donc un différentiateur important. L'ancien générateur [3] ne nous permettant plus de répondre aux exigences, sans cesse croissantes, en terme de capacité de stockage et de consommation, nous avons opté pour le développement de nouvelles mémoires. Le temps de conception des mémoires étant très long, il est préférable de développer une méthodologie pour les générer, les caractériser rapidement et les intégrer dans le flot de conception d'un circuit. L'approche générateur est la plus adaptée car elle permet de connaître très tôt la taille du bloc généré, ses caractéristiques temporelles ainsi que la puissance consommée. En outre, on peut disposer immédiatement des différentes vues de la mémoire (symbolique, fonctionnelle, dessin de masques...) à utiliser dans les différentes étapes du flot de conception. Nous exposons ici, la réalisation d'un générateur de ROMs embarquées pour des applications visant la faible consommation. L'architecture et les techniques utilisées pour réduire la consommation seront exposées dans la première partie. Nous détaillerons ensuite le fonctionnement du générateur, puis nous donnerons les résultats obtenus avec la nouvelle architecture, en les comparant avec ceux obtenus avec l'ancien générateur.

2 Architecture

2.1 Partitionnement en blocs / ligne de mot divisée

L'architecture classique d'une ROM est bien connue [4]. Les ROMs obtenues avec notre ancien générateur sont construites sur ce modèle qui se compose de quatre parties : le plan mémoire, le décodeur de lignes, le décodeur de colonnes et un bloc pour le contrôle. Cependant, cette organisation convient mal à des mémoires de grande taille : en effet, les charges dynamiques (charges liées aux grilles des transistors) deviennent trop importantes, notamment pour les portes du décodeur de ligne. Si l'on considère l'exemple d'une mémoire d'une capacité totale d'un mégabit avec un facteur de forme carré (1024 lignes $\times$ 1024 colonnes), le nombre de grilles vues par le décodeur de ligne est 1024. Cette charge importante contribue à augmenter à la fois le temps de propagation des signaux sur la ligne de mot et à accroître la puissance consommée lors de l'activation de ces signaux. Pour remédier à ces problèmes, nous avons opté pour une architecture de type multi-blocs en utilisant le principe de ligne de mot divisée [5]. Étant donné que le nombre maximal de blocs contigus pour un "buffer" de ligne reste inférieur à quatre (voir Fig 1), nous avons utilisé ce concept plutôt que celui de ligne de mot hiérarchique [6], plus adapté à un grand nombre de bloc contigus.

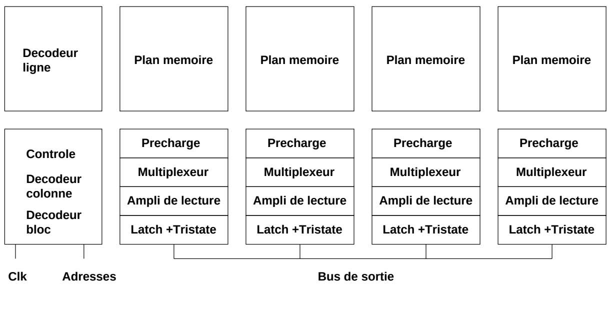

FIG. 1 – *Architecture multi-blocs*

Avec le principe de la ligne de mot divisée (Fig. 2), la capacité sur une ligne de mot vue par la sortie du décodeur de ligne est considérablement diminuée : la capacité totale, due aux grilles d'une ligne, est sectionnée en ligne de mot locales. Chaque décodeur local voit un nombre de grilles réduit et le décodeur de ligne global attaque un nombre de portes égal au nombre de blocs contigus. En revanche, la capacité liée au routage reste inchangée.

