

Systèmes intégrés adaptatifs ultra basse consommation pour l'Internet des Objets

Guillaume Patrigeon

▶ **To cite this version:**

Guillaume Patrigeon. Systèmes intégrés adaptatifs ultra basse consommation pour l'Internet des Objets. Micro et nanotechnologies/Microélectronique. Université de Montpellier, 2020. Français. NNT : . tel-02947275

HAL Id: tel-02947275

https://hal-lirmm.ccsd.cnrs.fr/tel-02947275

Submitted on 23 Sep 2020

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

THÈSE POUR OBTENIR LE GRADE DE DOCTEUR DE L'UNIVERSITÉ DE MONTPELLIER

En SyAM - Systèmes Automatiques et Micro-électroniques

École doctorale I2S - Information, Structures et Systèmes

Unité de recherche LIRMM

Systèmes intégrés adaptatifs ultra basse consommation pour l'Internet des Objets

Présentée par Guillaume Patrigeon

Le 16/07/2020

Sous la direction de Pascal Benoit

Devant le jury composé de

Jean-Michel PORTAL	Professeur à l'Université Aix-Marseille	Rapporteur
Sébastien PILLEMENT	Professeur à l'Université de Nantes	Rapporteur
Tanguy RISSET	Professeur à l'INSA Lyon	Examinateur
Laurent LATORRE	Professeur à l'Université de Montpellier	Examinateur
Lionel TORRES	Professeur à l'Université de Montpellier	Examinateur
Pascal BENOIT	Maître de Conférences à l'Université de Montpellier	Directeur de thèse

UNIVERSITÉ DE MONTPELLIER

Systèmes intégrés adaptatifs ultra basse consommation pour l'Internet des Objets

Table des matières

Table des matières V
Liste des figures VII
Liste des tableaux IX
1. Introduction générale 1
1.1. Contexte 1
1.2. Objectif de la thèse 3
1.3. Projet de recherche en lien avec la thèse 3
2. Contexte, état de l'art et problématique 5
2.1. Internet des Objets, réseaux de capteurs sans fils 5
2.1.1. Définition, domaines d'application et architecture 5
2.1.2. Nœud capteur 10
2.1.3. Stratégies pour l'économie d'énergie 14
2.2. Technologies émergentes 21
2.2.1. FD-SOI 28 nm 21
2.2.2. Mémoires non-volatiles 22
2.2.3. Mémoires magnétiques 25
2.3. Évaluation des technologies émergentes pour les nœuds capteurs 33
2.3.1. Évaluation des nœuds capteurs 33
2.3.2. Programmes applicatifs de référence 34
2.3.3. Architectures ultra-basse consommation 36
2.3.4. Évaluation du contrôleur 38
2.4. Conclusion 40
3. Méthodologie d'évaluation de nœuds de capteurs 43
3.1. Conception d'une architecture de microcontrôleur 44
3.1.1. Spécifications 45
3.1.2. Architecture MASTA 50
3.1.3. Interface logicielle 58
3.2. Environnement de validation 62
3.2.1. Plateforme de prototypage 63
3.2.2. Prototypage sur FPGA 66
3.2.3. Utilisation des ressources 69

3.3. Implémentation en technologie FD-SOI 28 nm 71
3.3.1. Cellules propres à la technologie 72
3.3.2. Réalisation du circuit 77
3.4. Conclusion 81
4. Exploration et résultats 83
4.1. Validation 83
4.1.1. Microcontrôleur de référence 84
4.1.2. Différences entre les architectures 85
4.1.3. Résultats 87
4.2. Hybridation des mémoires principales 88
4.2.1. Modification de la hiérarchie mémoire 88
4.2.2. Étude de cas 89
4.2.3. Évaluation 91
4.3. Simulation du microcontrôleur FD-SOI 28 nm 95
4.3.1. Extraction de l'activité du circuit 95
4.3.2. Estimation 95
4.3.3. Observations 97
4.4. Conclusion 98
5. Conclusion et perspectives 101
Bibliographie 105
Liste des publications 114

Liste des figures

Figure 1-1 : Historique et projection de 2019 du nombre d'objects connectés dans le monde **[1]**.... 1

Figure 2-1 : Représentation de l'architecture typique d'un réseau de capteurs sans fil..... 8

Figure 2-2 : (a) architecture typique d'un microcontrôleur (simplifié) ; (b) architecture typique d'un microporcesseur à quatre cœurs et deux niveaux de cache (simplifié)..... 9

Figure 2-3 : Composition d'un nœud..... 11

Figure 2-4 : Représentation du comportement le plus simple d'un nœud capteur..... 13

Figure 2-5 : Exemple de stratégie d'économie d'énergie par la réduction du nombre de messages envoyés..... 15

Figure 2-6 : Stratégies de réduction de l'énergie - (a) mise en veille simple ; (b) mise hors tension... 17

Figure 2-7 : Stratégie de réduction d'énergie - Système normalement éteint..... 18

Figure 2-8 : Représentation d'une bascule hybride **[27]**..... 19

Figure 2-9 : (a) transistor Flash à grille flottante **[34]** **[35]** ; (b) programmation de la mémoire Flash ; (c) effacement de la mémoire Flash..... 23

Figure 2-10 : (a) composant mémoire ferroélectrique ; (b) transistor ferroélectrique (FeFET)..... 24

Figure 2-11 : Composant mémoire à changement de phase (PCM)..... 25

Figure 2-12 : Perturbation par un champ magnétique des électrons traversant un matériau ferromagnétique, selon leur spin..... 25

Figure 2-13 : Magnétorésistance géante (GMR) - (a) état antiparallèle (AP) ; (b) état parallèle (P). 26

Figure 2-14 : Schéma simplifié d'une jonction magnétique à effet tunnel (MTJ)..... 27

Figure 2-15 : Cellule mémoire composée d'une MTJ et de son transistor d'accès ; la ligne rouge indique le chemin du courant de lecture **[49]**..... 27

Figure 2-16 : (a) magnétisation de la couche libre à l'aide de champs magnétiques externes **[49]** ; (b) cycle d'hystérésis représentant le comportement de la MTJ..... 28

Figure 2-17 : « *Toggle MTJ* »..... 28

Figure 2-18 : Séquence pour faire basculer l'état d'une MTJ de type « toggle », vue de dessus **[50]** ; les lignes verticales et horizontales représentent les lignes de métal générant les champs magnétiques, les flèches rouges symbolisent le passage du courant, les autres flèches la magnétisation des couches libres..... 29

Figure 2-19 : Séquence d'écriture d'une MTJ avec assistance thermique (TAS) **[51]** ; (a) chauffage ; (b) écriture ; (c) refroidissement..... 30

Figure 2-20 : (a) SOT-MTJ ; (b) chemin de lecture ; (c) chemins d'écriture **[54]**..... 31

Figure 3-1 : Flot d'évaluation..... 44

Figure 3-2 : Architecture du microcontrôleur 51
Figure 3-3 : Architecture du moniteur d'activité 54
Figure 3-4 : Interface esclave type AHB-Lite 55
Figure 3-5 : Chronogramme d'un transfert - Cycle (1) : phase d'adresse A ; cycle (2) : phase de donnée A et phase d'adresse B ; cycle (3) : phase de donnée B 57
Figure 3-6 : Comportement du programme d'amorçage 59
Figure 3-7 : Flot d'évaluation - Implémentation sur FPGA 62
Figure 3-8 : Schéma de la plateforme de prototypage de nœud capteur 63
Figure 3-9 : La première version du FlexNode, avec au centre le Cmod A7 64
Figure 3-10 : La seconde version du FlexNode, intégrant les périphériques 65
Figure 3-11 : (a) Exemple de circuit avec un signal d'activation d'horloge (*enable*) ; (b) Représentation de la bascule avec signal d'activation utilisée dans (a) 68
Figure 3-12 : (a) Exemple de circuit avec un signal d'horloge commandé par porte ; (b) Représentation de la cellule utilisée pour la commande du signal d'horloge du circuit (a) 68
Figure 3-13 : Schéma fonctionnel du microcontrôleur validé sur FPGA 70
Figure 3-14 : Flot d'évaluation - Modélisation 72
Figure 3-15 : Schéma fonctionnel du microcontrôleur implémenté 77
Figure 3-16 : Représentation du microcontrôleur avant son intégration dans le circuit final 80
Figure 3-17 : Circuit complet en technologie FD-SOI 28 nm 81
Figure 4-1 : Carte d'évaluation STM32F072 Nucleo-64 84
Figure 4-2 : Schéma fonctionnel simplifié de la STM32F072RBT6 85
Figure 4-3 : Représentation de l'architecture du système pour les quatre configurations 90
Figure 4-4 : Partie dynamique de l'énergie consommée par les mémoires durant la phase active 93
Figure 4-5 : Énergie perdue par les mémoires durant la phase de sommeil 94
Figure 4-6 : Estimation de la consommation d'énergie obtenue par simulation 96

Liste des tableaux

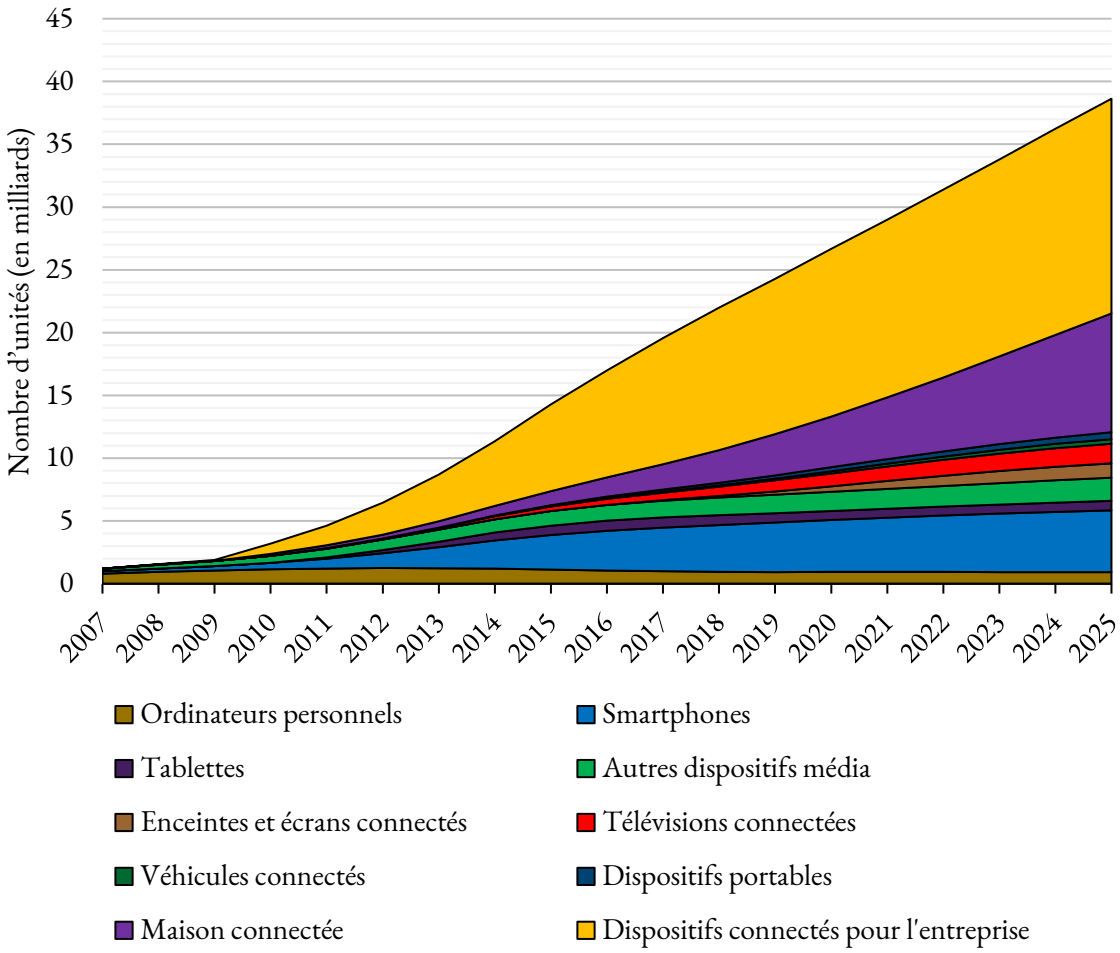
Tableau 2-1 : Comparaison des différentes technologies de mémoire magnétique [55] 32
Tableau 2-2 : Comparaison des différentes technologies mémoires [56] 33
Tableau 2-3 : Caractéristiques de microncontrôleurs ultra-basse consommation intégrant des technologies émergentes 37
Tableau 3-1 : Architecture mémoire des microcontrôleurs 32 bits très basse consommation du commerce 47
Tableau 3-2 : Architecture mémoire de prototypes de microcontrôleurs 32 bits très basse consommation 48
Tableau 3-3 : Spécifications de l'architecture MASTA 50
Tableau 3-4 : Organisation mémoire du moniteur d'activité 61
Tableau 3-5 : Registre de contrôle (CR, *Control Register*) du moniteur d'activité 61
Tableau 3-6 : Resources du FPGA utilisé par le microcontrôleur 71
Tableau 3-7 : Caractérisiques de la banque STT-MRAM de SPINTEC 74
Tableau 3-8 : Configuration des contrôleurs d'entrées/sorties 75
Tableau 3-9 : Caractéristiques des implémentations FPGA et ASIC du microcontrôleur 76
Tableau 4-1 : Comparaison des caractéristiques et des résultats obtenus entre le microcontrôleur implémenté sur FPGA et le STM32F072 Nucleo-64 88
Tableau 4-2 : Coût énergétique par opération pour chaque mémoire 91
Tableau 4-3 : Détails des opérations mémoires lors de la phase active de l'application de référence 92

1. Introduction générale

1.1. Contexte

38,6 milliards, c'est le nombre d'objets connectés dans le monde en 2025 selon les prédictions de Strategy Analytics réalisées en mai 2019 [1]. De plus en plus de circuits électroniques et micro-électroniques prennent place dans notre quotidien et leur nombre ne cesse d'augmenter (Figure 1-1).

Figure 1-1 : Historique et projection de 2019 du nombre d'objects connectés dans le monde [1]



Si leur nombre augmente, la taille de ces dispositifs quant à elle tend à diminuer, afin qu'ils soient moins intrusifs et plus facilement intégrables dans leur milieu d'évolution, grâce aux avancées technologiques réalisées ces dernières années. Mais cette réduction de la taille amène à la réduction d'une partie de ressources disponibles sur ces produits, et plus particulièrement l'énergie, point très critique pour les applications de l'Internet des Objets. En parallèle, les circuits embarqués deviennent de plus en plus complexes, afin d'y intégrer plus de sécurité, plus de fonctionnalités, et de les rendre

plus fiables. Cette augmentation de la complexité se traduit par une augmentation de la capacité à traiter l'information, et par conséquent un besoin en énergie plus grand. La technologie CMOS a toutefois évolué elle aussi, permettant de répondre à cette augmentation de la complexité tout en étant plus efficace énergétiquement. Cependant, cette amélioration progressive n'est pas éternelle et des limites de la technologie semi-conducteur ont été atteintes : les performances des processeurs augmentent difficilement depuis quelques années, si bien qu'ils sont mis en parallèle pour gagner en performances de calcul ; les effets parasites comme la résistance et l'effet capacitif des lignes de métal sont trop importants ; la densité de puissance augmente ; le nombre de défauts lors de la fabrication est plus important. Le développement des circuits intégrés est alors ralenti : pour réduire la proportion de circuits défaillants en sortie de production, le nombre de règles de conception augmente, allongeant alors le temps nécessaire à leur vérification.

La technologie CMOS traditionnelle n'est alors plus suffisante pour répondre aux contraintes posées par une grande partie des applications. Des solutions alternatives sont alors envisagées pour compléter ou remplacer la technologie traditionnelle et permettre la conception de circuits intégrés toujours plus rapides, plus petits, plus fiables, plus efficaces énergétiquement et moins coûteux. Deux points clefs sont ciblés : les éléments de mémorisation et la technologie transistor.

Différents types d'éléments dédiés à l'entreposage des données sont utilisés dans la hiérarchie mémoire des systèmes sur puces embarqués. Des registres du processeur et de ses périphériques aux mémoires principales, différentes solutions sont aujourd'hui utilisées selon les besoins exprimés en tous points de cette hiérarchie : temps de lecture, temps d'écriture, énergie nécessaire pour chaque opération, débit maximum, énergie statique, capacité, encombrement, non-volatilité, accès direct, procédé de fabrication, coût. Chaque solution présente des caractéristiques différentes, avec ses points forts mais également des contraintes. Ainsi, la technologie mémoire utilisée pour entreposer un programme applicatif n'est pas la même que celle utilisée pour contenir l'état du processeur. Aucune des solutions actuelles ne pouvant idéalement remplacer chaque élément de mémorisation d'un système sur puce sans contraintes, c'est un assemblage de différentes technologies qui permet d'obtenir une hiérarchie mémoire optimisée, mais certes pas encore idéale.

Afin d'améliorer l'efficacité énergétique de ces circuits, des stratégies d'économie d'énergie sont mises en place pour réduire leur consommation en courant, plus particulièrement en limitant ou en supprimant les pertes dues à des éléments internes ou à des parties de ces derniers lorsqu'ils sont inactifs, et ce grâce à des techniques de régulation dynamique de l'alimentation et de la performance des éléments concernés (DVFS, *Dynamic Voltage and Frequency Scaling*). Mais plus la stratégie est agressive en terme d'économie d'énergie, plus elle est contraignante : augmentation de la complexité des circuits, de leur temps de conception et de réalisation ; temps de réaction plus long face à un événement selon le mode d'économie d'énergie utilisé ; gestion des différentes stratégies pas toujours évidente ; temps de développement plus long...

L'utilisation de solutions alternatives est nécessaire pour outrepasser les limites des technologies actuelles. Si différentes options sont envisagées, l'intégration d'une technologie mémoire magnétique dans les systèmes embarqués est une solution prometteuse qui apporterait une nouvelle façon de gérer de l'énergie pour les applications de type Internet des Objets.

1.2. Objectif de la thèse

De nouvelles stratégies et solutions techniques sont proposées pour améliorer l'efficacité énergétique des dispositifs sans fils utilisés pour de nombreuses applications de l'Internet des Objets ; les solutions traditionnelles n'étant plus suffisantes pour une partie des applications. La combinaison de deux technologies sera à l'étude au cours de cette thèse : la technologie FD-SOI (silicium sur isolant) et les mémoires magnétiques, plus particulièrement de type STT. Le premier objectif de la thèse est de définir et mettre en place les outils nécessaires à l'évaluation de ces nouvelles solutions intégrées dans les microcontrôleurs, et de définir une méthodologie d'évaluation adaptée et tenant compte du contexte applicatif. Les technologies mémoires non-volatiles permettent de mettre en place de nouvelles stratégies pour la gestion de l'énergie, en particulier la réalisation de circuits dits normalement éteints ; le second objectif de la thèse est la mise en place et l'adaptation de ces stratégies grâce à ces technologies émergentes, d'optimiser l'intégration et l'utilisation de celles-ci, et de les évaluer dans le but de qualifier leurs apports et leurs limitations pour les applications embarquées.

1.3. Projet de recherche en lien avec la thèse

Cette thèse a été menée dans le cadre du projet de recherche collaborative internationale intitulé MASTA (*MRAM Based Design, Test and Reliability for Ultra-Low-Power SoC*, conception, test et fiabilisation d'architectures à base de mémoires magnétiques pour les systèmes sur puce à très faible puissance), financé par l'Agence Nationale de la Recherche (bourse ANR-15-CE24-003-01). Ce projet fait intervenir trois laboratoires de recherche : l'Institut Technologique de Karlsruhe (KIT, www.kit.edu), situé à Karlsruhe, en Allemagne ; SPINTEC (www.spintec.fr) situé à Grenoble, en France ; et le LIRMM.

L'objectif de ce projet de recherche est l'investigation, la conception, le développement et l'analyse d'architectures hybrides pour les calculateurs dits normalement éteints où les mémoires magnétiques sont utilisées dans l'ensemble de la hiérarchie mémoire (registres, caches, mémoires principales...) en complément de la technologie CMOS traditionnelle (www.lirmm.fr/masta).