Le concept de ligne de mot divisée a été étendu à tous les autres signaux qui commandent les blocs de précharge,

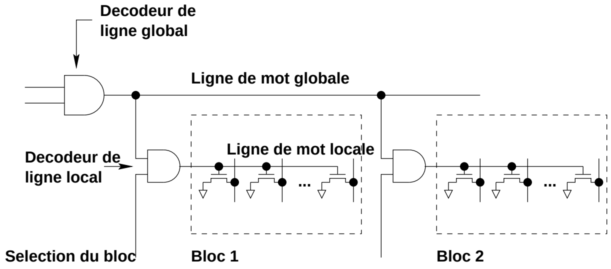

FIG. 2 – *Le principe de ligne de mot divisée*

de multiplexage et l'étage de sortie de manière à diminuer le temps d'accès et à réduire la consommation totale. Ainsi, la mémoire n'est plus organisée en deux dimensions mais en trois : une dimension X pour le décodeur de lignes, une dimension Y pour le décodeur de colonnes et une dimension Z pour le décodeur de blocs. Afin de réduire encore la puissance consommée, un seul bloc est activé lors d'un accès en lecture, les autres blocs n'ayant qu'une consommation statique. Ainsi, tous les bits d'un même mot sont stockés dans le même bloc.

2.2 Précharge sélective

Généralement, toute lecture est précédée d'une phase de précharge de toutes les lignes de bit à "vdd", ce qui est le cas dans l'ancien générateur. Ainsi, au moment de la lecture, à l'intérieur d'un même bloc, plusieurs lignes de bit peuvent être déchargées sans pour autant être utilisées. Sur la figure 3, les points noirs représentent des transistors NMOS connectant une ligne de mot aux lignes de bit. L'activation de la ligne de mot dans cet exemple, montre que la moitié (4 sur 8) des lignes préchargées à "1" seraient déchargées quelque soit la colonne lue. Afin d'économiser cette énergie perdue, nous utilisons le principe de précharge sélective [7], qui consiste à ne précharger que les lignes de bit qui seront nécessaires pour la lecture d'un mot. La précharge est conditionnée par un signal interne au bloc, ce qui permet de ne la déclencher que pendant une durée juste suffisante avant de démarrer la lecture. Ce conditionnement est nécessaire pour notre approche basse consommation, ce qui n'est pas toujours le cas dans les architectures traditionnelles [8]. En plus de ce signal de conditionnement, un bus parcourt la mémoire afin de déterminer les lignes de bit à précharger. Étant donné que cette information est la même que celle dans les multiplexeurs de sortie, on utilise le même décodeur. On utilise aussi le même bus pour le multiplexage et la précharge en plaçant le bloc de précharge juste au dessus du multiplexeur (Fig. 3) et non pas tout en haut de la mémoire comme c'est le cas traditionnellement [6].

Les transistors de précharge sont de type PMOS dans la mesure où l'on souhaite opérer à des basses tensions, proches de la tension de seuil pour lesquelles on ne veut subir aucune dégradation des signaux.

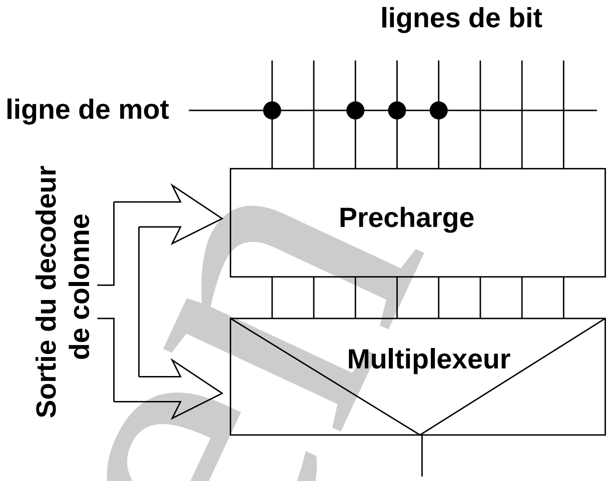

FIG. 3 – *Précharge sélective*

2.3 Pipe-line

L'inconvénient de la précharge sélective réside dans le fait qu'il est nécessaire de commencer tout cycle de lecture par précharger la ligne de bit du mot à lire et donc décoder les adresses des colonnes correspondantes. Afin de remédier à cet inconvénient, on utilise un pipe-line à deux étages : le premier étage réalise le décodage et le second la précharge et la lecture. Ainsi, lorsque l'on active la mémoire, l'information sur les blocs et les colonnes à précharger est déjà connue et la précharge peut alors démarrer immédiatement. Le schéma du pipe-line est donné à la figure 4. Quand le signal CK est à l'état bas, les adresses traversent le premier étage de "latches" et le décodage est réalisé. Quand le signal CK passe à l'état haut, l'information issue des décodeurs traverse le second étage de "latches" et rend possible la précharge sur les colonnes désignées. Les adresses en entrée de la mémoire peuvent alors changer car le premier étage de "latches" est verrouillé.

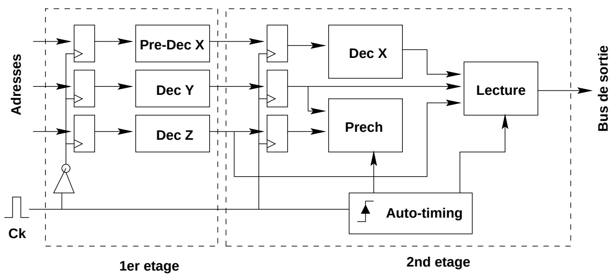

FIG. 4 – *Le pipe-line*

2.4 "Auto-timing"

Souvent deux phases sont nécessaires lors de l'écriture/lecture dans les mémoires statiques : une phase de précharge et une phase de lecture. La phase de précharge débute juste après la phase de lecture afin de replacer les lignes de bit dans un état préchargé pour la lecture suivante. Le temps pendant lequel la précharge reste active peut durer plus longtemps que nécessaire si la mémoire n'est pas accédée à sa fréquence maximale de fonctionnement. Pendant la phase de précharge, des courants de fuites apparaissent notamment au niveau des transistors NMOS du plan mémoire. Ces courants seront d'autant plus importants que la mémoire restera inactive. Aussi, pour réduire ces courants de fuites à travers le plan mémoire, une nouvelle gestion des phases a été établie : la phase de précharge ne débute plus systématiquement après la lecture pendant une durée inconnue mais juste avant la phase de lecture pendant le temps strictement nécessaire à l'établissement du niveau requis sur les lignes de bit. Si ce dispositif ne modifie pas le temps de cycle par rapport aux architectures traditionnelles, il allonge le temps d'accès qu'il rend quasiment égal au temps de cycle (Fig. 5).

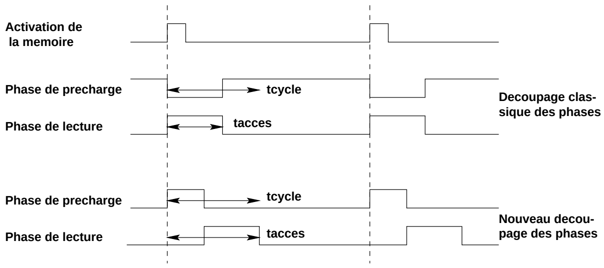

FIG. 5 – *Gestion des phases de précharge et de lecture*

Afin de générer les signaux pour l'activation de la précharge et de la lecture, une structure auto-contrôlée des signaux, "auto-timing", est utilisée (Fig. 6). Outre la réduction de la consommation statique, cette structure permet aussi de s'affranchir des contraintes de rapport cyclique du signal d'activation de la mémoire. De plus, les ROMs compilées par le générateur varient beaucoup en terme de taille (nombre de blocs, nombre de lignes) et ce pour plusieurs technologies différentes. Pour adapter la génération interne des signaux de précharge et de lecture à toutes ces variations, cette structure détermine automatiquement les temps de précharge et de lecture en utilisant une ligne de bit factice qui représente un pire cas où tous les bits de la colonne seraient codés par un transistor. Sur le diagramme de la figure 6, le front montant de CK déclenche le début de la précharge sur la ligne de bit factice et aussi dans le bloc à précharger en utilisant l'information issue du décodeur de bloc. Lorsque le niveau requis est atteint sur la ligne de bit factice, le signal fin de précharge est actif et pris en compte par l'auto-timing qui stoppe la précharge et rend actif le signal de lecture pour la ligne de bit factice et pour le bloc à lire en utilisant encore l'information provenant du décodeur de blocs. La fin de lecture est détectée en sortie de l'amplificateur de lecture. Étant donné que cette ligne de bit factice constitue un pire cas au niveau de la charge, on est certain que lorsque celle-ci est suffisamment déchargée pour activer l'amplificateur de lecture, toutes les autres lignes de bit de la mémoire auront déjà réagit. La fin de la lecture replace la mémoire dans un état stable, le signal de précharge n'étant pas actif.