2. Contexte, état de l'art et problématique

2.1. Internet des Objets, réseaux de capteurs sans fils

Depuis la création des premiers réseaux informatiques à grande échelle dans les années 1960, qui après plusieurs décennies d'évolution, a permis d'obtenir le réseau de communication mondial « internet » tel qu'on le connaît aujourd'hui, l'interconnexion de capteurs et de détecteurs à ces réseaux pour la récupération d'informations à distance a toujours suscité un grand intérêt. Initialement filaires, l'utilisation de réseaux sans fils pour ce type de dispositifs a rapidement été motivée par des frais de déploiement et de maintenance élevés [2]. Il faudra toutefois attendre la toute fin des années 1990 avant de voir les premiers réseaux de capteurs sans fil connectés aux réseaux informatiques conventionnels, en particulier grâce aux projets Smart Dust [3], [4] et WINS (*Wireless Integrated Network Sensors*) [5]. Dès les débuts du projet Smart Dust, les plus grandes difficultés rencontrées étaient liées à l'énergie, et les participants au projet ont mis en évidence l'importance de l'efficacité énergétique des systèmes intégrés, jusqu'alors mise au second plan par les concepteurs en faveur des performances seules de ces systèmes. L'utilisation de récupérateurs d'énergie, en particulier des cellules photovoltaïques, était déjà envisagée pour répondre au problème de la densité énergétique du produit [3]. Aujourd'hui, plusieurs milliards de dispositifs sans fil sont connectés à internet et peuvent communiquer entre eux ou avec d'autres composants du réseau mondial, dans le cadre de ce qu'on appelle l'Internet des Objets. Si les différentes technologies utilisées pour réaliser ces dispositifs ont évolué au fil des années, la principale source de contraintes des premiers réseaux de capteurs sans fil, l'énergie, reste encore aujourd'hui le problème majeur. C'est cette question de l'optimisation de l'efficacité énergétique des systèmes sur puce très basse consommation pour l'Internet des Objets qui nous a motivé à réaliser ce projet.

2.1.1. Définition, domaines d'application et architecture

L'Internet des Objets est défini par l'Union Internationale des Télécommunication (UIT, ou ITU pour *International Telecommunication Union*) comme une infrastructure mondiale pour la société d'information interconnectant des objets, qu'ils soient physiques ou virtuels, grâce aux technologies de l'information et de la communication, pour disposer de services évolués [6]. Les objets sont dotés de dispositifs capables d'agir (actionneurs), de saisir une information (capteurs), de la traiter et/ou de la transmettre. Les technologies mises en œuvre, les supports et protocoles de communications, les types de dispositifs utilisés et leur comportement sont choisis pour répondre aux services et aux contraintes imposées par une application donnée.

Au-delà de l'Internet des Objets, dans le cas de déploiement de dispositifs sans fils servant essentiellement à la saisie d'informations relatives à un objet ou à un milieu, on parle plus

spécifiquement de réseau de capteurs sans fils (WSN, *Wireless Sensors Network*), où ces dispositifs sont appelés « nœuds capteur ». Un réseau de capteurs sans fil n'est pas nécessairement connecté au niveau mondial ni au niveau régional. Quel que soit l'échelle de ces réseaux, l'utilisation de telles infrastructures est présente dans de nombreux domaines, pour une très grande diversité d'application :

- dans la sécurité civile, pour la prévention et la gestion des catastrophes, d'origines naturelles ou humaines, la surveillance des personnes à risques ou encore la localisation de victimes après une catastrophe [7] ;
- dans l'agriculture, pour le suivi des événements météorologiques, la bonne utilisation et l'optimisation des ressources, le contrôle de maturité, la détection de maladies ou d'invasion de parasites et de nuisibles ; dans l'élevage, pour le suivi des animaux domestiques, de leur santé, la détection de parturition et de ponte, la détection d'attaque de prédateurs [8] ;
- dans les villes (mais aussi bien au-delà), pour la gestion et l'optimisation du trafic urbain, des emplacements de stationnement, de l'acheminement et du traitement des eaux usées et des déchets, de la distribution de l'eau et de l'énergie, des services de livraison, des transports en commun, de la navigation des véhicules d'urgence et de secours, la surveillance des infrastructures routières et des bâtiments [9] ;
- dans la santé, pour le suivi à distance des conditions physiques des personnes, pour la détection de malaises, de chutes, d'évanouissements et autres situations de détresse, pour l'aide au diagnostic, pour la détection et le suivi d'épidémies, pour la détection de la contamination de l'eau et des produits comestibles, pour le suivi de la chaîne de froid des médicaments, des aliments et autres produits sensibles à la température [10] ;
- dans l'industrie, pour l'identification, la localisation, le suivi et la protection des objets, des outils, des marchandises et des personnes, pour l'optimisation des ressources, pour le diagnostic, la prise de décision et la reconfiguration automatisés, pour l'optimisation de la production en général [11] ;
- dans le secteur militaire, pour la détection, la localisation et le suivi de troupes, de véhicules de missiles et de drones, pour la détection de substances chimiques, d'armes biologiques et explosives, pour des services de navigation et de guidage [12].

On constate d'après cette liste non-exhaustive que les applications usant de réseaux de capteurs sans fil sont très nombreuses et ont des objectifs très différents. Les solutions techniques utilisées le sont aussi, si bien qu'il n'y a pas de solution générale couvrant tous les cas. Parmi les contraintes majeures, on retrouve :

- la taille des produits ;

- la capacité de transport des données, le débit, la latence, le taux d'échec et l'encombrement des réseaux de communication, quel que soit le type ;
- les performances de calcul et de traitement des données ;
- l'entreposage des données ;
- la sécurité des données, des biens et des personnes ;
- la fiabilité de l'application, dépendante de celle des nœuds et des autres acteurs ;
- l'autonomie, en terme de services et de gestion ;
- la flexibilité de l'architecture quant à son évolution, comme l'ajout de nœuds sur le réseau et l'ajout de nouveaux services ;
- le bilan énergétique, et dans le cas des produits sur récupérateur d'énergie et/ou sur accumulateurs d'énergie, l'autonomie en énergie.

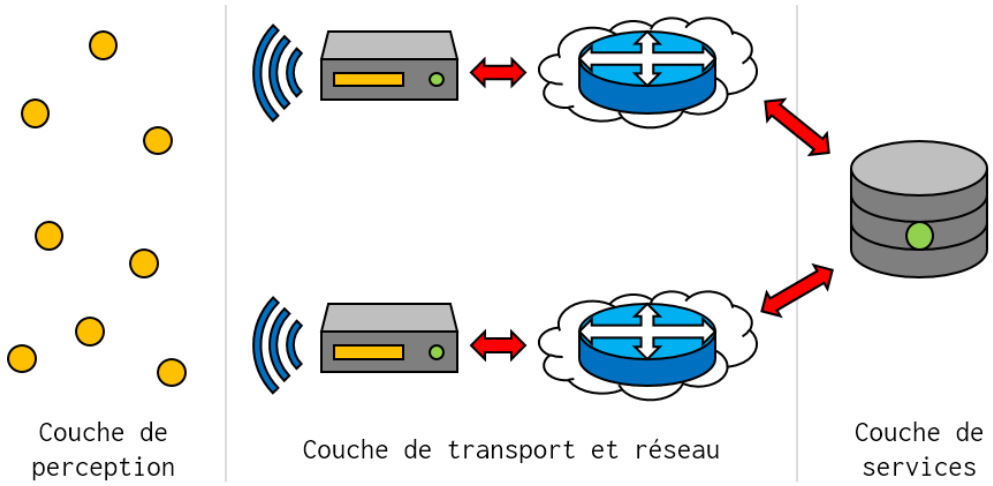
Ce sont les spécifications de l'application et les contraintes techniques qu'elle impose qui vont guider le choix vers la solution la plus adaptée à la situation, voir adapter une solution existante et l'optimiser pour l'application.

S'il n'y a pas de solution unique pour couvrir tous les cas, on retrouve toutefois, dans un grand nombre d'applications impliquant des réseaux de capteurs sans fil, une architecture typique, illustrée par la Figure 2-1, qui peut être décomposée en trois couches :

- la couche de perception, composée de nœuds au contact des objets ou de l'environnement à suivre ;
- la couche de transport et de réseau, assurant la circulation des informations entre les différents acteurs du réseau, parfois capable de réaliser un traitement et/ou un entreposage local des données ;
- la couche de services, réalisant l'entreposage, le traitement des données et la prise de décision.

Un réseau de capteurs sans fil fait intervenir de nombreux acteurs. Du nœud capteur qui récupère l'information jusqu'au serveur qui la transforme, en passant par les passerelles (*gateways*) et l'infrastructure réseau qui la transporte, chaque élément de la chaîne a une fonction qui lui est propre, avec son lot de contraintes techniques plus ou moins fortes. En effet, les éléments constituant la couche de services sont typiquement soumis à une charge de travail assez importante, afin de traiter le flot de données, plus ou moins important selon l'application, qui remonte jusqu'à eux. Les solutions techniques utilisées pour cette couche sont plutôt orientées performances de calcul et grande capacité d'entreposage des données. Les serveurs dédiés pour la gestion des services sont généralement placés dans des salles prévues, disposant d'une source d'énergie continue. Les éléments de la couche de transport n'ont pas toujours besoin d'une performance de calcul aussi grande que ceux de la couche de services, mais un traitement et un entreposage local des informations au niveau des passerelles est possible ; cela doit être pris en compte au moment du choix de la solution. Si la

Figure 2-1 : Représentation de l'architecture typique d'un réseau de capteurs sans fil



plupart des éléments réalisant la fonction réseau disposent d'une source d'énergie continue, les passerelles sont dans certains cas alimentées par batterie, et parfois associées à un récupérateur d'énergie (par exemple, des cellules photovoltaïques).

Quant aux éléments de la couche de perception, les nœuds capteurs, ceux-ci ne disposent pas d'une source d'énergie continue (dans le cadre d'un réseau de capteurs sans fil), et doivent nécessairement recourir à l'utilisation d'un ou plusieurs accumulateurs d'énergie (pile, batterie, super condensateur...) et parfois d'un ou plusieurs récupérateurs d'énergie. Cette contrainte technique impose une puissance consommée adaptée et une bonne gestion de l'énergie afin d'assurer une durée de vie longue du produit, et ainsi éviter des maintenances coûteuses trop régulières qui feraient rapidement grimper le coût total de l'application. Pour certaines d'entre-elles, notamment lorsqu'un dispositif doit être placé au près, sur ou dans un être vivant, humain comme animal, on demande également à ce que ces nœuds capteurs soient de petite taille, afin d'être le moins intrusif et/ou le plus discret possible. Or, cette dernière contrainte limite la taille des accumulateurs d'énergie et des récupérateurs d'énergie, ce qui rend la gestion de l'énergie encore plus critique. Concernant la puissance de calcul, celle-ci n'a pas besoin d'être très grande puisque la charge de travail d'un nœud capteur reste pour la plupart des applications très légère par rapport aux autres éléments de la chaîne. Cependant, cette charge de travail varie beaucoup d'une application à une autre, même à cette échelle, et va grandement impacter le choix du contrôleur et impacter la consommation en énergie du produit final.

Il est important de noter que la sécurité dans les applications de l'Internet des Objets suscite de plus en plus d'intérêt [13], et que les méthodes de sécurisation des données, qu'elles soient matérielles ou logicielles, nécessitent l'utilisation de ressources supplémentaires (calcul, capacité mémoire, modules de chiffrement, générateur de nombres aléatoires) et par conséquent plus d'énergie. De

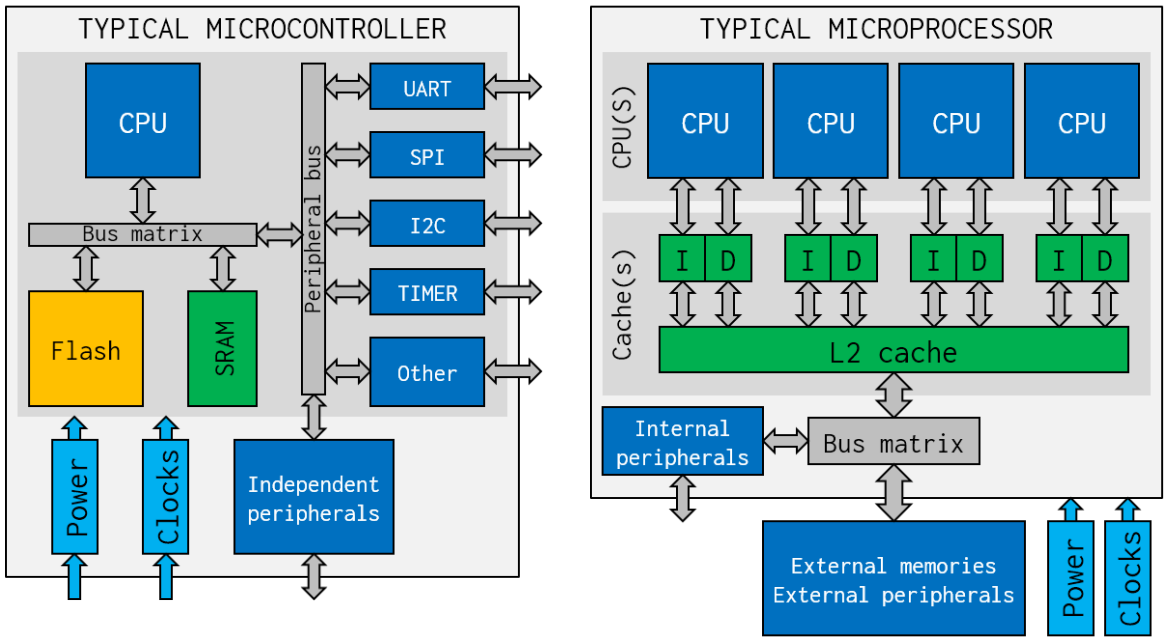
même, certains services gourmands en ressources sont de plus en plus présents en tout point de la chaîne, tel que la mise à jour à distance des produits [14], service utile pour faire évoluer, pour corriger et pour assurer de manière générale une partie de la maintenance d'un produit sans avoir à intervenir physiquement sur celui-ci (opération onéreuse et pas réalisable sur tous les produits).

Si les dispositifs de la couche de perception, bien que très contraints par l'aspect énergétique, ont également besoin de plus de ressources pour intégrer plus de sécurité et de services, les éléments de la couche de services, qui nécessitent de grandes performances de calcul et une grande capacité d'entreposage des données, sont eux aussi soumis à la question de l'énergie, pour des raisons économiques, écologiques et de durée de vie du matériel.

Pour les nœuds capteurs à très basse consommation d'énergie, on préférera prendre des microcontrôleurs comme unité principale : ces derniers sont généralement de petite taille, à faible coût, consommant peut d'énergie et embarquant un maximum de fonctions nécessaires, évitant alors l'ajout de composants supplémentaires qui, la plupart du temps, augmenteraient l'encombrement, le coût et/ou l'énergie consommée du produit final. Comme illustré par la Figure 2-2 (a), la plupart des microcontrôleurs du commerce intègrent un processeur à jeu d'instruction réduit (RISC, *Reduced Instruction Set Computer*), au moins une mémoire non-volatile (utilisée principalement pour contenir le programme et les données statiques), une mémoire volatile (utilisée pour les données dynamiques), un régulateur de tension associé à un module de gestion de puissance, un ou plusieurs

Figure 2-2 : (a) architecture typique d'un microcontrôleur (simplifié) ;
(b) architecture typique d'un microporcesseur à quatre cœurs et deux niveaux de cache (simplifié)

(a) (b)



oscillateurs pour générer des signaux d'horloge, des contrôleurs d'entrées/sorties, des modules de communication, et parfois des périphériques pour le traitement de signaux analogiques. Puisqu'il n'y a actuellement pas une seule solution adaptée à l'ensemble des applications existantes, les fabricants offrent une très large gamme de microcontrôleurs avec des options bien différentes (type et puissance du processeur, périphériques, options d'économie d'énergie, type et capacité mémoire...) pour répondre aux besoins du marché ; la Figure 2-2 (a) ne présente qu'une version simple de microcontrôleur.

Pour la couche de services, on préférera l'utilisation de contrôleurs de type micro-processeurs pour leurs performances de calculs, voir l'utilisation d'architectures dédiées ou reconfigurables, afin d'accélérer les calculs et les mouvements de données [15]. Illustrés par la Figure 2-2 (b), les micro-processeurs peuvent contenir un ou plusieurs processeurs, des antémémoires (aussi appelées mémoires caches, elles permettent d'accélérer l'accès aux données en limitant les mouvements de celles-ci, économisant ainsi du temps et de l'énergie) et quelques périphériques internes. Contrairement aux microcontrôleurs, les micro-processeurs n'intègrent typiquement pas les éléments de régulation de puissance ou de génération de signaux d'horloges, ces derniers étant externes, comme les mémoires principales (non-volatiles et volatiles) et une grande partie des périphériques.

La grande diversité des applications amène à une grande diversité des solutions ; toutes sont soumises à la question de l'énergie ; toutefois nous nous intéressons plus particulièrement aux applications où la contrainte énergétique est la plus forte : les réseaux de capteurs sans fils, et plus particulièrement les nœuds capteurs très basse puissance équipés de microcontrôleurs eux aussi à très faible consommation d'énergie.

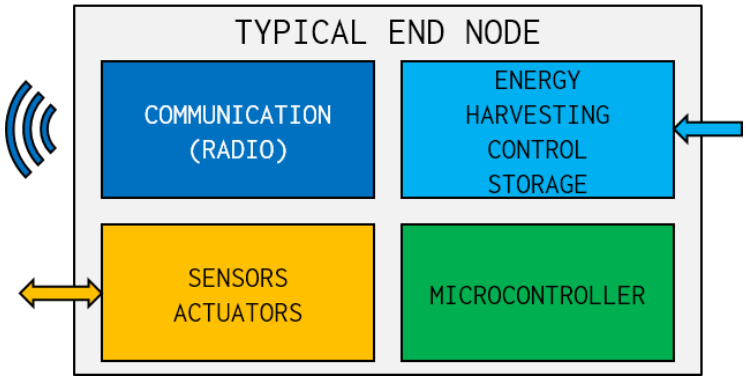
2.1.2. Nœud capteur

a) Composition

Un nœud capteur est un dispositif intégrant plusieurs composants dont le rôle principal est de convertir une ou plusieurs informations relatives à un objet en informations numériques, de les mettre en forme, éventuellement de réaliser un traitement sur celles-ci, puis de les transmettre. Il dispose typiquement d'un microcontrôleur, d'un ou plusieurs modules radio (au moins un émetteur ; la présence de récepteurs est optionnelle) et peut être équipé d'un ou plusieurs capteurs (et d'actionneurs dans les cas particuliers), d'un ou plusieurs accumulateurs d'énergie et dans certains cas d'un ou plusieurs récupérateurs d'énergie (Figure 2-3). La quantité et la nature de chacun de ces composants sont définies dans les spécifications de l'application.

Différents types d'accumulateurs d'énergie électriques sont utilisés pour alimenter les nœuds capteurs. Ils sont caractérisés par la technologie utilisée, leur tension nominale, la quantité de charge (et par extension la quantité d'énergie) qu'ils peuvent accumuler, le débit de charge (si rechargeable)

Figure 2-3 : Composition d'un nœud



et de décharge maximale, leur impédance interne, leur encombrement, leur capacité à être rechargés, leur dégradation en fonction de leur utilisation, leur perte naturelle d'énergie, et bien d'autres métriques. Plusieurs accumulateurs d'un même type peuvent être regroupés sous forme de batteries pour ajuster une partie de leurs caractéristiques (par exemple, la tension, la capacité en énergie, le débit maximum...) afin de les faire correspondre aux besoins de l'application. Le choix de l'accumulateur est très important car il est en relation directe avec la durée de vie du nœud. Il est également très dépendant de l'environnement dans lequel il va évoluer : les caractéristiques de l'accumulateur sont fonction des conditions de température et de pression. Leur tension, leur capacité, leur débit maximal, leur impédance interne, leur courant de fuite naturel varient au cours de leur existence ; ces variations doivent être prises en compte lors de l'estimation de l'autonomie du nœud capteur. L'utilisation de récupérateurs d'énergie et le remplacement de l'accumulateur ne sont pas toujours possibles ; il faut dans ces cas prévoir une solution capable de tenir suffisamment longtemps pour respecter les consignes de durée de vie du nœud capteur. Même si la récupération d'énergie ou le remplacement de l'accumulateur sont réalisables, la mise en place de ces solutions est parfois trop complexe pour être retenue.

b) Comportement

L'énergie disponible sur un nœud capteur est limitée, et de nombreuses contraintes vont restreindre l'utilisation d'une partie des solutions techniques actuellement disponibles. La gestion de l'énergie est un point critique de l'application : il est nécessaire d'adapter le comportement du nœud et de mettre en place une stratégie pour optimiser la consommation d'énergie afin de garantir la durée de vie de l'application à moindre coût.

La mission principale d'un nœud capteur est de mesurer une grandeur physique et/ou de capturer un événement relatif à un objet, un environnement, afin de transmettre cette information. En fonction de la nature de l'information à relever et des besoins de l'application, le nœud capteur peut

avoir un comportement bien différent. Beaucoup d'applications n'ont pas besoin de réaliser un relevé d'information en continu, soit parce que l'information à relever possède une certaine inertie (par exemple, la température ambiante) et un échantillonnage trop fréquent générerait une redondance, soit parce que les besoins de l'application n'imposent pas un temps de réaction immédiat ou une fréquence d'échantillonnage élevée. Pour une partie des applications, le nœud capteur n'a besoin de travailler que dans une plage horaire spécifique, ou sur demande explicite d'une autre entité.

Le comportement du nœud capteur peut être dans la plupart des cas découpé en plusieurs phases : celles où il est actif, appelées phases d'activité ; et celles où il est en attente d'un événement (synchrone, comme le signal d'une horloge, ou asynchrone), appelées phases de sommeil. Durant les phases d'activité, il est amené à mesurer une grandeur physique et/ou capturer en événement, transformer l'information, la mettre en forme, l'entreposer et/ou la transmettre [16]. Ces fonctions constituent typiquement les tâches principales d'un nœud capteur, mais d'autres services peuvent également être implémentés, par exemple : la mise à jour du programme applicatif ; le diagnostic du système ; une mise en veille prolongée...

La Figure 2-4 présente le comportement le plus simple d'un nœud capteur au cours du temps, décomposé en six étapes, dont cinq réalisées durant la phase active : réveil (1) ; mesure d'une ou plusieurs grandeurs physiques et/ou capture d'événements (2) ; transformation, mise en forme et/ou entreposage de l'information (3) ; transmission de l'information (4) ; préparation à la phase de sommeil (5) ; et enfin la phase de sommeil (6).

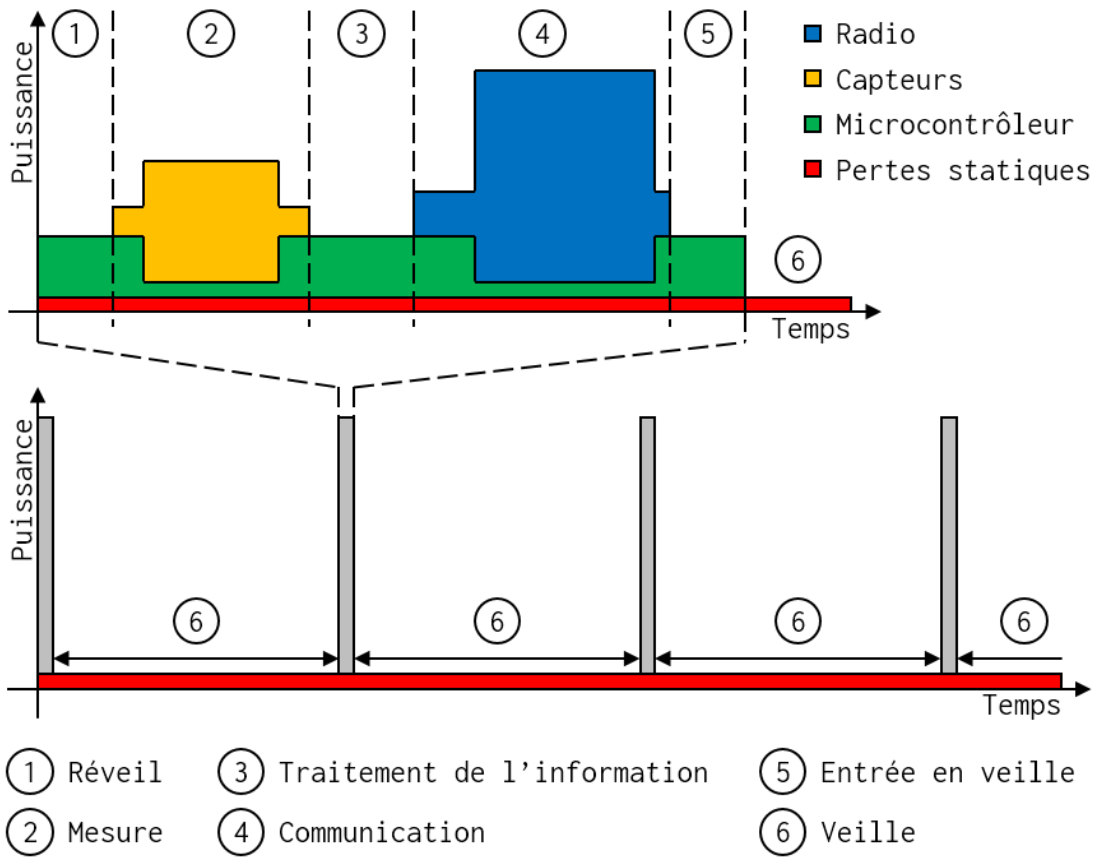
L'utilisation d'un périphérique externe au microcontrôleur, par exemple un capteur ou un module de communication radio, peut également nécessiter plusieurs étapes nécessaires pour réaliser la fonction voulue avec celui-ci : mise en puissance, configuration, opération, récupération de données, arrêt... La gestion de ces composants est influencée par la stratégie de gestion de l'énergie.

Le réveil du nœud peut être synchrone, c'est-à-dire déclenché par un événement dont la fréquence d'apparition est connue. Cet événement est très souvent généré par une horloge dite temps-réel (RTC, *Real Time Clock*), un module pouvant être intégré dans le contrôleur du nœud capteur, assurant son réveil après une période de sommeil définie. Souvent dotés d'une fonction calendrier, les modules RTC peuvent être programmés pour réveiller un nœud à une heure et/ou à une date précise. Ces périphériques sont généralement indépendants, même intégrés au contrôleur ; de ce fait, ils peuvent continuer à opérer si ce dernier est dans un état de sommeil profond (cet état est détaillé dans la partie suivante). L'événement synchrone peut également être généré par un compteur interne au contrôleur, parfois par une composante du réseau si la solution de communication le permet [17].

Le réveil du nœud peut aussi être asynchrone, c'est-à-dire déclenché par un événement dont l'apparition n'est pas contrôlée et ne peut être prédite. Certains capteurs peuvent effectuer une mesure ou une capture en continu et générer un signal de réveil pour sortir le contrôleur de sa phase de sommeil. Ce dernier peut alors lancer une procédure spécifique suite à l'événement déclencheur, par exemple, lancer une alerte sur le réseau ou réaliser des mesures complémentaires et/ou plus précises. Cela permet d'augmenter la réactivité du nœud face à un événement en évitant d'augmenter la fréquence de réveil de celui-ci, permettant dans certains cas d'optimiser sa consommation d'énergie. Comme pour le réveil synchrone, un réveil asynchrone peut être demandé par le réseau.

Dans beaucoup d'applications, les deux types de réveils, synchrone et asynchrone, sont combinés. Cela permet au nœud d'être réactif à un ou plusieurs événements tout en assurant une activité minimale régulière, même si aucun événement n'a été capturé durant la période programmée. Ce comportement est souvent utilisé pour garantir une certaine sûreté de fonctionnement de l'application : si sa tâche périodique implique de communiquer avec le reste du réseau, il est possible de détecter si un nœud est déconnecté de celui-ci. Il peut alors être déclaré comme défaillant, et ce changement d'état est un événement pouvant faire l'objet d'une alerte.

Figure 2-4 : Représentation du comportement le plus simple d'un nœud capteur



Quel que soit l'événement qui réveil le contrôleur, le nœud capteur peut passer une très grande partie de son temps dans sa phase de sommeil. Comme illustré par la Figure 2-4, le nœud consomme toujours un peu de puissance même s'il est inactif. Cette perte de courant, appelée fuite (*leakage*), peut être responsable d'une perte non négligeable de l'énergie disponible sur le nœud et réduire sa durée de vie. La part d'énergie perdue par la consommation statique d'un circuit est même d'autant plus grande que le nœud technologique est petit, et si le rapport temps actif sur temps inactif est suffisamment bas, cette consommation statique peut représenter la perte d'énergie majoritaire [18]. Pour limiter cette perte, différentes stratégies sont utilisées.

2.1.3. Stratégies pour l'économie d'énergie

La gestion de l'énergie disponible sur les nœuds capteurs est l'un des points critiques de l'application. Afin de réduire la consommation d'énergie d'un nœud capteur, plusieurs solutions disponibles sont employées.

a) Réduire la consommation d'énergie des périphériques externes

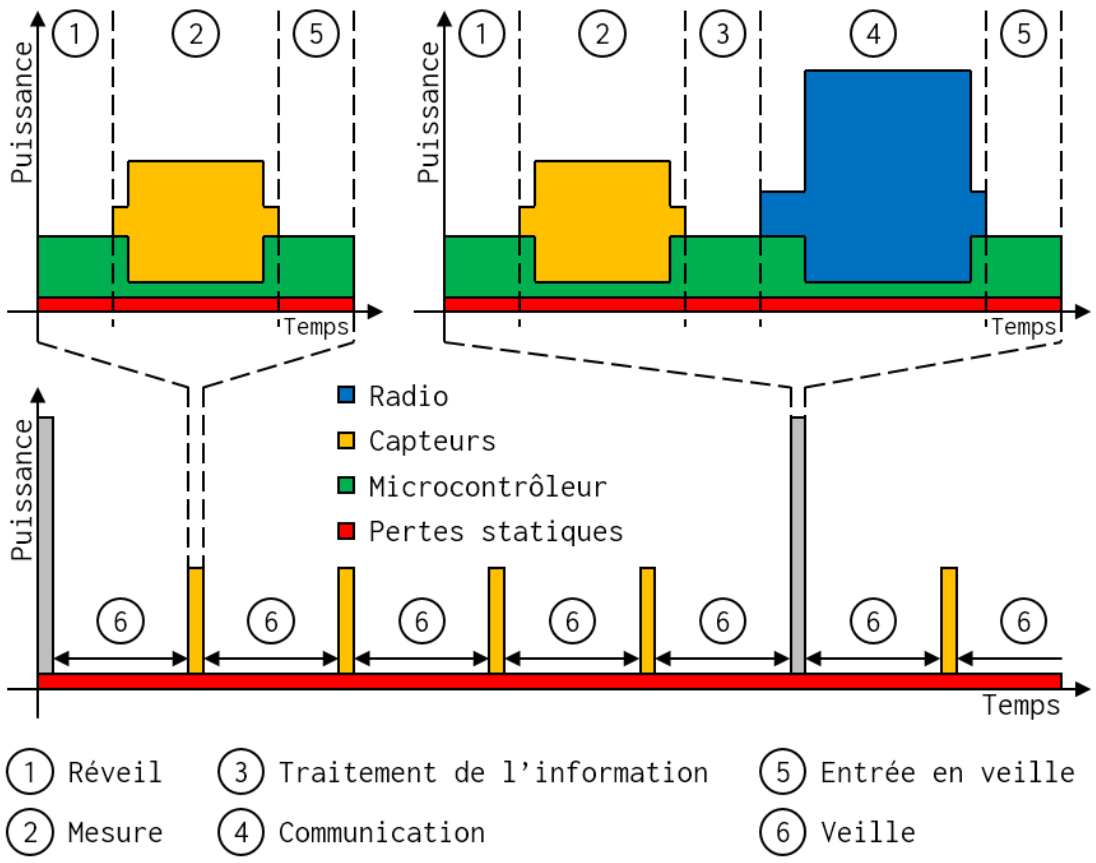
Comme introduit précédemment, le comportement du nœud capteur est divisé en phases d'activité et de sommeil. Durant les phases de sommeil, certains composants du nœud ne sont pas utilisés. S'ils disposent d'un mode d'économie d'énergie, la première solution revient à utiliser ce mode pour réduire le courant résiduel qui traverse ces composants. Au-delà des phases de sommeil, ce mécanisme, ainsi que tous les autres mécanismes suivants peuvent aussi être utilisés durant la phase d'activité lorsqu'un composant n'est pas utilisé.

Pour réduire encore l'énergie qu'ils consomment durant la phase de sommeil, l'alimentation de ces composants inactifs peut, dans une partie des cas et à l'exception du contrôleur principal, être coupée afin de supprimer le courant résiduel qui les traverse, même désactivés [19]. Pour que cette action soit efficace, il est nécessaire que le temps de sommeil soit suffisamment long pour que l'économie d'énergie faite pour chaque composant compense l'énergie nécessaire au redémarrage de celui-ci, l'énergie nécessaire à sa reconfiguration et à la sauvegarde de ses paramètres si besoin, et l'énergie perdue dans le système de gestion de sa ligne d'alimentation. En effet, il est parfois nécessaire d'ajouter des composants, qui consomment une certaine puissance en continu, pour gérer les lignes d'alimentation des autres composants. Bien évidemment, il faut que le gain sur la consommation d'énergie du nœud entier soit significatif pour justifier l'ajout de nouveaux composants, qui ont un coût financier et spatial.

Une autre stratégie consiste à différer des actions, comme un calcul ou le travail d'un composant, afin de réduire leur part de consommation d'énergie [20]. En effet, certaines actions n'ont pas besoin d'être réalisées à chaque réveil du nœud.

Dans la plupart des applications, l'information relevée n'a pas besoin d'être transmise dans l'immédiat aux autres objets du réseau. Il est alors possible de regrouper des données recueillies sur plusieurs minutes, plusieurs heures ou plusieurs jours et de n'envoyer qu'un seul message au réseau comportant l'ensemble des données de cette période, comme illustré par la Figure 2-5. Un message radio a un coût initial en énergie non négligeable à cause des différentes couches de protocole ajoutant chacune des informations nécessaires à l'infrastructure réseau (ces informations sont dépendantes des solutions techniques choisies ; on retrouve dans la plupart des cas un signal de synchronisation, un identifiant, parfois un destinataire, un code de vérification ou de correction d'erreurs...), augmentant ainsi la taille du message, et donc son coût énergétique [21]. Certains protocoles radios imposent également une taille de message minimale et/ou l'utilisation de symboles non sécables pour représenter une taille fixe de données. Envoyer une donnée de petite taille n'est donc pas rentable car le coût énergétique de l'envoi est principalement dû à son encapsulation ; si l'envoi des données peut être différé, il est préférable de réduire le nombre de messages envoyés en les regroupant. Cette méthode permet également de réduire la charge du réseau.

Figure 2-5 : Exemple de stratégie d'économie d'énergie par la réduction du nombre de messages envoyés



b) Modes d'économie d'énergie dans les microcontrôleurs classiques

S'il n'est pas possible de couper directement l'alimentation en puissance du contrôleur, il est possible de réduire l'énergie qu'il consomme lorsqu'il n'est pas utilisé grâce aux différents mécanismes qu'il intègre. Les microcontrôleurs intègrent de nombreux mécanismes plus ou moins agressifs pour répondre au mieux aux différents cas, permettant ainsi aux développeurs d'optimiser l'énergie disponible dans leurs produits (quelques exemples : [22] [23] [24]) :

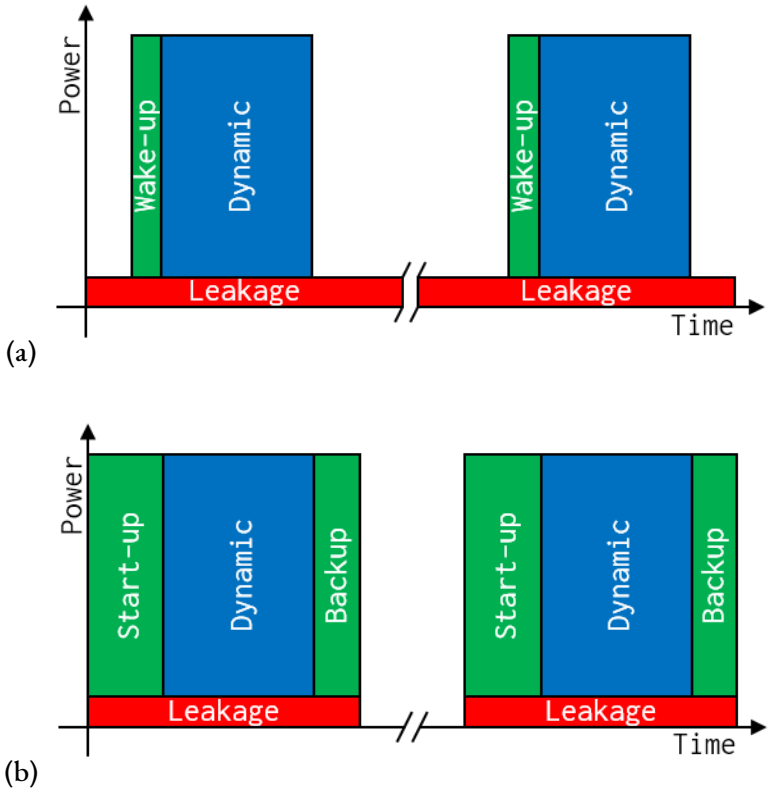
- arrêt du processeur ;
- arrêt des périphériques inutilisés (typiquement l'arrêt du rafraîchissement sur front d'horloge, mais leur alimentation peut aussi être coupée dans certains microcontrôleurs) ;
- arrêt d'un oscillateur externe (lorsqu'implémenté), qui consomme généralement plus qu'un oscillateur interne (typiquement de type RC) ; arrêt des boucles à verrouillage de phase (PLL, *Phase Locked Loop*) et autres synthétiseurs de signaux d'horloge ; arrêt d'un oscillateur interne ; plus généralement, diminution de la fréquence de fonctionnement ;
- réduction de la tension d'alimentation interne en fonctionnement (limite la fréquence maximale et force l'arrêt de certains blocs comme la PLL en dessous d'une certaine tension) ;
- réduction de la tension d'alimentation interne à l'arrêt (mode rétention) ;
- mise en mode rétention des mémoires volatiles (interdisant alors leur utilisation) ;
- arrêt des régulateurs de tension principaux (à l'exception de certains périphériques dédiés notamment pour sortir le microcontrôleur de ce mode d'économie agressif, tous les blocs internes ne sont plus alimentés et les registres perdent leur contenu, de même pour les mémoires volatiles qui ne sont pas en mode rétention, alimentées par leur propre régulateur).

Il est possible d'utiliser plusieurs de ces mécanismes en même temps pour optimiser les gains, même de sélectionner des méthodes différentes en fonction des contraintes de chaque situation d'inactivité que le microcontrôleur peut rencontrer lors du déroulement de l'application (phase d'inactivité très courte, besoin d'une réactivité élevée, nécessité de garder une partie du système actif lors de l'attente d'un périphérique interne ou externe, durant une communication...). Certains mécanismes ne sont pas immédiats et ont un impact sur le fonctionnement du microcontrôleur, sur l'entrée et/ou sur la sortie d'un mode d'économie d'énergie. Les régulateurs de tension ne peuvent pas fournir la tension voulue instantanément après un changement de consigne ; le temps nécessaire pour l'atteindre retarde l'utilisation de composants internes et/ou un changement de la fréquence de fonctionnement (typiquement quelques microsecondes). Le démarrage d'un générateur de signal d'horloge peut prendre du temps s'il est nécessaire d'attendre sa stabilisation (par exemple, un générateur à quartz, une PLL...). Si l'alimentation d'un bloc est coupée, celui-ci doit être reconfiguré après la remise sous tension. De manière générale, plus le mode d'économie d'énergie est agressif, plus le délai avant le rétablissement à un mode normal ou même à un mode moins agressif sera long. Dans

le cas de la perte du contenu des registres ou des mémoires, il est parfois nécessaire de relancer le programme applicatif depuis le début, ou de déplacer ce contenu soit dans une mémoire non-volatile, soit dans une mémoire volatile qui ne perdra pas son contenu lors du changement de mode (gardée fonctionnelle ou placée en mode rétention). C'est pourquoi ces modes doivent être utilisés judicieusement afin que, d'une part, l'énergie dépensée après une étape de sommeil courte, en prenant en compte l'entrée dans ce mode, le retour à l'état voulue et l'énergie dépensé durant le sommeil, ne soit pas plus grande que celle nécessaire à un mode moins agressif, et que, d'autre part, les contraintes de réactivité de l'application soient respectées. Le temps de réactivité requis par les applications communes peut varier de quelques microsecondes à quelques secondes ; et leur fréquence de réveil, qui détermine le temps pouvant être passé en sommeil, peut lui varier de quelques millisecondes à plusieurs heures, voire plusieurs jours [25]. Certaines stratégies ne sont tout simplement pas applicables pour des fréquences de réveil trop élevées.

La Figure 2-6 illustre la différence entre une stratégie de mise en veille simple d'un système (a), réduisant fortement la puissance consommée par celui-ci, et une stratégie de mise hors tension (b), plus agressive quant à la réduction des courants résiduels mais demandant des opérations de démarrage et d'arrêt plus longues.