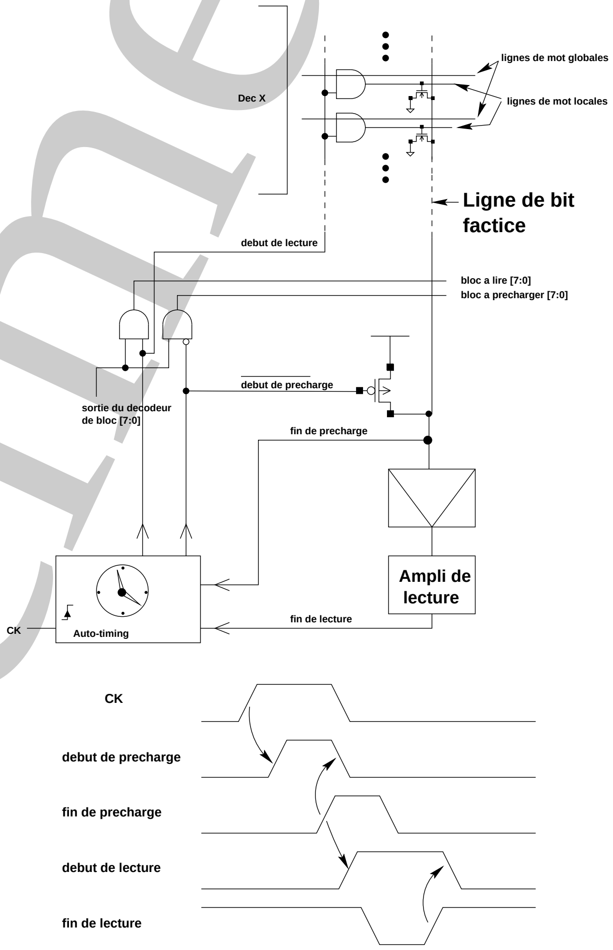

FIG. 6 – *auto-timing*

3 Le générateur

3.1 Développement et validation

Le flot de développement et de validation utilisé est présenté à la figure 7. La méthode de conception utilisée est de type “full-custom” dans la mesure où les structures sont répétitives. La partie contrôle n’est pas synthétisée non plus afin de garder une maîtrise totale des temps de propagation des signaux et de la consommation. Des schémas logiques de quelques instances sont créés et validés à l’aide de simulations logiques. Une fois la validation fonctionnelle réalisée, on procède au dimensionnement des transistors, en fonction des exigences en terme de vitesse et de consommation. Enfin, le dessin des masques de fabrication peut alors commencer, suivi des vérifications habituelles : vérification des règles de dessin (DRC), comparaison du dessin des masques de fabrication avec les schémas logiques (LVS). Une dernière simulation sur des blocs complètement extraits permet de rendre compte des effets des capacités parasites sur le fonctionnement de la mémoire et de réitérer le flot de conception en cas de besoin.