Figure 2-6 : Stratégies de réduction de l'énergie - (a) mise en veille simple ; (b) mise hors tension

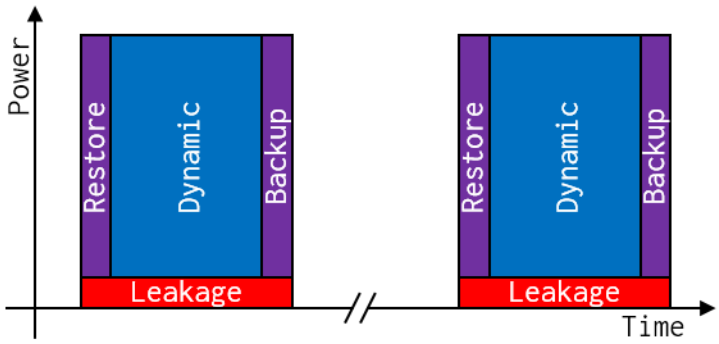


Pour une application dont les réveils sont uniquement synchrones, c'est-à-dire connus et prévisibles, le choix de la stratégie optimale peut être étudiée. Mais dans le cas où des réveils asynchrones sont requis pour l'application, ce choix devient beaucoup plus complexe, car certains événements sont trop imprévisibles pour tenter une adaptation dynamique de la stratégie d'économie d'énergie. La grande variété des mécanismes dédiés à la réduction de la consommation d'énergie, leurs différentes combinaisons possibles et l'incertitude liée au comportement du nœud capteur et des événements extérieurs sont autant de facteurs qui compliquent le choix d'une stratégie d'économie d'énergie pour les développeurs de nœuds capteurs.

c) Systèmes normalement éteints

Afin de pouvoir profiter du gain d'énergie offert en coupant l'alimentation du processeur et des autres composants internes aux microcontrôleurs sans avoir à redémarrer entièrement l'application, il est possible de sauvegarder leur état dans des éléments mémoires prévus à cet effet. Ces mémoires peuvent être de nature volatile et seront alors placées en mode rétention, consommant toujours un peu d'énergie et nécessitant un régulateur de tension dédié ; ou bien non-volatile, permettant une coupure de l'alimentation de tous ces modules. Dans ce dernier cas, on parle de système normalement éteint (*Normally-Off Computing*) car son alimentation est coupée lors de son état stable, la phase de sommeil [20]. À la remise sous tension du processeur et des autres composants internes, l'état précédent la mise hors tension est restauré sans avoir à relancer la totalité du programme applicatif. Cette méthode, illustrée par la Figure 2-7, est dans la plupart des cas plus rapide et plus économe en énergie qu'un redémarrage complet du système [26].

Figure 2-7 : Stratégie de réduction d'énergie - Système normalement éteint



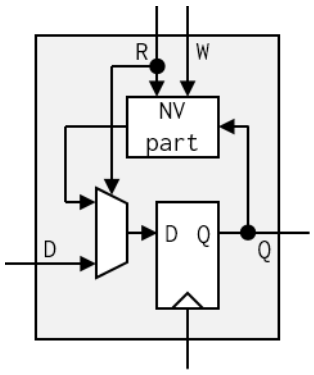
Dans la hiérarchie mémoire du microcontrôleur, plusieurs éléments contiennent l'état du système : les mémoires principales (programme et données), les registres du processeur et les registres des périphériques. La mémoire programme est typiquement de technologie Flash, donc non-volatile. Elle pourrait toutefois être remplacée par une autre technologie mémoire non-volatile. La mémoire

de données est quant à elle volatile (typiquement SRAM), il est donc nécessaire d'utiliser une mémoire non-volatile à la place ou de sauver son contenu dans une mémoire non-volatile, dans la mémoire programme par exemple, mais cette opération de copie prendrait du temps et de l'énergie. Pour la sauvegarde du contenu des registres, ceux du processeur ou ceux des périphériques, cela va dépendre de la solution utilisée et de la pertinence de cette opération, dépendant de la fonction de chaque bascule de ces composants.

Il existe deux méthodologies pour la sauvegarde de l'état d'un système : la sauvegarde implicite, réalisée à chaque changement d'état ; et la sauvegarde explicite, événementielle, demandée soit par le matériel soit par le logiciel. Dans le premier cas, celui de la sauvegarde implicite, des bascules et des verrous purement non-volatiles sont utilisés. Dans le second cas, la sauvegarde explicite, plusieurs solutions sont possibles : l'utilisation de bascules hybrides, le transfert du contenu des bascules dans une matrice mémoire dédiée ou le transfert dans la mémoire principale.

Un verrou ou une bascule non-volatile entrepose directement son contenu dans des éléments non-volatiles (un seul verrou non-volatile typiquement pour la bascule ; l'autre est volatile). Une bascule hybride possède une partie volatile, fonctionnant comme une bascule classique, et une partie non-volatile qui va copier ou restaurer le contenu de la partie volatile sur commande externe (comme précédemment, un seul des deux verrous voit son contenu sauvegardé par la partie non-volatile) [27] [28]. Il existe également des éléments semi-volatiles, c'est-à-dire qui ne maintiennent leur contenu que pour une durée généralement fixée à la conception. Ne permettant pas de sauver des données sur le long terme, ils sont toutefois utilisés pour maintenir un état lors de microcoupures. La Figure 2-8 représente le fonctionnement d'une bascule hybride.

Figure 2-8 : Représentation d'une bascule hybride [27]



Les signaux de commandes d'écriture et de lecture de la partie non-volatile des cellules hybrides, ainsi que la logique générant ces signaux sont mutualisés entre plusieurs bascules afin de réduire la taille et la consommation d'énergie du contrôleur qui leur est dédié. Chacune de ces actions est alors parallélisée entre les bascules mais il est parfois nécessaire de découper la sauvegarde du système complet en plusieurs cycles. En effet, l'écriture d'un trop grand nombre de bascules en parallèle peut

provoquer un pic de consommation de courant trop grand pour les lignes d'alimentations internes. Il est parfois nécessaire de répartir cette charge dans la durée, et le nombre de bascules positionnées sur une même ligne doit être limité lors du placement des cellules dans le circuit [27].

Les cellules non-volatiles sont plus compactes que les cellules hybrides et ne nécessitent pas de circuit de contrôle dédié ; toutefois, selon la solution technologique retenue, les éléments non-volatiles peuvent être responsables d'une performance moindre ou une consommation d'énergie plus élevée que des bascules volatiles classiques, c'est pourquoi la solution hybride est un bon compromis : durant l'écoulement normal du programme, la partie volatile est utilisée. Celle-ci étant comparable à une cellule classique, le processeur obtient des performances similaires à son implémentation volatile ; la partie volatile n'est utilisée que pour sauver et restaurer son état.

Toutes les informations qui décrivent l'état d'un système n'ont pas besoin d'être sauvées pour être disponibles au réveil du microcontrôleur. En effet, certaines d'entre elles sont inutiles au réveil car erronées ou remplacées. Par exemple, les différents étages du processeur sont nettoyés et rechargés au redémarrage pour assurer la cohérence de leur contenu ; la synchronisation avec les opérations de lecture et d'écriture en mémoire doit être faite avant l'entrée en phase de sommeil, l'état du bus et des contrôleurs des banques mémoires est donc réinitialisé ; l'état des registres à décalage des modules de communication sont obsolètes au réveil... Le remplacement des bascules volatiles par leurs homologues hybrides doit dans l'idéal être fait au cas-par-cas afin d'optimiser l'énergie de la sauvegarde et de la restauration de leur état, et potentiellement optimiser le temps de ces opérations si elles nécessitent plus d'un cycle, mais ce n'est pas toujours une tâche aisée, en particulier à cause du grand nombre de bascules présent dans ces architectures.

Pour que cette stratégie soit efficace d'un point de vue énergétique, il est nécessaire que le temps de sommeil soit suffisamment long pour que la perte d'énergie due à la puissance statique consommée $P_{Leakage}$ (alors non dépensée) soit plus grande que l'énergie nécessaire à la sauvegarde E_{Backup} et à la restauration $E_{Restore}$ de l'état du système [29] (on néglige dans ce cas $E_{Startup}$ correspondant à l'énergie de remise sous tension du système) :

$$E_{Sleep} \geq E_{Backup} + E_{Restore} \tag{2-1}$$

On peut en déduire le temps de sommeil T_{Sleep} minimal pour que l'opération soit efficace :

$$T_{Sleep} \geq \frac{E_{Backup} + E_{Restore}}{P_{Leakage}} \tag{2-2}$$

Lorsque le microcontrôleur peut être réveillé par des événements externes à celui-ci, il n'est pas toujours possible de prédire la durée des phases de sommeil, c'est pourquoi ce temps minimal T_{Sleep} doit être le plus petit possible. Pour cela, les opérations de sauvegarde et de restaurations doivent non seulement être rapides mais aussi les plus économes en énergie.

Pour intégrer cette stratégie dans des microcontrôleurs, il est nécessaire de réaliser une évaluation en profondeur de cette dernière, afin de quantifier les gains potentiels et les possibles contraintes, mais aussi pour l'optimiser et ainsi profiter au mieux des bénéfices de celle-ci.

2.2. Technologies émergentes

Les différentes stratégies présentées dans la partie précédente ciblent en premier lieu la réduction de l'énergie perdue par les courants de fuite des composants. Bien qu'elles aient une certaine efficacité contre ces pertes d'énergie, elles imposent un développement supplémentaire pour être mises en place et ne sont pas toujours sans contrainte (compromis entre efficacité énergétique, réactivité et complexité de chaque solution). C'est pourquoi l'intégration de technologies émergentes dans les systèmes sur puces en complément ou en remplacement des technologies traditionnelles est étudiée.

2.2.1. FD-SOI 28 nm

En effet, les limitations de la technologie traditionnelle CMOS sur substrat massif (*bulk*) sont devenues une réelle contrainte quant à l'évolution des systèmes sur puces ces dernières années. Si la réduction de la taille du transistor, rendue possible grâce à l'amélioration de la finesse de gravure, a permis d'augmenter la densité et les performances des systèmes sur puces, la densité de puissance a quant à elle augmenté [30]. Cette augmentation de la densité de puissance, à laquelle les courants de fuite contribuent plus que l'activité dynamique lorsque les largeurs de grille de transistor sont inférieures à une trentaine de nanomètres, pose d'importants problèmes de dissipation thermique. D'autres problèmes de fiabilité et de rentabilité du processus de fabrication n'encouragent également pas la poursuite d'une finesse de gravure toujours plus grande dans cette technologie [31]. En réponse à ces limitations de la technologie CMOS sur substrat massif, d'autres architectures CMOS sont développées. Les principales alternatives sont le FinFET (*Fin-shaped Field Effect Transistor*, transistor à effet de champ en forme d'ailette), orienté hautes performances ; et la FD-SOI (*Fully-Depleted Silicon On Insulator*, silicium sur isolant à déplétion complète), dynamiquement ajustable entre performances et basse consommation énergétique.

L'innovation principale de la technologie FD-SOI consiste à introduire une couche d'oxyde afin d'isoler le substrat du canal, du drain et de la source du transistor. Cette isolation, même si pas parfaite, va d'une part permettre de réduire grandement les courants de fuite du transistor entre ces zones (canal, drain et source) et le substrat, et d'autre part permettre des polarisations de ce dernier plus importantes et alors plus intéressantes que ce qui peut être fait avec la technologie classique sur substrat massif, limitée par des effets diodes internes. Par conséquent, l'ajustement dynamique du seuil du transistor est plus facile et bien plus grand, permettant une optimisation accrue de l'efficacité énergétique des circuits intégrés et de leurs performances [32].

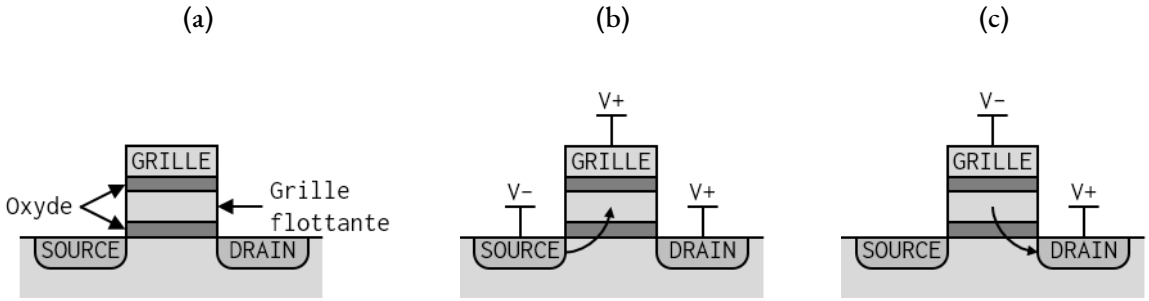
2.2.2. Mémoires non-volatiles

Au-delà de la partie logique, les mémoires embarquées font aussi l'objet d'études depuis plusieurs années pour être remplacées par des technologies plus rapides, plus flexibles et moins gourmandes en énergie [33]. Dans la hiérarchie du microcontrôleur, plusieurs types d'éléments mémoires sont utilisés : les bascules et par extension les registres ; la mémoire de données et la mémoire programme. Chaque élément de cette hiérarchie est une piste d'amélioration qui doit être explorée afin d'optimiser au mieux l'énergie consommée par le contrôleur. Typiquement, les bascules sont volatiles, de même que la mémoire de données ; la mémoire programme elle est non-volatile. Pour obtenir l'un des modes d'économie d'énergie les plus agressifs, un système normalement éteint, il est nécessaire que les éléments mémoires volatiles ne le soient plus. Différentes technologies mémoires non-volatiles sont étudiées pour être intégrées dans les microcontrôleurs, soit pour compléter soit pour remplacer les solutions actuelles.

a) Mémoire Flash

Les mémoires Flash, et plus particulièrement celles de type NOR, sont déjà présentes dans la plupart des microcontrôleurs en particulier comme mémoire programme [34]. Une cellule mémoire est composée d'un transistor, similaire à un transistor MOSFET classique mais disposant d'une « seconde grille » dite flottante, c'est-à-dire qui n'est pas connectée à l'un des terminaux du composant comme illustré par la Figure 2-9 (a). Cette grille située dans l'oxyde isolant de la première grille (alors appelée grille de contrôle) modifie la tension de seuil du transistor en fonction des charges qui y sont emprisonnées. La tension nécessaire à appliquer sur la grille de contrôle pour ouvrir le canal entre la source et le drain du transistor est donc dépendante de l'état de la grille flottante ; c'est grâce à cette variation que la valeur « enregistrée » est lue. Pour charger la grille flottante, une forte tension est appliquée sur la grille de contrôle et le drain, forçant alors un fort courant à traverser le canal du transistor et augmentant sa température. Les électrons traversent alors l'isolant ; ce phénomène est appelé « injection de charges chaudes » (HCI, *Hot Carrier Injection*, Figure 2-9 (b)) [35]. Pour la décharger, une forte tension doit être appliquée entre le drain et la grille de contrôle (la source reste ouverte) ; un effet tunnel décharge la grille flottante (Figure 2-9 (c)).

Il existe deux types d'agencement de mémoire Flash très répandues : le type NOR (« non ou ») et le type NAND (« non et »). Ces noms font référence à l'architecture utilisée pour chacune de ces mémoires. Cette différence dans l'implémentation de banque mémoire Flash permet d'optimiser une partie de leurs métriques : le type NAND a une bien meilleure densité, des temps de programmation et d'effacement plus courts et un coût de fabrication moins élevé que le type NOR ; cependant le temps d'accès en lecture est beaucoup plus grand et il ne supporte pas la méthode d'accès aléatoire en lecture [36] [37]. Les microcontrôleurs ont besoin d'accéder à leur mémoire programme



(typiquement Flash) rapidement et de façon aléatoire, c'est pourquoi le type NOR est utilisé dans ces composants.

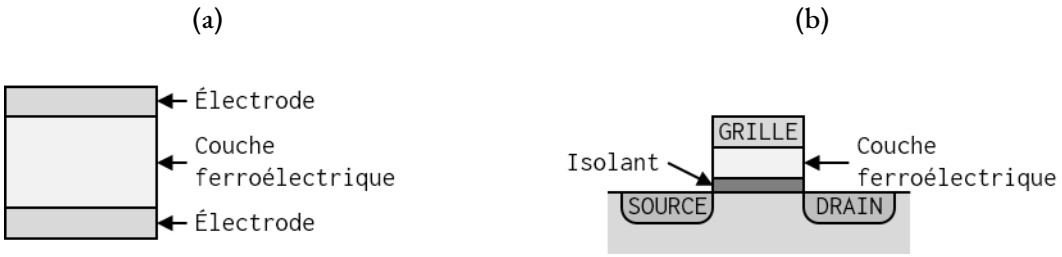
Figure 2-9 : (a) transistor Flash à grille flottante [34] [35] ;
(b) programmation de la mémoire Flash ; (c) effacement de la mémoire Flash

Cependant, cette technologie a ses contraintes. Les mémoires Flash de type NOR, même si plus rapides que le type NAND sur ce point, sont limitées en vitesse de lecture (typiquement quelques dizaines de mégahertz). Des cycles d'attentes sont nécessaires pour opérer à des cadences plus élevées, entraînent une dégradation des performances (due à une réduction du nombre d'instructions par cycle). Ce type de mémoire ne supporte pas l'écriture directe. Il n'est pas possible d'effacer une valeur isolée sans affecter une partie de la mémoire. Pour effacer une valeur, il faut effacer la page où elle se situe, et donc potentiellement effacer d'autres données. Cette opération est coûteuse en temps et en énergie, en plus de la programmation ou reprogrammation de chaque donnée, et l'endurance des mémoires Flash est limitée.

b) Mémoires ferroélectriques

Les mémoires ferroélectriques (lorsqu'on parle de banque, FeRAM ou FRAM) utilisent des cellules composées d'un condensateur ferroélectrique, servant à entreposer une information, et d'un transistor MOS classique, servant à isoler l'élément mémoire des autres cellules logiques d'un système. Le condensateur ferroélectrique utilise un matériau ferroélectrique et non pas un isolant diélectrique comme la plupart des condensateurs classiques [38] (Figure 2-10 (a)). Ce matériau ferroélectrique est polarisé lorsqu'il est soumis à un champ électrique suffisamment fort, champ électrique généré lorsqu'une tension est appliquée entre les deux électrodes du condensateur. Pour lire la valeur inscrite, le contenu de la cellule est effacé : la réaction en courant est différente en fonction de la polarisation de la couche ferroélectrique ; elle permet de déterminer cette valeur. La lecture est par conséquent destructrice, et la donnée doit être réécrite s'il est nécessaire de la conserver [39]. Pour contourner cette contrainte, des transistors FeFET (*Ferroelectric Field Effect Transistor*) utilisant un matériau ferroélectrique entre la grille et le substrat sont développés (Figure 2-10 (b)) ;

Figure 2-10 : (a) composant mémoire ferroélectrique ; (b) transistor ferroélectrique (FeFET)



leur fonctionnement est similaire à celui de la technologie Flash et il est possible de les réaliser dans un nœud technologique avancé [40] [41].

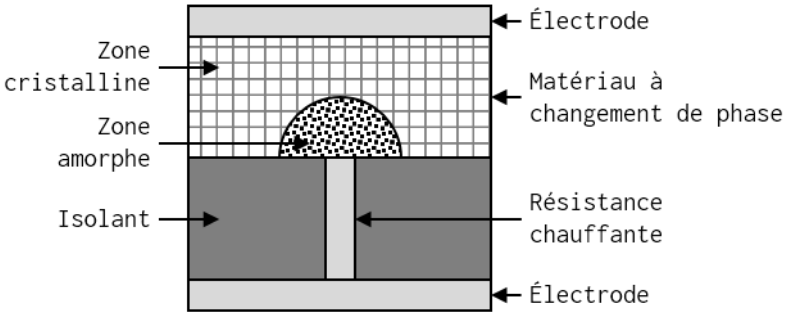
Plus flexible que la technologie Flash, la solution ferroélectrique ne l'a cependant pas (encore) remplacée : la structure des matériaux ferroélectriques est complexe et rend son intégration et la miniaturisation des cellules mémoires difficiles à réaliser [42]. Toutefois cette technologie est toujours étudiée aujourd'hui, et des microcontrôleurs très basse consommation utilisant celle-ci sont commercialisés depuis quelques années : Texas Instrument, au travers de sa série de microcontrôleurs MSP430 [43], met en avant les avantages de la FRAM par rapport à la mémoire Flash (plus flexible car à accès direct, meilleure endurance, permettant d'obtenir une meilleure efficacité énergétique des produits sur batterie) pour les applications de type nœuds capteurs à contraintes énergétiques fortes.

c) Mémoires à changement de phase

Les mémoires à changement de phase (PCM, *Phase Change Memory*, parfois PCRAM) utilisent les propriétés de matériaux tel que les verres de chalcogénure. La résistance de ce matériau change en fonction de sa structure : c'est cette propriété qui est utilisée pour entreposer une donnée. Cette structure est sensible à la température : à l'aide d'une résistance traversée par un courant plus ou moins fort, le matériau à changement de phase est chauffé à une température précise et durant un temps suffisamment long pour rendre sa structure cristalline où sa résistance est la plus faible, ou amorphe où sa résistance est la plus élevée. Il n'est pas nécessaire de rendre la structure du matériau complétement amorphe ; créer une région assez grande pour obtenir la variation de résistance désirée est suffisant (Figure 2-11).

La mesure de sa résistance permet de lire la valeur [44]. Comme pour beaucoup d'autres technologies, elle peut être associée à un transistor pour l'isoler du circuit. En comparaison à la mémoire Flash, les banques de mémoire à changement de phase ont une meilleure endurance, sont plus rapides en lecture et en écriture, et supportent l'accès aléatoire.

Figure 2-11 : Composant mémoire à changement de phase (PCM)



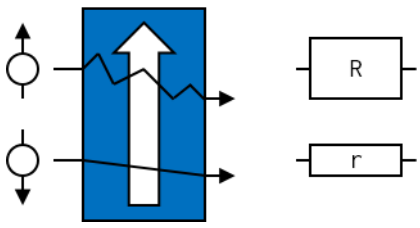
d) Mémoires résistives

Les mémoires résistives (lorsqu'on parle de banque, ReRAM ou RRAM) sont réalisées à partir de memristances (*memristor*), composants dont la résistance change si une tension suffisamment élevée est appliquée à ses bornes. Il peut être associé à un transistor lui permettant d'être isolé du reste du circuit, mais puisque son changement d'état est obtenu par l'application d'une tension à ses bornes et non par un courant comme d'autres technologies, l'utilisation de ces composants sans transistor est étudiée dans le but d'obtenir une mémoire très compact et indépendante du substrat, ce qui permettrait l'extension de la mémoire dans la troisième dimension pour obtenir de très grandes capacités sur une petite surface.

2.2.3. Mémoires magnétiques

La spintronique est un domaine particulier de l'électronique basé sur l'utilisation d'une propriété de l'électron dans les circuits électroniques pour réaliser diverses fonctions, dont la mémorisation de l'information. Le spin est une propriété de toute particule correspondant au moment magnétique de ce dernier. Pour l'électron, cette propriété ne peut avoir que deux états possibles : l'état haut (*up*) et l'état bas (*down*), représentés respectivement par les symboles ↑ et ↓. Lorsqu'un électron se déplace

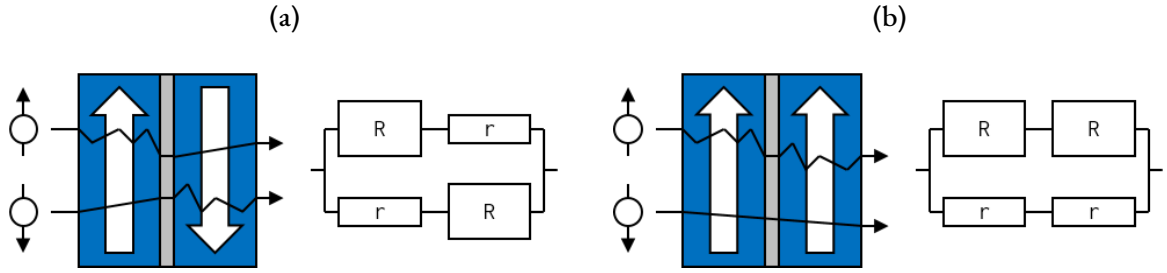
Figure 2-12 : Perturbation par un champ magnétique des électrons traversant un matériau ferromagnétique, selon leur spin



dans un matériau ferromagnétique, son mouvement est perturbé par les champs magnétiques externes. Selon l'orientation de ce champ et celle du spin de l'électron, cette perturbation est plus ou moins grande. Cette perturbation du déplacement des électrons est assimilable à une variation de résistance du matériau : cet effet est la magnétorésistance (Figure 2-12). Si la variation de résistance est faible, elle reste toutefois mesurable et exploitable.

Cet effet a pu être amélioré en faisant traverser des électrons au travers de couches ferromagnétiques séparées par des couches antiferromagnétiques : si l'orientation des couches ferromagnétiques sont parallèles (P), les électrons dont le spin correspond à cette orientation ne seront que très peu perturbés lors qu'ils les traverseront, laissant apparaître une résistance équivalente plutôt faible ; si l'orientation des couches ferromagnétiques sont opposées, c'est-à-dire antiparallèles (AP), tous les électrons quel que soit leur spin seront perturbés à un moment ou un autre dans leur traversée, laissant apparaître une résistance plus élevée comme illustré par la Figure 2-13. Cet effet est la magnétorésistance géante (GMR, *Giant MagnetoResistance*) [45]. Si pour ce phénomène on observe deux extrêmes (état parallèle et antiparallèle), la magnétisation des couches ferromagnétiques peut prendre d'autres directions, et la résistance équivalente est fonction de la différence de leur orientation. Cette particularité est utilisée pour réaliser des capteurs de flux magnétiques pour diverses applications [46]. Dans le domaine du numérique, les valeurs extrêmes de la GMR résultantes sont utilisées pour mémoriser une information binaire ; on retrouve dans le commerce des disques durs dont la tête de lecture est basée sur la GMR.

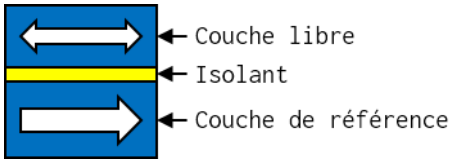
Figure 2-13 : Magnétorésistance géante (GMR) - (a) état antiparallèle (AP) ; (b) état parallèle (P)



Lorsque la couche intermédiaire antiferromagnétique est remplacée par un isolant suffisamment mince, un effet tunnel est observé. Cet effet est un phénomène de la mécanique quantique : une particule qui ne peut traverser une barrière dans la physique classique a, dans la mécanique quantique, une certaine probabilité de la traverser. Cette probabilité est dépendante de l'épaisseur de la barrière et des conditions de la particule et des milieux dans lesquelles elle évolue. Dans le cas d'un électron passant d'une couche ferromagnétique à une autre, cette probabilité est dépendante de son spin et de la magnétisation des couches ferromagnétiques (en plus de la nature des matériaux choisis et de leur forme) [47] [48]. L'effet tunnel va accentuer la magnétorésistance de la cellule, permettant des variations de résistances équivalentes plus grandes que celles de la GMR, ce qui permet dans le

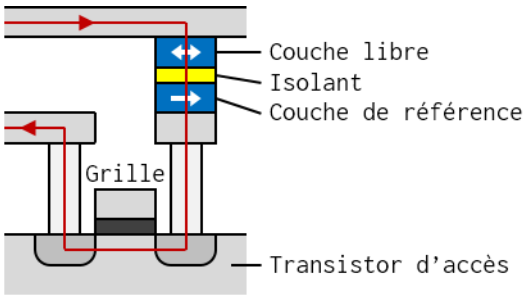
domaine numérique de réaliser une comparaison plus rapide afin d'obtenir l'état binaire de la jonction (parallèle ou antiparallèle). Cette cellule, composée de couches ferromagnétiques séparées par un isolant, est appelée jonction magnétique à effet tunnel (MTJ, *Magnetic Tunnel Junction*, illustrée par la Figure 2-14) et l'effet obtenu avec celle-ci est appelé magnétorésistance à effet tunnel (TMR, *Tunnel MagnetoResistance*).

Figure 2-14 : Schéma simplifié d'une jonction magnétique à effet tunnel (MTJ)



Dans la plupart des applications, et en particulier pour la réalisation de mémoires, la magnétisation de l'une des couches ferromagnétiques de la MTJ est fixée et sert de référence, tandis que la seconde couche ferromagnétique est laissée libre : elles sont alors respectivement appelées couche de référence (RL, *Reference Layer*) et couche libre (FL, *Free Layer*). Dans le domaine du numérique la MTJ est typiquement utilisée avec un transistor (Figure 2-15), permettant de l'isoler d'un circuit pour former une cellule mémoire capable d'enregistrer une information binaire. La forme de la cellule est travaillée pour assurer l'anisotropie magnétique de la couche ferromagnétique libre, et ainsi stabiliser la magnétisation de celle-ci dans le temps. Si la lecture de l'état binaire (P ou AP) de la cellule se fait typiquement par comparaison de sa TMR avec une valeur de référence, l'opération d'écriture, c'est-à-dire la magnétisation de la couche libre, est réalisée différemment selon la version de la technologie.

**Figure 2-15 : Cellule mémoire composée d'une MTJ et de son transistor d'accès ;
la ligne rouge indique le chemin du courant de lecture [49]**

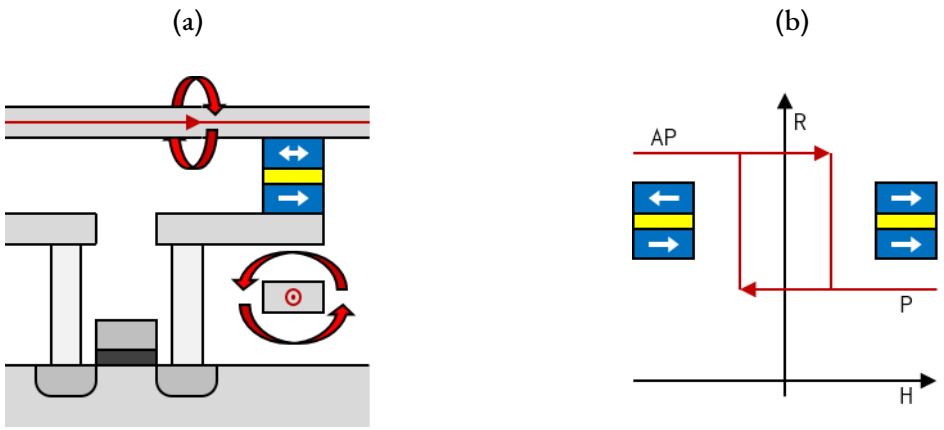


a) MTJ conventionnelle

La première version dite « conventionnelle » consiste à appliquer un champ magnétique extérieur sur la couche libre pour modifier son orientation magnétique, et ainsi faire passer l'état de la cellule à l'état souhaité : parallèle ou antiparallèle. Le champ magnétique est généré à partir d'un

fort courant passant au travers d'une ligne de métal proche de la MTJ. Lorsque les MTJ sont regroupées sous forme de banque mémoire (MRAM), deux lignes de métal perpendiculaires sont utilisées comme illustré par la Figure 2-16 ; la combinaison de ces deux lignes étant nécessaire pour faire basculer l'état de la cellule. L'une des lignes correspond à la sélection du mot dans la mémoire, l'autre à la sélection individuelle des bits du mot [49]. Cette méthode souffre de deux contraintes majeures : le courant nécessaire pour générer un champ magnétique capable de faire basculer l'état d'une MTJ est très grand, rendant cette opération d'écriture très gourmande en énergie ; et certaines cellules sont plus ou moins sensibles que d'autres aux champs extérieurs, en particulier à cause de variations lors de la fabrication, et ont une probabilité non nulle d'être écrites par erreur lorsqu'elles ne sont pas sélectionnées ou, à l'inverse, non-écrites lorsqu'elles le sont.

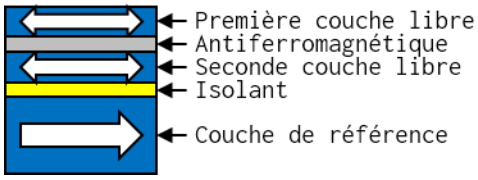
Figure 2-16 : (a) magnétisation de la couche libre à l'aide de champs magnétiques externes [49] ; (b) cycle d'hystérésis représentant le comportement de la MTJ

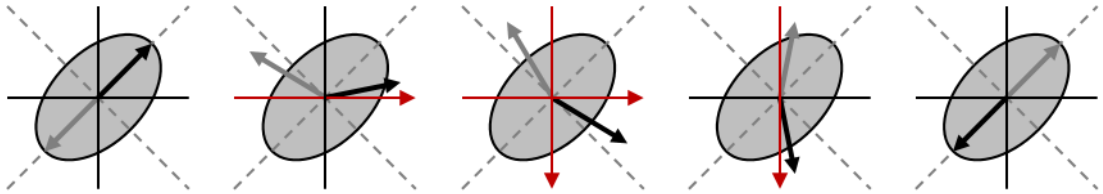


b) Toggle-MTJ

La seconde version vise à réduire la probabilité d'erreurs d'écriture de la méthode conventionnelle en améliorant la sélection des MTJ lors des opérations d'écriture. Une seconde couche ferromagnétique dont l'orientation magnétique est libre est ajoutée à la cellule, séparée de la première couche libre par un matériau amagnétique, comme illustré par la Figure 2-17 [50]. Ces deux couches

Figure 2-17 : « *Toggle MTJ* »



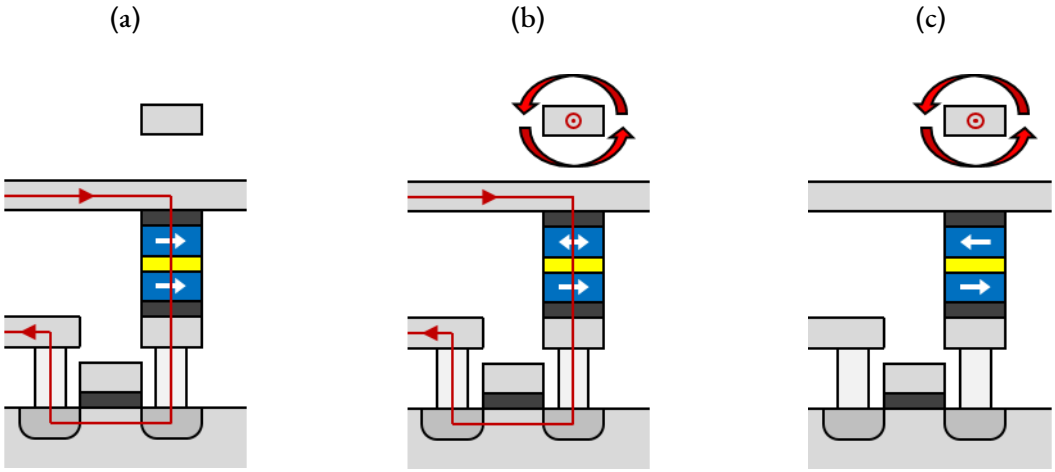


alors couplées magnétiquement permettent d'améliorer leur anisotropie, et par conséquent d'assurer la stabilité de la donnée contenue dans la MTJ. Une séquence particulière détaillée dans la Figure 2-18 doit être exécutée avec les champs magnétiques pour pouvoir faire basculer la magnétisation des couches libres. On peut voir la MTJ de forme ovale, dont l'axe principal est incliné de 45° par rapport aux lignes de métal perpendiculaires, représentées par les lignes verticales et horizontales. La forme ovale de la MTJ est nécessaire pour assurer l'anisotropie magnétique des couches libres : la magnétisation va plus facilement s'orienter dans la longueur de cette forme que dans sa largeur (représentée par les pointillés). Cette magnétisation ne pouvant qu'être basculée et non forcée dans un sens, il est nécessaire de lire l'état de la cellule mémoire avant d'effectuer la séquence pour la faire basculer. Ce principe de fonctionnement est à l'origine de son nom : « *toggle MTJ* » (jonction magnétique à retournement). Comme pour la version conventionnelle, cette méthode d'écriture de la MTJ est très gourmande en énergie, car elle nécessite toujours la création de deux champs magnétiques à partir de lignes de métal.

Figure 2-18 : Séquence pour faire basculer l'état d'une MTJ de type « toggle », vue de dessus [50] ; les lignes verticales et horizontales représentent les lignes de métal générant les champs magnétiques, les flèches rouges symbolisent le passage du courant, les autres flèches la magnétisation des couches libres

c) Assistance thermique (TAS-MTJ)

Les jonctions à commutation assistée par la température (TAS, *Thermally Assisted Switching*) utilisent une barrière thermique pour verrouiller leur couche libre lorsqu'elles ne sont pas sélectionnées [51]. Cette barrière thermique est levée lorsque la couche est chauffée, et cette dernière peut alors être magnétisée par un champ magnétique extérieur. Ce champ est généré par une seule ligne de métal au lieu de deux grâce à la sélectivité offerte par la barrière, ce qui en fait une solution moins gourmande en énergie que les précédentes. La barrière thermique protégeant permet de réduire encore le nombre d'erreurs d'écriture, d'une part parce qu'elle protège la MTJ des champs extérieurs lorsqu'elle n'est pas sélectionnée, et d'autre part parce que la sensibilité de cette dernière est augmentée lorsqu'elle l'est. Cette barrière permet aussi d'améliorer la stabilité thermique de la couche libre, assurant la rétention de la donnée pour une période plus longue. Cependant cette méthode est lente à cause de la séquence d'écriture nécessaire au bon déroulement de l'opération : la jonction est d'abord chauffée en faisant passer un courant à travers celle-ci, en utilisant le même



chemin que pour le courant de lecture (étape de chauffage, Figure 2-19 (a)) ; un champ magnétique est ensuite généré en faisant passer un courant suffisamment fort dans la ligne d'écriture, tout en continuant de chauffer la cellule pour assurer que la barrière thermique ne se régénère pas trop tôt (étape d'écriture, Figure 2-19 (b)) ; le courant traversant la jonction est ensuite coupé tout en maintenant le champ magnétique jusqu'à ce que la jonction refroidisse pour éviter une inversion de la magnétisation avant la régénération de la barrière thermique (étape de refroidissement, Figure 2-19 (c)) [51].

Figure 2-19 : Séquence d'écriture d'une MTJ avec assistance thermique (TAS) [51] ; (a) chauffage ; (b) écriture ; (c) refroidissement

d) Transfert de spin (STT-MTJ)

Dans les versions précédentes, les MTJ sont écrites à l'aide d'un champ magnétique externe généré par un courant traversant des lignes de métal, opération coûteuse en énergie. Grâce au phénomène de transfert de spin (STT, *Spin-Transfer Torque*) il est possible de modifier la magnétisation de la couche libre en faisant traverser la jonction par un courant plutôt qu'en utilisant un champ magnétique externe [52]. Lorsqu'une grande quantité d'électrons dont le spin est polarisé traverse un matériau ferromagnétique, ils exercent un couple magnétique sur celui-ci. Si ce couple est suffisamment grand, le matériau est magnétisé par les électrons [53]. Lorsque les électrons traversent la MTJ en passant d'abord par la couche de référence, dont l'orientation magnétique est fixe, cette dernière agit comme un polarisant et la majorité des électrons passant la jonction tunnel auront un spin en adéquation avec sa magnétisation. Si le courant est suffisamment fort, le couple magnétique généré dans la couche libre sera suffisant pour forcer la magnétisation de celle-ci en adéquation avec le spin des électrons en surnombre, c'est-à-dire dans un état parallèle à la magnétisation de la couche de référence. Bien qu'en traversant la MTJ en passant d'abord par la couche libre la majorité des électrons auront un spin en adéquation avec la magnétisation de cette couche, une minorité auront

le spin opposé. Les électrons dont le spin est en adéquation avec la couche de référence traverseront plus facilement l'isolant à cause du phénomène de l'effet tunnel ; les autres électrons resteront majoritairement piégés dans la couche libre, augmentant leur population. Si le courant est suffisamment fort et maintenu suffisamment longtemps, le couple magnétique généré par ces électrons forcera la magnétisation de la couche libre dans un état antiparallèle à la magnétisation de la couche de référence. En utilisant la méthode du transfert du spin, la cellule n'a pas besoin d'être sensible aux champs magnétiques extérieurs, et sa robustesse fasse à ces perturbations peut être augmentée. La version perpendiculaire de la jonction magnétique à effet tunnel à transfert de spin (pSTT-MTJ, *perpendicular Spin-Transfer Torque Magnetic Tunnel Junction*) permet d'obtenir une densité de cellules mémoires plus grande que les version planaires.

e) Effet spin-orbital (SOT-MTJ)

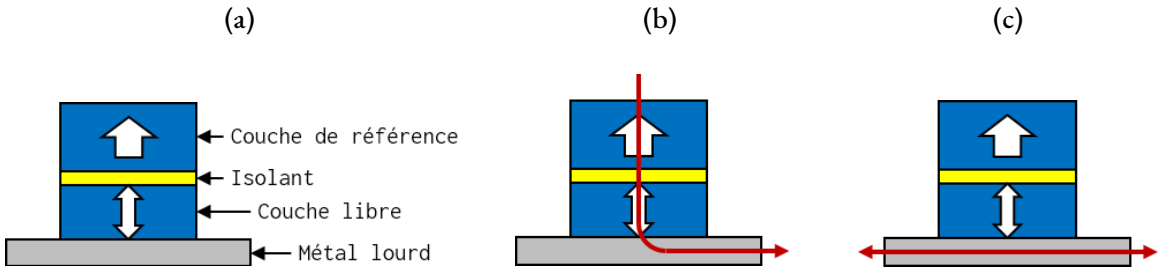


Figure 2-20 : (a) SOT-MTJ ; (b) chemin de lecture ; (c) chemins d'écriture [54]

Lorsqu'un courant traverse un morceau de métal lourd, les électrons vont se disperser différemment sur les surfaces du métal en fonction de leur spin. Ce phénomène est l'effet Hall du spin (SHE, *Spin Hall Effect*) et la modification de la trajectoire des électrons qu'il induit est à la base d'une autre méthode de magnétisation des couches libres des MTJ [54]. Le transfert de spin par l'effet spin-orbital (SOT, *Spin Orbit Torque*) consiste à faire passer un courant au travers d'un métal lourd sur lequel est posée une couche ferromagnétique libre. En fonction du sens du courant dans le métal, les électrons de même spin s'accumulent soit sur la surface où est posée la couche ferromagnétique soit sur la face opposée. Si suffisamment d'électrons sont accumulés, un couple magnétique se forme et s'il est suffisamment puissant, il magnétise la couche ferromagnétique en adéquation avec le spin des électrons. En faisant passer le courant dans le sens inverse, la magnétisation de la couche ferromagnétique est inversée. Contrairement à la STT-MTJ, ici aucun courant n'est forcé à traverser la jonction car les chemins d'écriture et de lecture sont différents (Figure 2-20). Le courant utilisé pour obtenir l'effet SOT est plus faible que pour la STT et le temps d'écriture est beaucoup plus court ; c'est pour l'instant la méthode de magnétisation qui nécessite le moins d'énergie et le moins de temps parmi celles décrites ici. Bien que cette solution paraisse idéale, l'architecture de la SOT-MTJ a trois terminaux ; il est nécessaire de contrôler au moins deux de ces terminaux pour pouvoir

l'isoler dans les banques mémoires, c'est-à-dire au moins deux transistors ou un transistor et une diode. Cette différence affecte la densité des cellules mémoires basées sur des SOT-MTJ. Cependant, il est possible de placer une seconde MTJ sur la face du métal lourd opposé à la première, permettant de gagner un peu de surface l'ors d'une implémentation différentielle.

f) Comparaison des technologies de mémoire magnétique

La première version de la technologie de mémoire magnétique, précédemment appelée « conventionnelle », n'a jamais été commercialisée ni produite à grande échelle. La SOT-MRAM n'est pas encore commercialisée contrairement aux générations précédentes (*Toggle*, TAS et STT), car c'est une technologie assez jeune. Bien que celle-ci possède les meilleures caractéristiques comme souligné par le Tableau 2-1, il a été choisi dans le cadre du projet MASTA de rester sur la technologie STT, plus mature, sur laquelle nous disposons d'une meilleure expérience et d'un état de l'art plus conséquent que celui de la SOT.