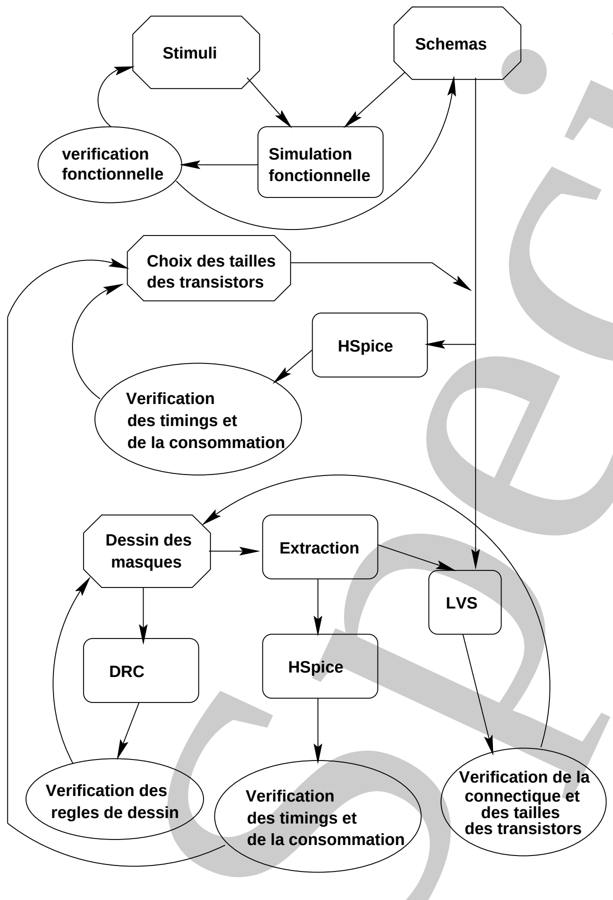

FIG. 7 – *Flot de conception*

Comme il serait difficile de générer toutes les configurations possibles pour valider exhaustivement le générateur, un jeu représentatif des configurations critiques est défini. Ces configurations (quelques dizaines), permettent de vérifier le bon fonctionnement et la validité du générateur.

3.2 Caractérisation

Les temps de propagation et les contraintes indiqués par le générateur sont pré-calculés c’est-à-dire qu’ils sont issus d’une caractérisation effectuée sur des schémas critiques utilisés uniquement pendant la phase de développement puis recoupés avec une caractérisation faite sur silicium. Les chemins critiques sont paramétrables et peuvent ainsi reproduire la flexibilité du générateur. Pour la création des chemins critiques, nous avons utilisé un modèle basé sur des générateurs de courant contrôlés par des ampèremètres, pour représenter les capacités dynamiques des transistors [9]. Un circuit de test contenant un jeu réduit des configurations critiques est réalisé afin de vérifier la fonctionnalité et de mesurer les performances des mémoires.

3.3 Interface utilisateur

Une interface écrite en C permet à l’utilisateur de rentrer les caractéristiques de la mémoire à générer. Les ROMs compilées sont dédiées à des applications embarquées avec une capacité totale qui varie de 64Kb à 4Mb avec des mots de 8, 16 ou 32 bits de largeur. Ce qui correspond aux demandes les plus courantes, dans la mesure où les ROMs servent souvent à stocker du code pour des micro-contrôleurs dont la largeur en bits des mots est fixée. L’utilisateur a le choix, pour une capacité mémoire donnée, entre plusieurs facteurs de forme en partant de quatre modèles possibles pour l’organisation de la mémoire. En agissant sur le facteur de forme, on modifie le temps d’accès, la consommation et la surface de manière à atteindre les objectifs fixés. La figure 8 montre les différents modèles d’organisation des blocs pour construire une mémoire. La structure de base est le modèle 1 qui, en étant symétrisé plusieurs fois, donne les autres modèles possibles. La capacité d’une mémoire augmente avec le numéro du modèle : la plus petite des capacités (64 kb) est générée à partir du modèle 1 et la plus grande (4 Mb) à partir du modèle 4.