Tableau 2-1 : Comparaison des différentes technologies de mémoire magnétique [55]

	Toggle	TAS	STT	SOT
Surface par cellule	50 F^2	< 50 F^2	< 50 F^2	< 50 F^2
Temps de lecture	35 ns	30 ns	< 10 ns	< 5 ns
Temps d'écriture	35 ns	30 ns	< 10 ns	< 5 ns
Courant d'écriture	> 30 mA	< 5 mA	< 50 µA	< 50 µA

g) Comparaison avec les autres technologies

En comparaison avec les autres technologies mémoires non-volatiles présentées dans la partie précédente, la STT-MRAM est celle qui possède la meilleure endurance [36] (Tableau 2-2). Elle fait également partie des technologies avec les temps d'accès en lecture et en écriture les plus courts, permettant une utilisation de celle-ci aussi bien comme mémoire programme ou mémoire de données sans affecter les performances d'un microcontrôleur à basse consommation d'énergie. Pouvant opérer à des tensions d'alimentation faible, elle ne nécessite pas de circuit d'élévation de la tension nécessaire dans la plupart des microcontrôleurs intégrant des mémoires Flash. L'énergie nécessaire pour les opérations de lecture et d'écriture est également l'une des plus faible. Sa densité est cependant plus faible que les autres technologies non-volatiles, mais cette métrique bien qu'importante n'est pas la plus critique pour l'application ciblée (intégration dans une microcontrôleur à très basse consommation d'énergie) et est beaucoup plus grande que la technologie volatile SRAM.

La STT-MRAM est l'une des technologies mémoires non-volatiles des plus prometteuses pour être intégrée dans les microcontrôleurs, en remplacement des technologies actuelles mais aussi pour la réalisation de systèmes normalement éteints.

Tableau 2-2 : Comparaison des différentes technologies mémoires [56]

	SRAM	Flash NOR	STT-MRAM	PCRAM	RRAM
Surface par cellule	> 100 F^2	10 F^2	< 50 F^2	< 30 F^2	< 12 F^2
Alimentation	< 1 V	> 10 V	< 1.5 V	< 3 V	< 3 V
Temps de lecture	~1 ns	~50 ns	< 10 ns	< 10 ns	< 10 ns
Temps d'écriture	~1 ns	> 10 µs	< 10 ns	~50 ns	< 10 ns
Énergie d'écriture	~1 fJ/bit	~100 pJ/bit	~0.1 pJ/bit	~10 pJ/bit	~0.1 pJ/bit
Endurance	> 10^{16}	> 10^5	> 10^{15}	> 10^9	> 10^6
Non-volatile	Non	Oui	Oui	Oui	Oui

2.3. Évaluation des technologies émergentes pour les nœuds capteurs

Aussi bien pour l'utilisation de la technologie FD-SOI et l'intégration de mémoires magnétiques dans les systèmes intégrés, leur évaluation dans un contexte applicatif est nécessaire pour évaluer et optimiser les bénéfices apportés par ces deux technologies émergentes. Plusieurs niveaux d'évaluation sont à considérer (au-delà des études des matériaux et de leurs propriétés physiques) : les évaluations au niveau cellule, que nous ne réalisons pas (les données que nous utilisons proviennent soit de nos partenaires, soit de la littérature) ; les évaluations au niveau bloc, au niveau système et enfin au niveau applicatif. Dans le cas des applications type Internet des Objets, l'évaluation au niveau applicatif peut aussi se faire à plusieurs échelles selon l'infrastructure utilisée : évaluation à l'échelle du nœud, à l'échelle du réseau local, ou à l'échelle du réseau global.

2.3.1. Évaluation des nœuds capteurs

Pour pouvoir réaliser une estimation de la « durée de vie » d'un nœud capteur, c'est-à-dire son autonomie en énergie, il est nécessaire de caractériser les différents composants qu'il intègre en prenant en compte le contexte applicatif, c'est-à-dire tous les événements extérieurs au nœud qui affecteront son comportement et son efficacité énergétique au court de son existence.

Une première estimation de la consommation d'énergie des différents périphériques, en particulier les capteurs et les modules radio, peut être réalisée à partir des valeurs fournies par le fabricant, disponibles dans la documentation technique de ces produits. Cette approche est simple et peu précise, mais elle permet aux développeurs de nœuds capteurs de réaliser une première estimation rapide sans avoir besoin de posséder les produits étudiés, ce qui est très souvent fait lors de la réalisation d'une veille technologique. La mesure est la solution typiquement utilisée pour réaliser une estimation précise de l'efficacité d'une solution : à l'aide d'un outil de mesure de courant ou de puissance consommée, l'énergie dépensée par chaque élément constituant le nœud est mesurée pour chaque phase d'activité ou de sommeil. À partir de cette caractérisation, il est possible

d'extrapoler la consommation d'énergie totale du nœud pour les durées de fonctionnement et les quantités de travail souhaitées. Quant à la durée de vie de ou des accumulateurs d'énergie, elle est bien souvent extrapolée à partir de leur documentation technique.

Si ces méthodes permettent d'effectuer des estimations sur des produits existants, il devient plus difficile de les réaliser sur des produits inexistants, en particulier lorsqu'on cherche à évaluer l'apport de nouvelles architectures (et dans notre cas, intégrant des technologies émergentes) dans ce genre de produit. En effet, pour pouvoir estimer les gains en termes de performances et d'énergie d'une nouvelle architecture, il faut soit disposer d'un prototype matériel implémentant celle-ci, soit disposer d'un modèle comportemental et énergétique de celle-ci et être capable dans ce dernier cas de reproduire son fonctionnement en réel.

Si le sujet étudié n'est qu'une partie d'un système en remplacement d'une solution existante en faisant partie, il est théoriquement possible de réaliser une évaluation en caractérisant le système et sa partie à modifier. Cependant, les produits existants sont comparables à des boîtes noires, où les informations disponibles sont celles du fabricant, très souvent limitées à la documentation technique, et celles qu'il est possible de mesurer directement sur le produit. Il est parfois possible de dégager d'autres informations en réalisant plusieurs mesures dans des conditions différentes, mais elles ne sont parfois pas suffisantes pour pouvoir caractériser correctement un bloc interne d'un système. Or, il est nécessaire de comprendre la contribution d'un bloc sur les performances et la consommation d'énergie globale du système pour pouvoir étudier son remplacement par une autre solution.

2.3.2. Programmes applicatifs de référence

Nous désirons comparer les performances d'exécution et la consommation d'énergie de différentes solutions. L'utilisation de programmes applicatifs de référence (*benchmark*) est très répandue afin de comparer les caractéristiques de plusieurs architectures pour une même charge de travail. Il existe beaucoup de programmes de référence différents, chacun permettant d'évaluer un type de solution pour une application, une tâche ou une charge de travail particulière : performances de calcul (capacité à réaliser des opérations complexes, de gérer des grands nombres, entiers ou à virgule flottante...) ; gestion de la mémoire (mise en évidence de la latence et du débit mémoire pour des opérations de lecture, d'écriture, de copie de donnée, optimisation des antémémoires...) ; gestion des applications multitâches ; exécution de tâches en parallèle (*multiprocessing*) ; gestion de tâches dites temps-réel ; optimisation de l'énergie... Ces programmes permettent d'obtenir une « valeur » afin d'évaluer une solution pour une charge de travail particulière.

La mesure de l'énergie consommée par un système lors de l'exécution d'un programme de référence permet aussi d'évaluer et de comparer différentes solutions sur ce point. Combiner les indices de performance de calcul à cette consommation énergétique permet d'obtenir l'efficacité

énergétique, une métrique très importante pour l'évaluation des solutions pour les applications dont la gestion d'énergie est un point critique, en particulier les réseaux de capteurs sans fils. Certains programmes de référence sont très répandus et les fabricants de micro-processeurs et microcontrôleurs n'hésitent pas à en faire usage et à afficher leurs résultats, ce qui offre une base de données intéressante sur l'efficacité des produits actuellement disponibles sur le marché.

EEMBC (*Embedded Microprocessor Benchmark Consortium*) [57] proposent une série de programmes applicatifs de référence aussi bien pour l'évaluation de systèmes orientés hautes performances de calcul que pour l'évaluation de l'efficacité énergétique de petites architectures. Le CoreMark est un programme applicatif de référence mis au point par EEMBC pour évaluer les performances des microcontrôleurs et micro-processeurs utilisés dans les systèmes embarqués [58]. Il peut être compilé et exécuté sur des architectures ayant des largeurs de bus de données de 8, 16, 32 et 64 bits, et réalise des opérations sur des matrices de nombre, des manipulations de listes chaînées (recherche par index et valeur, tri, ajout et suppression d'éléments, fusion de listes avec tri), des générations de code de détection d'erreurs. Le CoreMark donne en sortie un score utilisé pour comparer les solutions entre elles.

ULPMark est une suite de programmes de référence, toujours réalisés par EEMBC, dédiés plus particulièrement à reproduire le fonctionnement cyclique des nœuds capteurs sans fil pour évaluer et comparer l'efficacité énergétique des microcontrôleurs basse consommation à destination de ce type d'applications. Un programme de cette suite est le CoreProfile, aussi appelé ULPMark-CP [59], conçu pour reproduire un comportement périodique alternant entre phase active et phase de sommeil. Durant la phase active, diverses charges de travail sont exécutées : fonctions mathématiques (approximation linéaire, filtrage), tables de conversion, recherche de chaînes de caractères, copie de tableaux, fonctions de tri, permutation de données et manipulation de sorties. Contrairement au CoreMark dont le score obtenu ne reflète que la performance de calcul, le CoreProfile prend aussi en compte l'énergie consommée durant son exécution, témoignant alors de l'efficacité énergétique de la solution. Cette suite comprend également un programme de référence dédié à l'évaluation de l'efficacité énergétique des périphériques, le PeripheralProfile (ou ULPMark-PP).

Une autre suite plus récente de programmes applicatifs de référence réalisée par EEMBC, IoTMark, utilise des capteurs et des modules de communication radio afin de reproduire le fonctionnement d'un nœud capteur complet et ainsi prendre en compte une partie des éléments externes au contrôleur dans son évaluation. Ne supportant qu'un nombre réduit de modules radio et de périphériques de communication, le nombre d'applications ciblées par cette suite est assez réduit, mais reste cependant très intéressant.

Une solution alternative consiste à concevoir ou prendre le programme de l'application ciblé pour réaliser l'évaluation et la comparaison de solutions dans les meilleures conditions. Ce programme peut aussi être celui de produits existants, opérationnels et déployés au sein d'un réseau de nœuds capteurs. Cette méthode ne permet cependant pas de profiter des données déjà existantes qu'offre

d'autres programmes de référence, disponibles dans la littérature, chez les fabricants ou sur les plates-formes mettant à dispositions ces programmes comme EEMBC.

2.3.3. Architectures ultra-basse consommation

Le Tableau 2-3 présente différentes études menées pour réaliser des microcontrôleurs ultra-basse consommation intégrant des technologies émergentes et/ou des stratégies d'économie d'énergie agressives, avec l'objectif d'optimiser cette consommation énergie.

L'approche utilisée pour atteindre cet objectif est différente d'une étude à une autre. Des bascules hybrides à mémoire résistive sont utilisées dans [60] ; on retrouve aussi des bascules hybrides dans [61] et dans [62] mais utilisant une autre technologie mémoire, de type ferroélectrique. Pour [26], ce ne sont pas des bascules hybrides mais une banque mémoire non-volatile de type ferroélectrique qui est utilisée pour sauvegarder l'état du processeur. Selon l'application visée, certains travaux sont également portés sur la réduction de l'énergie lorsque le microcontrôleur est en activité. Ainsi, [63] et [64] cherchent à réduire cette puissance active en utilisant des techniques de polarisation du substrat (*body-biasing*), permettant entre autre d'optimiser l'énergie dans les transistors.

Les bascules hybrides utilisées pour atteindre la non-volatilité du processeur utilisent des éléments mémoires non-volatiles de type résistif pour [60] et de type ferroélectrique pour [61] et [62]. Dans le cas de [26], les banques mémoires non-volatiles sont de type ferroélectriques et les mémoires principales, non-volatiles, le sont également. Pour [60], la mémoire programme est une mémoire de type résistif, mais la mémoire de données elle est hybride avec une partie non-volatile de type résistif.

Les bascules utilisées dans [64] ne sont pas hybrides, ni purement non-volatiles, mais elles possèdent toutefois une architecture particulière pour réduire leur courant de fuite durant les phases de sommeil. En effet, contrairement aux bascules classiques composées de deux verrous, ces bascules en possèdent un troisième appartenant à un domaine de puissance différent du reste du processeur. Lorsque le système passe en phase de sommeil, ces verrous additionnels copient le contenu de l'esclave de leur bascule. Le restant du système voit alors son alimentation coupée durant la phase de sommeil, alors que les verrous additionnels restent alimentés durant cette phase. Au réveil, leur contenu est recopié sur les esclaves des bascules du processeur. Le type de transistor utilisé pour ces verrous additionnels ne sont pas les mêmes que ceux utilisés pour le reste des bascules, ceci dans l'objectif de minimiser les pertes d'énergie durant les phases de sommeil sans affecter les performances du processeur lorsqu'il est en activité.

Les mémoires utilisées dans [63] sont toutes volatiles et de petite capacité comparé aux autres microcontrôleurs présentés dans ce tableau. La mémoire programme étant volatile celle-ci doit être initialisée au démarrage du microcontrôleur. Pour cela, un composant externe à celui-ci doit être prévu à cet effet, ce qui est une contrainte non négligeable pour l'intégration d'une telle solution.

Tableau 2-3 : Caractéristiques de microncontrôleurs ultra-basse consommation intégrant des technologies émergentes

Work	Lallement 2018 [63]	Wang 2017 [60]	Zwerg 2017 [26]	Su 2017 [61]	Izumi 2015 [62]	Singhal 2015 [64]
Technology	28 nm FD-SOI	65 nm SVT	130 nm	130 nm	130 nm	90 nm
Power supply	-	0.8 V (core) 3 V (NV)	1.2 V / 1.5 V	1.5 V	1.2 V / 3.0 V	3 V
Frequency	16 MHz	100 MHz	8 MHz	25 MHz	24 MHz	16 MHz
Processor	Cortex-M0+	-	Cortex-M0+	8-bit	Cortex-M0	MSP430
Memory architecture	4 kB SRAM (program) 4 kB SRAM (data)	8 kB ReRAM 4 kB NVSRAM Resistive NVFF	96 kB ROM 2×32 kB FRAM Non-Volatile Array	128 B register file 8 kB SRAM Ferroelectric NVFF	16 kB 6T4C NVRAM (program and data) Ferroelectric NVFF	64 kB NVRAM 8 kB SRAM MTCMOS FF
Active power	2.7 pJ/cycle	33 µW/MHz	150 µA/MHz at 1.2 V	106 µW/MHz	6.14 µA	28.3 µW/MHz
Sleep power	0.7 µW at 0.5 V	0 W	0 W	0 W (CPU)	-	0.32 µW at 3 V
Wake-up and recovery	-	0.45 nJ 20 ns (NVFF) 170 ns (NVSRAM)	380 nC 438 µs	5.04 pJ/bit 3 µs recovery 46 µs total	-	-
Backup	-	0.40 µJ 4 µs (NVFF) 1.02 ms (NVSRAM)	-	14.4 pJ/bit 7 µs backup 14 µs total	-	-

Parmi les autres microcontrôleurs, ceux présentés par [60], [26] et [62] possèdent une architecture mémoire entièrement composée d'éléments non-volatiles, permettant d'obtenir un système normalement éteint et ainsi la consommation d'énergie la plus basse possible durant les phases de sommeil en minimisant le déplacement des données.

2.3.4. Évaluation du contrôleur

Nous nous intéressons très particulièrement au contrôleur central des nœuds capteurs, c'est-à-dire le microcontrôleur, afin d'évaluer l'intégration de technologies émergentes dans l'objectif d'améliorer l'efficacité énergétique de ces systèmes. Il est nécessaire de pouvoir réaliser une exploration de l'architecture de ces composants suffisamment fine pour obtenir une certaine précision dans nos estimations. Un microcontrôleur intégrant de nombreux blocs de nature différente (régulateurs de tension, horloges, éléments analogiques, mémoires...), il est impératif de discriminer la contribution de chacun sur la consommation d'énergie totale du circuit. Malheureusement, cette discrimination n'est pas réalisable sur les microcontrôleurs existants car ils sont dépourvus des outils nécessaires pour effectuer cette tâche : les lignes d'alimentation sont partagées pour un grand nombre des éléments du circuit (parfois des lignes d'alimentation séparées sont utilisées pour des blocs spécifiques, par exemple pour la RTC, les parties analogiques ou le contrôleur de réveil, mais ce n'est pas suffisant pour notre analyse) et les mécanismes d'isolement ne sont pas suffisants (pas ou trop peu d'alimentations commandées). L'architecture doit être accessible si nous voulons réaliser une exploration de celle-ci ; cependant, nous n'avons pas accès aux différentes architectures utilisées dans les produits du commerce. C'est pourquoi, plutôt que de reprendre entièrement un microcontrôleur existant, nous avons réalisé notre propre architecture à partir des informations disponibles.

Concernant l'évaluation des performances d'une architecture et l'estimation de sa consommation d'énergie, elles peuvent être réalisées par différentes méthodes : par la réalisation d'un prototype suivit de mesures sur celui-ci et par la simulation.

Implémenter l'architecture à étudier dans la technologie souhaitée pour effectuer des mesures sur cette dernière reste la méthode qui permet d'obtenir les résultats les plus justes. Opérant en temps réel et pouvant être interfacé avec d'autres composants, le prototype obtenu peut être directement intégré dans son environnement final, ce qui permet de l'évaluer en prenant en compte le contexte applicatif. Cependant, la réalisation d'un circuit intégré prend beaucoup de temps (conception, vérification, fabrication et validation) et est très onéreux, et le produit obtenu n'est pas flexible : pour chaque modification voulue, il faut fabriquer un nouveau circuit. La réalisation d'un prototype matériel en vue d'essais et d'évaluations est de préférence réservée pour une étape de validation avancée.

Certains processeurs utilisés dans les contrôleurs basse consommation intègrent des modules de suivi (*trace module*), outils puissants utilisés pour l'analyse et l'optimisation de programmes. Ces outils sont capables de relever des informations relatives à l'exécution d'une application et peuvent être utilisés pour analyser l'activité du processeur [65]. Ils ne sont cependant pas disponibles sur tous les processeurs, en particulier sur ceux de petite taille, leur présence étant dépendante du concepteur et des choix d'intégration du fabriquant. De plus, ces modules sont centrés sur le processeur et ne peuvent relever des événements externes à celui-ci. De la même manière, certains micro-processeurs intègrent des moniteurs de performances (PMU, *Performance Monitoring Unit*), qui, bien qu'initialement prévus pour analyser la performance des processeurs, peuvent être utilisés pour obtenir une estimation de la puissance consommée par ces derniers [66]. Ces moniteurs de performances sont composés de compteurs programmables, configurables depuis le logiciel, utilisés pour capturer des événements indicateurs de la performance des processeurs [67]. Le nombre de compteurs est toutefois limité, et, encore une fois, toujours centré sur les processeurs.