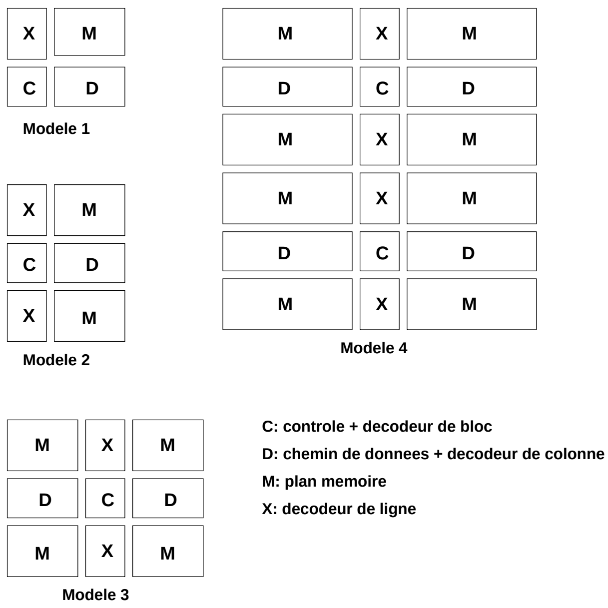

FIG. 8 – *les différents modèles d’organisation*

Une vue symbolique est fournie ainsi qu'un modèle fonctionnel intégrant les temps de propagation et les contraintes afin d'être intégrés à la méthodologie de conception. Une vue dite "abstract" est générée et est utilisée pour commencer le placement et le routage du circuit avant même que le dessin de masques ne soit encore connu ce qui permet de retarder jusqu'au dernier moment la programmation de la mémoire. Les différentes vues que peut compiler le générateur, pour une configuration donnée, sont indiquées sur la figure 9.

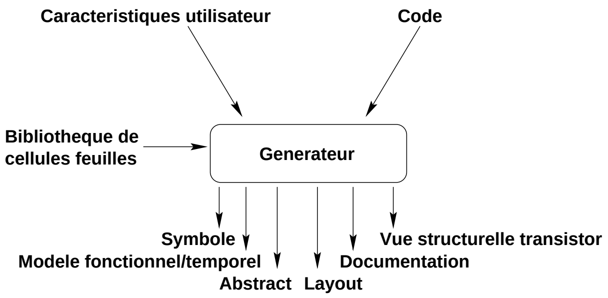

FIG. 9 – *Flot de génération d'une mémoire*

La construction finale du dessin des masques des mémoires est réalisée par le logiciel "Structure Compiler" de Cadence, à partir d'une bibliothèque de cellules feuilles préexistante. Les vues symbole, "abstract" et la vue structurelle au niveau transistor sont créées à l'aide de programmes en langage Skill de la chaîne Cadence.

3.4 Validation par l'utilisateur

Comme nous l'avons indiqué précédemment, toutes les configurations possibles ne sont pas validées. Aussi, pour éliminer tout risque de défaillance, l'utilisateur a la possibilité de vérifier lui-même chaque mémoire générée en lançant successivement une vérification des règles de dessin (DRC), une extraction du dessin des masques de fabrication, puis une comparaison du résultat de l'extraction avec une "netlist" au niveau transistor créée automatiquement par le générateur. La cohérence des différentes vues est aussi assurée : nom des "pins", leur type (entrée, sortie, alimentation) (Fig. 10).

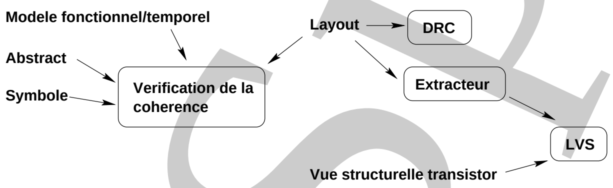

FIG. 10 – *Flot de vérification utilisateur*

4 Résultats

Actuellement, le compilateur existe pour trois technologies différentes : CMOS $0,5\mu m$, CMOS/NVM $0,5\mu m$ et CMOS $0,35\mu m$. Les tableaux 1 et 2 donnent la densité et la puissance dynamique consommée obtenues avec l'ancien [3] et le nouveau générateur. Les valeurs sont données pour une technologie $0,35\mu m$. L'ancien générateur ne peut fournir qu'une capacité maximale de 128 Kb ce qui oblige à faire la comparaison sur les petites configurations du nouveau générateur. On voit que la densité à été améliorée avec un gain de 20% sur de petites configurations, donc le cas le plus défavorable, puisque le plan mémoire est peu important par rapport au reste de la mémoire. En revanche, pour les configurations maximales, le gain en densité est de 92%. C'est pour la puissance consommée que le gain est le plus important. Pour une capacité de 128 Kb, la puissance est divisée par 22, et on note que la puissance consommée par l'instance 32 Kb de l'ancien générateur correspond à la valeur de la nouvelle instance 4096 Kb, ce qui représente une division par 128 de la puissance consommée.