La simulation est la méthode la plus flexible : chaque modification désirée peut être apportée au modèle, tant que celui-ci est éditable, et il est possible de représenter n'importe quel comportement, d'un bloc ou d'une technologie. Elle peut être relancée à souhait et il est possible de paralléliser plusieurs simulations. Un simulateur permet de se dispenser de matériel, ce qui en fait une solution moins coûteuse et plus rapide à mettre en place que la précédente. La simulation peut être réalisée à plusieurs niveaux selon l'abstraction voulue : au niveau transistor, très précise mais très lente ; au niveau registre ou RTL (*Register Transfer Level*), plus rapide qu'au niveau précédent mais toujours très lente, précise au bit près et nécessitant des modèles comportementaux et énergétiques des cellules utilisées ; utilisation de modèles précis au cycle (CAM, *Cycle Accurate Model*), plus rapides à simuler que des modèles RTL mais ajoutant une abstraction faisant perdre la précision au bit et certaines contraintes temporelles internes ; simulation au niveau instruction (ISS, *Instruction Set Simulator*), encore plus rapide, mais se détache de la partie matérielle à cause de l'abstraction, ce qui amène à la perte d'informations et de la précision au cycle [68] ; les différents niveaux logicielles (routines bas niveau, routines du noyau, appels aux fonctions d'une bibliothèque...) jusqu'à la modélisation des états d'un programme applicatif. Cependant, si augmenter le niveau d'abstraction permet de diminuer le temps de simulation, la précision diminue [69]. Or, il est nécessaire d'avoir une précision suffisamment fine pour évaluer les modifications faites sur une architecture, et plus particulièrement au niveau de la technologie. De plus, pour simuler un contrôleur de nœud capteur et réaliser une évaluation prenant en compte le contexte applicatif, il est nécessaire de modéliser tous les éléments externes (capteurs, modules de communication, réseau, environnement...) qui viennent modifier son comportement. La création de modèles pour tous ces événements externes est possible, mais c'est un développement supplémentaire, une possible source d'intrusion d'erreurs, et la modélisation de tous les cas possibles peut prendre beaucoup du temps.

Entre l'implémentation matérielle et la simulation, il est possible de prototyper une architecture à l'aide de circuits logiques reconfigurables, notamment des FPGA (*Field-Programmable Gate Array*, réseau de portes programmables). Ces composants sont utilisés pour reproduire le comportement d'une architecture, décrite la plupart du temps dans un langage de description matérielle (HDL, *Hardware Description Language*). Dans le cadre de l'évaluation d'architectures et d'applications, il existe des outils et des méthodologies de simulation couplés avec une implémentation sur FPGA pour profiter d'une accélération matérielle tout en gardant une grande précision [70] [71], qui peuvent aussi être utilisés pour réaliser une évaluation de la consommation d'énergie [72]. Cependant, ces outils bien que précis et rapides ne sont pas conçus pour respecter la dimension temporelle, et ne profitent alors pas du fait que les FPGA sont des circuits physiques pouvant être interfacés avec d'autres composants. Cela est dû aux types d'architectures auxquels sont destinés ces outils : des architectures orientées haute performance, qui peuvent difficilement être implémentées sur un FPGA à cause de leur taille et des ressources nécessaires à leur fonctionnement. Ce dernier point met en évidence des limitations de l'implémentation sur FPGA, qui sont les limitations physiques de ce composant programmable. Toutefois, pour des petites architectures comme celles des microcontrôleurs basse consommation, il est possible d'implémenter son architecture dans son intégralité à l'intérieur d'un FPGA et de réaliser une évaluation en temps réel. En effet, ces contrôleurs sont généralement cadencés à des fréquences de quelques dizaines de mégahertz, voir quelques centaines de mégahertz pour les plus puissants, des fréquences atteignables pour une partie des FPGA du commerce. En reproduisant une architecture de microcontrôleur sur un FPGA, il est possible de l'interfacer avec des périphériques externes comme ceux utilisés sur les nœuds capteurs, et ainsi réaliser une évaluation de cette architecture en prenant en compte les événements extérieurs, et donc le contexte applicatif. On peut ainsi réaliser une plateforme de prototypage de nœud capteur, intégrable à un réseau existant.

L'activité de tout signal d'une architecture synthétisée sur le FPGA peut être capturée à l'aide d'outils implémentés dans ce dernier. En sélectionnant les signaux ayant une forte corrélation avec la puissance consommée instantanée de cette architecture ou d'une partie de celle-ci, pour une technologie donnée, il est possible d'obtenir un modèle énergétique pour ce bloc. Alimenté par l'activité de ces signaux, obtenu avec le FPGA, ce modèle permet d'obtenir en temps réel la consommation d'énergie d'une architecture ou d'un bloc pour une application donnée.

2.4. Conclusion

Aujourd'hui, on utilise des réseaux de capteurs sans fils dans de nombreux domaines, pour la réalisation d'applications très diverses. L'énergie est le point critique des applications embarquées de type nœud capteur, c'est pourquoi de nombreux efforts sont fournis aujourd'hui pour augmenter l'efficacité énergétique des microcontrôleurs basse puissance. Différentes stratégies sont utilisées afin

d'économiser l'énergie disponible sur les nœuds capteurs, en particulier l'organisation des différentes tâches du microcontrôleur en plusieurs phases où une régulation de l'alimentation et de la charge de travail de chaque composant du nœud est réalisée et optimisée pour réduire l'énergie dépensée, tout en garantissant la qualité de service du nœud. Lorsqu'un composant du nœud est inactif, son alimentation peut être coupée ; cependant, ce n'est pas applicable sans pénalité pour tous les composants, principalement à cause d'un temps de démarrage et de reconfiguration trop coûteux en temps et en énergie, en particulier pour les contrôleurs des nœuds, typiquement des microcontrôleurs. Différents modes de puissance sont disponibles sur ces derniers, faisant appel à une régulation de leurs tensions d'alimentation et de leur fréquence de fonctionnement pour optimiser leur consommation d'énergie en fonction des différentes phases de l'application. Grâce à ces stratégies, il est possible de supprimer une partie des fuites de courant des composants inactifs, responsables de la perte d'une quantité non négligeable d'énergie. Malgré cela, plus le mode d'économie d'énergie utilisé est agressif, plus le temps de réaction du nœud est long ; c'est un compromis encore trop restrictif pour une partie des applications de l'Internet des Objets.

De nouvelles solutions technologiques sont alors envisagées pour compléter ou remplacer les solutions actuelles afin d'augmenter l'efficacité énergétique des nœuds capteurs, parmi lesquelles deux sont prometteuses : la technologie FD-SOI et les mémoires magnétiques, en particulier de type STT et SOT, jusque-là peu évaluées pour la réalisation de microcontrôleurs normalement éteints. Bien que prometteuses, il est toutefois nécessaire de réaliser des évaluations de ces nouvelles solutions, de leur intégration et des nouveaux paradigmes qu'elles offrent. La réalisation d'un prototype matériel étant plutôt réservée pour les étapes de validation d'une solution, la simulation est aujourd'hui le principal moyen pour évaluer l'intégration de nouvelles technologies, de nouvelles architectures voire de nouvelles stratégies. Cependant, il y a un compromis entre la précision désirée des résultats et le temps d'évaluation. En effet, plus la simulation est précise, plus elle est lente à être exécutée, or il est nécessaire de garder une précision élevée pour évaluer les modifications que nous voulons apporter aux microcontrôleurs tout en étant capables d'exécuter des applications de longue durée. De plus, les outils d'évaluation avancés sont majoritairement dédiés à l'évaluation d'architectures à haute performance de calcul, avec peu de modèles d'architectures similaires à ce qu'on pourrait trouver dans un microcontrôleur ; aucun outil actuel ne semble adapté pour pouvoir réaliser ces travaux d'investigations de manière complète, rapide et efficace pour des microcontrôleurs à faible puissance. Face à cette problématique, nous avons choisi de mettre en place une nouvelle méthode d'évaluation des systèmes numériques basse puissance à partir d'une plateforme de prototypage FPGA, nous permettant d'augmenter la vitesse de l'évaluation par rapport à la simulation classique et de prendre plus facilement en compte le contexte applicatif.

3. Méthodologie d'évaluation de nœuds de capteurs

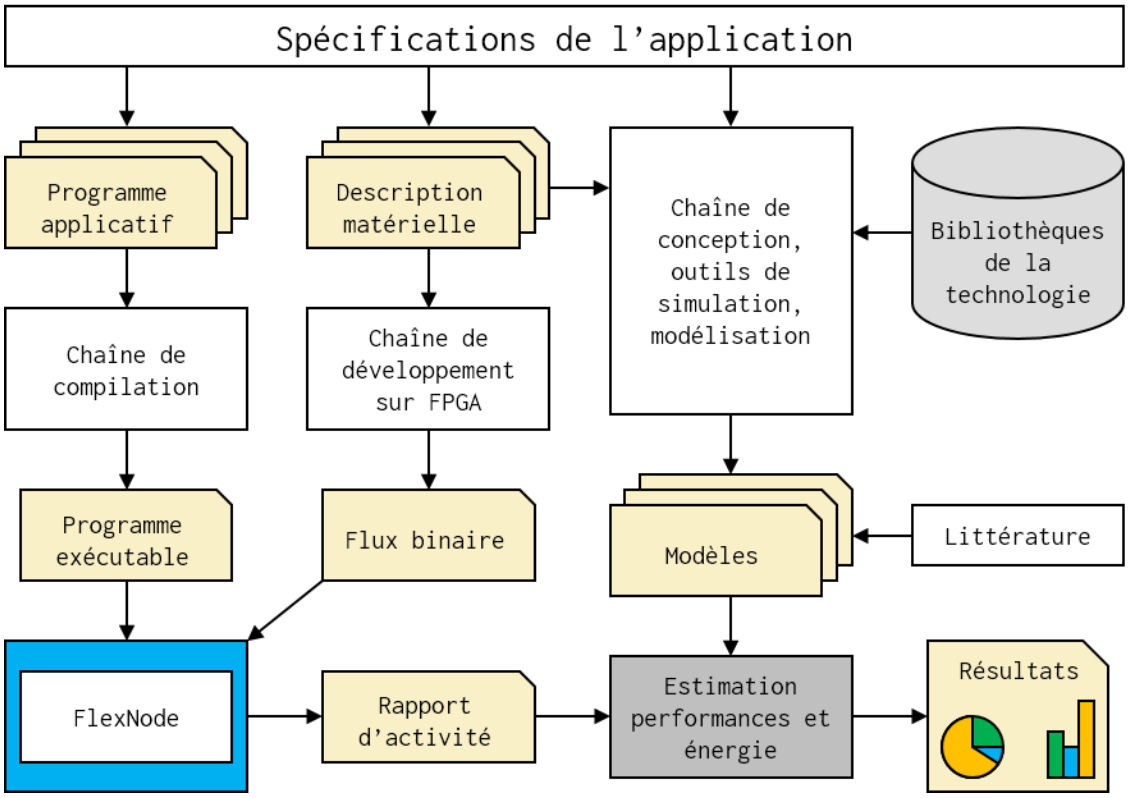
Typiquement, l'estimation de la consommation énergétique d'un circuit, d'un bloc destiné(e) à une implémentation ASIC (*Application-Specific Integrated Circuit*, circuit intégré propre à une application) est réalisée à partir d'un modèle énergétique de celui-ci alimenté par l'activité du système étudié. Ce modèle énergétique peut être composé de plusieurs modèles indépendants, obtenus grâce à des simulations bas niveau, des bibliothèques d'une technologie fournies par un fabricant, de mesures d'un prototype, parfois disponibles dans la littérature. Quant à l'activité du système, elle correspond au comportement de ce dernier dans une application donnée. Elle peut être obtenue par différentes méthodes selon la nature du système étudié. Dans le cas d'un circuit numérique, cette activité est typiquement obtenue grâce à des simulations RTL. Pour les composants programmables tel que les microcontrôleurs, le comportement du circuit est dépendant d'un programme applicatif obtenu à partir d'une description dans un langage de programmation comme l'assembleur ou le langage C. Si des événements extérieurs sont nécessaires à l'évaluation, ils sont injectés durant la simulation, de façon préprogrammée et déterministe. Au cours de cette simulation, l'activité du système est relevée puis fournit à un outil d'analyse qui, à partir de cette activité, de la description du système et du ou des modèles énergétiques des éléments qui le compose, donnera une estimation de la puissance consommée par ce système.

Cette méthodologie présente plusieurs difficultés. Pour commencer, la simulation RTL d'un système est une opération très lente, dont le temps nécessaire à sa complétion s'allonge lorsque le système simulé prend de la complexité. De plus, les études impliquant des applications nécessitant des temps d'exécution longs en sont également pénalisés. Ensuite, les événements externes sont difficiles à reproduire. Typiquement préprogrammées, la prise en compte des différentes combinaisons possibles est très compliquée car elle demande de relancer plusieurs fois une simulation, opération déjà très lente, ce qui allonge une nouvelle fois le travail d'évaluation. Il est alors difficile de prendre en compte un grand nombre de cas de figure, et des situations critiques ne pourraient pas être détectées, or les applications de type nœud capteur sont dépendantes des événements extérieurs.

Afin d'accélérer l'évaluation de ces systèmes numériques, nous proposons de substituer l'étape de simulation par du prototypage sur FPGA. Les modèles énergétiques sont toujours obtenus avec les méthodes traditionnelles (simulation bas niveau, mesure...) mais l'activité du circuit n'est quant à elle plus récupérée de la même manière. L'architecture étudiée, dans notre cas celle d'un microcontrôleur, est synthétisée puis implémentée sur une plateforme de prototypage à l'aide d'outils de développement FPGA. L'architecture ne subit que peu de changements (exclusivement des adaptations de blocs propres aux technologies telles que les mémoires ou les entrées/sorties) et son comportement reste identique entre les versions FPGA et ASIC. Afin de relever des informations sur son activité, nous ajoutons sur cette architecture un outil dédié à cette tâche : le moniteur d'activité.

Le FPGA réalisant le lien physique avec le monde extérieur, des périphériques externes sont câblés à celui-ci afin d'obtenir un prototype complet et ainsi réaliser une évaluation au niveau applicatif, en prenant en compte les événements externes. Une fois l'activité récupérée, elle vient alimenter les différents modèles énergétiques pour obtenir une estimation de la consommation énergétique de l'architecture étudiée, comme représenté par la Figure 3-1.

Figure 3-1 : Flot d'évaluation



3.1. Conception d'une architecture de microcontrôleur

Notre objectif est d'évaluer l'intégration de technologies mémoires magnétiques dans les microcontrôleurs très basse puissance. Pour réaliser ce travail d'exploration, nous devons avoir la maîtrise d'une architecture de microcontrôleur pour pouvoir la modifier à souhait. Cela implique d'avoir accès à celle-ci à un grain suffisamment fin, au niveau cellule, mais aussi de connaître parfaitement son fonctionnement. N'ayant pas accès à des architectures avancées satisfaisant nos besoins, nous avons décidé de réaliser notre propre microcontrôleur. L'objectif ici est donc de réaliser une architecture adaptée pour l'exploration, pouvant être intégrée sur un nœud capteur, afin de réaliser des évaluations dans le contexte des applications type Internet des Objets. Les différentes caractéristiques de cette architecture sont étudiées pour se rapprocher de réalisations existantes et pour pouvoir être utilisées avec d'autres composants (capteurs, moteurs radios et autres

périphériques externe) afin de faciliter le prototypage d'un nœud autour de notre microcontrôleur. Notre architecture doit être dotée bien évidemment d'un processeur et de mémoires pour l'entreposage du programme applicatif et des données, mais aussi des périphériques de communication nécessaires pour interagir avec différents composants externes. Nous y intégrerons également les outils nécessaires pour obtenir l'activité du circuit, nécessaire pour réaliser des estimations en terme de performance et de consommation d'énergie.

3.1.1. Spécifications

Les microcontrôleurs peuvent être classés selon plusieurs critères : le type de processeur (hautes performances, basse puissance), les modes d'économie d'énergie, les périphériques et les accélérateurs matériels, le nombre de broches, la hiérarchie mémoire (technologie, capacité, fonctionnalités). Les spécifications de l'application déterminent quel microcontrôleur utiliser. Les fabricants proposent une très grande variété de microcontrôleurs afin de répondre au mieux aux besoins de chaque application et de ses contraintes, chaque solution offrant un compris entre performances de calcul, basse puissance, haute capacité mémoire et flexibilité. On trouve des microcontrôleurs avec différents types et tailles de boîtiers, nombre de broches, processeurs, fréquences de fonctionnement, périphériques, interfaces de communication, modules analogiques, technologies mémoires, capacités mémoires, modes de consommation. Les microcontrôleurs sont regroupés en fonction de l'application ciblée, et ne vont pas toujours répondre aux mêmes critères. Par exemple, les microcontrôleurs dédiés aux domaines de l'automobile, de la défense, de l'aviation ou du spatial ont des contraintes de longévité, de fiabilité et de résistance face aux aléas du climat (en particulier la température) importantes ; pour d'autres contraintes comme le suivi des produits ou la garantie du maintien de la production pour une durée minimale, des essais renforcés sont demandés. Les microcontrôleurs destinés à être intégrés sur les nœuds capteurs très basse consommation sont conçus pour avoir une faible puissance consommée, en fonctionnement comme en mode économie d'énergie, et une grande efficacité énergétique. Ils possèdent typiquement un processeur de petite taille, des mémoires de capacité plus faible que la plupart des microcontrôleurs prévus pour des applications plus gourmandes, et certains périphériques ne sont pas toujours disponibles (par exemple, le CAN, un bus de communication industriel que l'on retrouve dans le domaine automobile). Certains microcontrôleurs intègrent un moteur radio afin de se passer de l'ajout d'un module externe, ce qui permet de réduire la taille du nœud, sa consommation d'énergie et potentiellement son coût [73].

La largeur du bus de données est l'une des caractéristiques importantes de l'architecture des microcontrôleurs. Elle impacte le jeu d'instruction et la façon dont les adresses sont gérées (et limitées) et, par conséquent, la taille du processeur, ses performances et la consommation d'énergie de l'architecture. Le choix de ce paramètre dépend du type de données à manipuler et des

performances souhaitées : à conditions égales, un processeur 32 bits a une puissance consommée instantanée plus grande qu'un processeur 8 ou 16 bits, mais a tendance à être plus performant pour la manipulation de grands types de données et la réalisation de calculs complexes ; le bilan énergétique n'est donc pas systématiquement plus grand. Aujourd'hui, les largeurs les plus répandues dans les microcontrôleurs du commerce sont 8, 16, et 32 bits. D'autres largeurs de bus de données existent (4 bits par exemple), mais elles sont de moins en moins utilisées et ne sont plus mises en avant (ou tout simplement plus disponibles) dans les catalogues des fabricants. L'évolution du marché des microcontrôleurs montre que les architectures de 32 bits de largeur de données sont de plus en plus utilisées [74] ; c'est pourquoi nous nous sommes intéressés plus spécifiquement à ces derniers.

Le Tableau 3-1 présente l'architecture mémoire de microcontrôleurs destinés aux applications très basse consommation disponibles sur le marché. Chaque microcontrôleur possède au moins une mémoire de type non-volatile, typiquement en technologie Flash, principalement destinée à entreposer les instructions du programme de l'application et les données statiques, et au moins une mémoire volatile pour les données dynamiques. Cette dernière est typiquement une SRAM (*Static Random Access Memory*, mémoire statique à accès direct). Certains microcontrôleurs disposent d'autres types de mémoires. Parmi les plus communes :

- les mémoires mortes (ROM, *Read Only Memory*), contenant la plupart du temps des programmes d'amorçage réalisés par le fabricant, programmes permettant la récupération des instructions d'un programme applicatif par une interface communication ;
- les EEPROM (*Electrically Erasable Programmable Read-Only Memory*, mémoire morte effaçable électriquement et programmable), une mémoire non-volatile proche de la Flash mais moins rapide et plus volumineuse que cette dernière, qui peut être utilisée par l'application, souvent pour sauvegarder des données ou des paramètres sur le long terme, en prévention d'un éventuel redémarrage du microcontrôleur, une coupure de son alimentation ou l'entrée dans un mode d'économie d'énergie agressif entraînant la perte du contenu des mémoires volatiles ;
- les mémoires volatiles de sauvegarde (*backup memory*), mémoires généralement de faible capacité et alimentées indépendamment du reste du microcontrôleur, afin qu'elles gardent leur contenu lors de l'utilisation de modes d'économie d'énergie agressifs.

Certains microcontrôleurs n'ont pas de mémoire reprogrammable pour l'entreposage du programme applicatif, utilisant à la place de la mémoire Flash une simple mémoire morte (dont le contenu est fixé lors de la conception) ou une mémoire programmable une seule fois (mémoire OTP, *One-Time Programmable memory*) pour des raisons de coût, d'espace, de performances et de consommation d'énergie, mais leur nature ne permet pas la reconfiguration du produit. Il est possible de trouver encore d'autres technologies mémoires utilisées dans les microcontrôleurs, mais leur

présence est plus rare. Certains produits disposent d'une interface pour l'ajout d'une mémoire externe afin d'augmenter leur capacité et leurs fonctions.