TAB. 1 – *Performances obtenues avec l'ancien générateur ($0,35\mu m$ process typique)*

Capacité (Kb)	Densité (Kb/mm^2)	Puissance (mW/MHz)
32	133	0.79
64	168	1.36
128	191	2.49

TAB. 2 – *Performances obtenues avec le nouveau générateur ($0,35\mu m$ process typique)*

Capacité (Kb)	Densité (Kb/mm^2)	Puissance (mW/MHz)
64	213	0.10
128	237	0.11
192	291	0.13
256	301	0.14
4096	368	0.88

La précharge sélective contribue pour beaucoup dans la réduction de la consommation : sur le plan de la consommation statique tout d'abord, les fuites au niveau du plan mémoire sont limitées par l'utilisation d'un signal de précharge de durée précise (Cf. section 2.4) et par la précharge d'un nombre limité de ligne de bit. Sur le plan de la consommation dynamique, seules 32 lignes de bit dans toute la mémoire sont préchargées quelque soit la configuration. Si l'on considère l'exemple d'une configuration d'un mégabit, organisée en 16 blocs de 256 lignes et 256 colonnes chacun, on ne précharge que 32 lignes de bit au lieu des 4096 lignes de bit si la mémoire était organisée de manière classique (256 lignes et 4096 colonnes).

5 Conclusion

Dans cet article, nous avons présenté un générateur de mémoires ROM basse consommation. Nous avons mis en œuvre plusieurs techniques pour réduire la consommation tout en restant indépendant du procédé de fabrication, ce qui favorise la réutilisation et la portabilité du générateur. Par rapport à un ancien générateur, nous avons à la fois, augmenté la capacité adressable ainsi que la densité, et diminué la puissance consommée. Le générateur a été validé sur silicium et est disponible dans plusieurs technologies submicroniques. Toute la circuiterie est réalisée de manière à pouvoir utiliser les mémoires générées à basse tension d'alimentation.

Références

[1] A. Wild, J. Quigley, J. Feddeler, S. Ledford, J. Caravella, *et al.*, "A 0.9-V microcontroller for portable applications," *IEEE J. Solid-State Circuits*, vol. 32, pp. 1049-1055, July 1997.

[2] D. Draper, M. Crowley, J. Holst, G. Favor, A. Schoy, et al., "Circuit techniques in 266-MHz MMX-enabled processor," *IEEE J. Solid-State Circuits*, vol. 32, pp. 1650-1664, Nov. 1997.

[3] ATMEL CBIC AT55K/ECAT05 Library Databook, Sept. 1997.

[4] J. M. Rabaey, "Digital integrated circuits, a design perspective," Upper Saddle River, NJ : Prentice Hall, 1996.

[5] M. Yoshimoto, K. Anami, H. Shinohara, T. Yoshihara, H. Takagi, *et al.* «A divided word-line structure in the static RAM and its application to a 64k full CMOS RAM,» *IEEE J. Solid-State Circuits*, vol. SC-18, pp. 479-485, Oct. 1983.

[6] T. Hirose, H. Kuriyama, S. Murakami, K. Yuzuriha, T. Mukai, et al., "A 20-ns 4-Mb CMOS SRAM with hierarchical word decoding architecture," *IEEE J. Solid-State Circuits*, vol. 25, pp. 1068-1074, Oct. 1990.

[7] N. Weste, and K. Eshraghian, "*Principles of CMOS VLSI Design : A Systems Perspective, Second edition,*" Reading, MA., Addison-Wesley, p. 586, 1993.

[8] B. Price, "*Semiconductor memories, Second edition,*" Chichester, UK, John Wiley & Sons, p. 8, 1991.

[9] A. Turier, L. Ben Ammar, A. Amara, "An Accurate Power and Timing Modeling Technique Applied To A Low-Power ROM Compiler," *Patmos'98*, Lynby, Denmark, 1998, pp. 181-189.