Le Tableau 3-2 présente des architectures réalisées dans un contexte académique. Ces dernières ont été grandement optimisées pour une application précise, c'est pourquoi elle présentent une hiérarchie mémoire parfois différente des microcontrôleurs présentés dans le Tableau 3-1. En effet, l'application étant connue au moment de leur réalisation, les concepteurs ont pu dimensionner les différentes mémoires en fonction des besoins de celles-ci, voir en intégrant directement le programme applicatif dans une mémoire morte à la capacité adaptée, limitant ainsi le matériel inutilisé et par conséquent les pertes d'énergie associées. Il existe toutefois quelques similarités : bien souvent les instructions et les données sont séparées dans des mémoires différentes et la capacité des mémoires sont du même ordre de grandeur. L'utilisation de mémoires volatiles pour l'entreposage des instructions implique que ces dernières doivent être fournies au microcontrôleur par un élément externe, par exemple avec une mémoire Flash externe pour [75], par un programmateur pour [76] ou directement par un ordinateur. Dans ce dernier cas, un programme d'amorçage contenu dans une mémoire morte est intégré dans l'architecture.

Tableau 3-1 : Architecture mémoire des microcontrôleurs 32 bits très basse consommation du commerce

Product	Frequency	Processor	Flash	SRAM	EEPROM
STM32L0 series [77] STMicroelectronics	32 MHz	ARM Cortex-M0+	8 to 192 kB	2 to 20 kB	512 to 6144 B
STM32F0 series [78] STMicroelectronics	48 MHz	ARM Cortex-M0	16 to 256 kB	4 to 32 kB	-
LPC800 series [79] NXP Semiconductors	30 MHz	ARM Cortex-M0+	4 to 64 kB	1 to 16 kB	-
LPC1100 series [80] NXP Semiconductors	50 MHz	ARM Cortex-M0 ARM Cortex-M0+	4 to 256 kB	2 to 36 kB	512 to 4096 B
nRF51 series [73] Nordic Semiconductor	32 MHz	ARM Cortex-M0	128 to 256 kB	16 to 32 kB	-
PSoC 4 family [81] Cypress Semiconductor	48 MHz	ARM Cortex-M0 ARM Cortex-M0+	8 to 256 kB	2 to 32 kB	-
RX100 series [82] Renesas	32 MHz	Renesas RXv1	8 to 512 kB	8 to 64 kB	-
PIC32MM Family [83] Microchip	25 MHz	32-bit MIPS	16 to 256 kB	4 to 32 kB	-
SAM L MCUs [84] Microchip	48 MHz	ARM Cortex-M0+ ARM Cortex-M23	16 to 256 kB	4 to 40 kB	1024 to 8192 B

Tableau 3-2 : Architecture mémoire de prototypes de microcontrôleurs 32 bits très basse consommation

	Frequency	Processor	Program memory	Data memory
Brown 2020 [85]	7 MHz	Cortex-M0	256 kB (shared)	
Myers 2019 [76]	50 MHz	Cortex-M33	128 kB ROM 128 kB SRAM	16 kB SRAM 4 kB backup
Lallement 2018 [63]	16 MHz	Cortex-M0+	4 kB SRAM	4 kB SRAM
Zwerg 2017 [26]	8 MHz	Cortex-M0+	96 kB ROM	2×32 kB FRAM
Paul 2017 [75]	297 MHz	IA-32	16 kB ROM 8 kB cache	8 kB DTCM 64 kB (shared)
Izumi 2015 [62]	24 MHz	Cortex-M0	16 kB NVRAM (shared)	

Si la plupart des fabricants proposent leur propre architecture d'unité centrale dans les microcontrôleurs 8 bits, ce n'est pas le cas pour le marché des microcontrôleurs 32 bits. Certains fabricants continuent de développer leur propre solution en 32 bits, comme Renesas et ses processeurs RX [86] ou Texas Instrument et leur C28x ; mais beaucoup intègrent des processeurs issus d'un concepteur tiers tel que les Cortex-M de ARM ou les processeurs de MIPS Technologies comme indiqué par le Tableau 3-1. La majorité des microcontrôleurs présentés dans ce tableau utilise les solutions de ARM, et intègre les processeurs les plus petits et les plus orientés basse consommation parmi les Cortex-M disponibles du concepteur : le Cortex-M0, le Cortex-M0+ et, le plus récent, le Cortex-M23. On note également que ces architectures ont des fréquences de fonctionnement assez basses, inférieures à 50 MHz.

Dans le cas des architectures présentées dans le Tableau 3-2, les processeurs de ARM sont également très utilisés. Toutefois depuis quelques temps un grand intérêt est porté sur les architectures de processeur basé sur le jeu d'instruction RISC-V, un jeu d'instruction ouvert et libre qui pour des raisons de souveraineté de la propriété intellectuelle présente un enjeu certain, en opposition aux solutions actuelles presque exclusivement privées (ARM, Renesas, MIPS, Intel...).

Les microcontrôleurs intègrent de nombreux outils pour communiquer et interagir avec le monde extérieur. En dehors des contrôleurs d'entrées/sorties classiques, on retrouve dans une très grande partie des microcontrôleurs les périphériques suivants :

- UART (*Universal Asynchronous Receiver Transmitter*, émetteur-récepteur asynchrone universel), une interface série pour une liaison point-à-point ;

- SPI (*Serial Peripheral Interface*, interface série pour périphériques), un bus de communication série synchrone avec une hiérarchie maître-esclaves ;

- I²C (*Inter-Integrated Circuit*, liaison inter-circuit intégré)[1], un autre bus de communication série synchrone avec une hiérarchie maître-esclaves dont dispose de nombreux capteurs ;
- CAN (*Controller Area Network*), un bus de communication industriel très utilisé dans l'automobile avec une couche protocole comprenant identification, taille de message et code de détection d'erreur entièrement gérée de façon matérielle ;
- USB (*Universal Serial Bus*), bus série universel ;
- ADC (*Analog to Digital Converter*), convertisseur analogique-numérique ;
- DAC (*Digital to Analog Converter*), convertisseur numérique-analogique ;
- des compteurs (*timer*), parfois outillés pour générer des signaux avec une modulation de la largeur d'impulsion (PWM, *Pusle Width Modulation*) et/ou pour mesurer des délais, des largeurs d'impulsion, des rapports cycliques, le déphasage entre plusieurs signaux.

D'autres outils dédiés à des fonctions particulières sont également présents dans une partie des microcontrôleurs basse consommation : contrôleur d'afficheurs à cristaux liquides (ACL, ou LCD pour *Liquid Crystal Display*) ; contrôleur de touches capacitives ; périphériques dédiés à la sécurité, en particulier des modules de chiffrement type AES (*Advanced Encryption Standard*) et des générateurs de nombres aléatoires (RNG, *Random Number Generator*) ; des horloges temps réel (RTC, *Real Time Clock*).

Les processeurs embarqués pour microcontrôleur de la série Cortex-M de ARM sont assez répandus dans les systèmes sur puce du commerce, comme constaté précédemment. Nous avons choisi l'un de ces processeurs comme unité principale afin de pouvoir facilement trouver des microcontrôleurs aux caractéristiques similaires qui serviront de référence matérielle. De plus, les processeurs Cortex-M0 et Cortex-M3 sont disponibles dans une version offusquée au travers du programme DesignStart proposé par ARM, synthétisable et pouvant être implémenté sur un FPGA. La version r1p0 du Cortex-M0 a été retenue. ARM donne une limitation en fréquence à 50 MHz pour ce processeur, ce qui correspond aux valeurs présentées dans le Tableau 3-1. Puisque le Cortex-M0 possède une interface de bus de type AHB-Lite, c'est ce format qui sera utilisé pour le bus de communication principal.

Une architecture mémoire similaire à celles présentées dans le Tableau 3-1 sera utilisée : une première mémoire à accès direct (RAM, *Random Access Memory*) de 128 ko servira pour les instructions du programme de l'application et les données statiques ; une seconde mémoire à accès direct de 16 ko sera dédiée aux données dynamiques. Une mémoire morte contenant un programme d'amorçage sera également disponible ; sa capacité sera dépendante de la taille de ce programme.

[1] Aussi appelé par certains constructeurs TWI (*Two Wire Interface*) ou TWSI (*Two Wire Serial Interface*), interface (série) à deux lignes.

Notre microcontrôleur doit disposer de périphériques pour communiquer avec des composants externes à celui-ci. Parmi les périphériques souvent utilisés, certains d'entre eux ne sont pas nécessaires pour un nœud sans fil, en particulier ceux dédiés à un réseau, une connexion filaire comme le CAN ou l'USB. Puisque nous nous intéressons aux réseaux de capteurs sans fils, ces périphériques ne seront pas retenus. À l'inverse, certains périphériques sont nécessaires pour pouvoir connecter le microcontrôleur avec des capteurs, des moteurs radio ou autres composants présents sur un nœud capteur : UART, SPI et I²C.

Afin d'obtenir des informations sur le système au cours de son fonctionnement, le microcontrôleur sera doté d'outils permettant la mesure de son activité, nécessaire pour réaliser une évaluation de ses performances et des estimations de consommation d'énergie.

Le Tableau 3-3 résume les différents choix stratégiques concernant l'architecture MASTA.

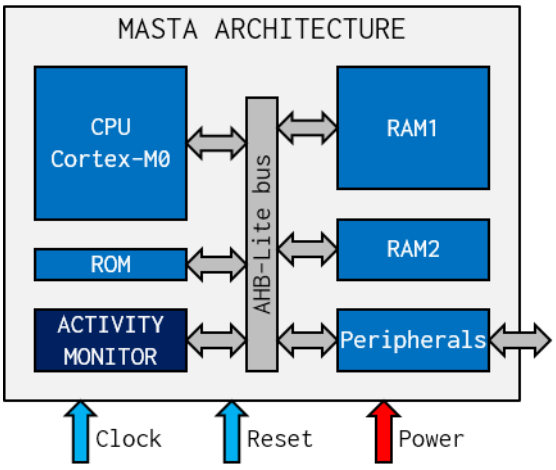
Tableau 3-3 : Spécifications de l'architecture MASTA

Processeur	ARM Cortex-M0
Fréquence de fonctionnement maximale	50 MHz
Architecture mémoire	ROM 2 ko (programme d'amorçage) RAM 128 ko (mémoire programme) RAM 16 ko (mémoire de données)
Périphériques	GPIO UART SPI I²C Moniteur d'activité

3.1.2. Architecture MASTA

L'architecture réalisée est présentée par la Figure 3-2. Elle est composée du processeur Cortex-M0 de ARM, d'une mémoire morte de 2 ko (ROM, *Read-Only Memory*), contenant un programme d'amorçage (*bootloader*), d'une mémoire volatile de 128 ko, de périphériques pour la gestion des entrées/sorties (GPIO, *General Purpose Inputs/Outputs*), la communication et la gestion des signaux de réinitialisation, et la configuration minimale d'un bus type AMBA 3 AHB-Lite (*Advanced Microcontroller Bus Architecture 3 Advanced High-performance Bus*) pour une architecture simple maître (comprenant un esclave par défaut, un décodeur d'adresse et un multiplexeur). Un dernier périphérique, nommé Moniteur d'Activité (*Activity Monitor*) et présent dans l'architecture, est utilisé pour capturer différents événements. Ces captures sont ensuite utilisées dans le flot d'exploration pour obtenir des estimations quant à la performance du système. Ces différents composés sont présentés dans le détail dans la suite.

Figure 3-2 : Architecture du microcontrôleur



a) Processeur ARM Cortex-M0

Le Cortex-M0 est un processeur 32 bits de type RISC à 3 étages implémentant le jeu d'instruction ARMv6-M (ISA, *Instruction Set Architecture*). Il possède une unique interface de communication de type AMBA 3 AHB-Lite avec système d'adressage sur 32 bits, supportant des transactions de données de largeur 8, 16 et 32 bits. Cette interface est aussi utilisée pour l'acheminement des instructions et des données, en provenance et à destination des mémoires et des périphériques externes au processeur. Le Cortex-M0 intègre un contrôleur d'interruptions externes (NVIC, *Nested Vectored Interrupt Controller*), pouvant supporter jusqu'à 32 lignes d'interruption et une ligne d'interruption non blocable (NMI, *Non-Maskable Interrupt*). En interne, le cœur de ce processeur possède un multiplieur simple cycle ou multi-cycles (option configurable par l'intégrateur) et une vingtaine de registres d'état, dont 13, nommés de R0 à R12, sont dédiés à un usage général. Certains composants sont optionnels : un compteur 24 bits avec ligne d'interruption dédiée ; un contrôleur de réveil (WUC, *Wake-Up Controller*) ; une interface de débogage supportant jusqu'à quatre points d'arrêt matériels.

Le jeu d'instruction supporté par le Cortex-M0, ARMv6M, est principalement composé d'instructions encodées sur 16 bits pour la réalisation d'opérations logiques et arithmétiques usuelles ; seulement 6 instructions sont au format 32 bits. De ce fait, ce processeur a la capacité de récupérer deux instructions en une seule opération de récupération, limitant ainsi le nombre d'appels à la mémoire par rapport au nombre d'instructions à exécuter.

Nous utilisons le processeur fourni dans le programme DesignStart de ARM, qui est une version offusquée du Cortex-M0 ne possédant ni module de débogage ni mode basse consommation. Il est délivré sous la forme d'une liste d'interconnexions (*netlist*) synthétisable qui, malgré l'offuscation, permet toutefois de visualiser l'état des registres du processeur.

b) Périphériques de communication

Afin de pouvoir communiquer avec des composants externes, nous avons choisi d'intégrer dans notre microcontrôleur différents périphériques dédiés à la communication : UART, SPI et I^2C. L'un des périphériques UART fait la liaison avec un ordinateur hôte. Il est utilisé pour transmettre au microcontrôleur les instructions du programme applicatif à exécuter puis pour communiquer avec l'application elle-même. Les autres interfaces sont utilisées pour communiquer avec les autres composants du nœud de capteur (capteurs, moteurs radio).

Typiquement, les microcontrôleurs implémentant un processeur ARM possèdent un bus de communication interne dédié aux périphériques, utilisant le protocole AMBA 3 APB (*Advanced Peripheral Bus*) du même concepteur [87]. Un pont fait alors la liaison entre le bus périphérique vers le bus principal (utilisant le protocole AHB-Lite). Toutefois, nous n'utilisons pas de bus dédié aux périphériques dans notre microcontrôleur : au cours de la conception de ce dernier, le premier périphérique (de type UART) a directement été connecté au bus principal de type AHB-Lite afin de vérifier rapidement le concept. Les autres périphériques ont été conçus à partir des périphériques UART et GPIO (ces derniers étant connectés à un bus AHB-Lite sur les microcontrôleurs traditionnels), et ce de façon progressive selon l'évolution des besoins. Aucun n'a été adapté pour fonctionner sur un bus de type APB, cependant cette adaptation peut faire partie d'une future évolution du circuit.

c) Gestion des entrées et sorties

Plusieurs périphériques sont dédiés à la gestion des entrées/sorties du microcontrôleur.

Le module de gestion des entrées/sorties à usage général (GPIO, *General Purpose Inputs/Outputs*) est un contrôleur basique qui permet la lecture des entrées du système et le contrôle des sorties sur demande du processeur. Selon la technologie et les cellules utilisées, il est possible d'ajouter d'autres options manipulables par le processeur, tel que des résistances de tirage (*pull-up resistor*) ou de rappel (*pull-down resistor*), un cycle d'hystérésis pour la lecture de l'entrée ou encore la modification du courant de sortie (*slew rate control*). Bien que d'une technologie à une autre, et d'une cellule standard à une autre, ces options puissent varier, les contrôleurs de plots bidirectionnels sont typiquement composés d'un étage de sortie à trois états (*tristate buffer*) commandé par deux signaux, l'un activant l'étage de sortie et l'autre donnant la valeur à reproduire, et d'un étage d'entrée (*input buffer*) avec un signal de sortie reproduisant la valeur à l'entrée de la borne. À partir de ces trois signaux, on peut obtenir différents modes de fonctionnement pour chaque borne du microcontrôleur : entrée à haute impédance, sortie à drain ouvert (*open drain*) ou sortie symétrique (*push-pull*).

Un module nommé PPS (*Peripheral Pin Select*, sélection des bornes périphériques) permet d'attribuer le contrôle d'une borne du microcontrôleur à la sortie du périphérique GPIO correspondant ou à un autre périphérique du système. Il est divisé en deux sous-modules, l'un réalisant la connexion entre les signaux de commande en provenance des périphériques et en

direction des cellules dédiées aux entrées/sorties (nommé *PPSOUT*), l'autre réalisant la connexion entre le retour de ces cellules et l'entrée des périphériques (nommé *PPSIN*). Comme tous les autres périphériques, il est contrôlable par le processeur grâce à une interface AHB-Lite. Il est utilisé pour câbler n'importe quelle entrée ou sortie d'un périphérique sur n'importe quelle borne du microcontrôleur. Une même sortie de périphérique peut contrôler plusieurs broches en sortie, mais une même broche ne peut être contrôlée que par un seul périphérique à la fois. De la même manière, l'entrée d'un périphérique ne peut être reliée qu'à une seule borne, mais une même borne peut être reliée à plusieurs entrées de périphériques.

d) Contrôleurs mémoires

Pour connecter les mémoires au bus AHB-Lite, des contrôleurs dédiés sont nécessaires. Ces derniers doivent traduire les transactions du protocole AHB pour répondre au fonctionnement des interfaces mémoires. Dans notre microcontrôleur, deux types de mémoires seront utilisés : une mémoire à lecture seule (ici une mémoire morte) simple port de 32 bits (SPROM, *Single Port ROM*) et des mémoires à accès direct simple port de 32 bits (SPRAM, *Single Port RAM*). Les mémoires à accès direct peuvent aussi bien être des mémoires volatiles, comme la classique SRAM, ou des mémoires non-volatiles.

Pour les mémoires type SPROM, le contrôleur est très simple car tous les signaux nécessaires à l'interface mémoire sont disponibles ou peuvent être produits dès la première phase de la transaction. L'interface SPROM ne supportant que des lectures de mots de 32 bits (4 octets), les deux derniers bits de l'adresse ainsi que la largeur du mot sont ignorés. Une erreur est générée par le contrôleur si une demande d'écriture est détectée, si la largeur du mot demandée excède 4 octets ou si un problème d'alignement est détecté.

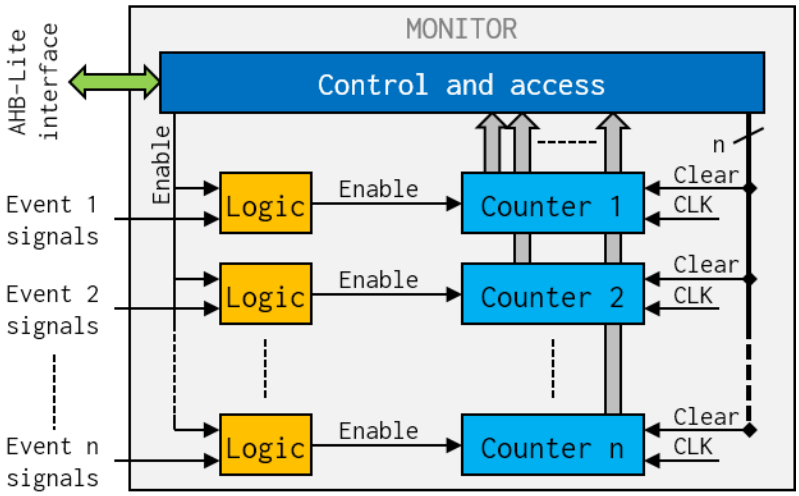
Il existe deux grandes catégories d'interfaces RAM, celle avec écriture directe (*standard write*) et celle avec écriture retardée (*late write*). Dans le cas de l'écriture directe, la valeur à écrire doit être fournie avec l'adresse et les autres signaux d'instruction (en un seul cycle d'horloge) à l'interface de la mémoire. Dans le second cas, l'écriture retardée, la valeur à écrire doit être fournie au cycle d'horloge suivant. Puisque le protocole AHB-Lite impose un retard d'un cycle entre les signaux de contrôle et les données lors des transactions d'écriture, les mémoires à écriture retardée sont les plus simples à adapter au bus AHB-Lite. Cependant, ce ne sont pas les interfaces les plus répandues ; nous ne disposons pas de mémoire de ce type. Il faut donc un contrôleur mémoire adapté pour interfacer une mémoire à accès direct au bus AHB-Lite tout en respectant les contraintes des deux protocoles.

Pour cela, un premier registre tampon est utilisé pour retarder les signaux de contrôle d'un ou plusieurs cycles, pour fournir les signaux et les données à l'interface mémoire durant un même cycle. Un second registre tampon permet de retarder les données si la transaction d'écriture doit être différée, dans le cas où une ou plusieurs demandes de lecture surviendraient après la demande d'écriture.

e) Moniteur d'activité

Nous souhaitons capturer l'activité de plusieurs éléments de notre microcontrôleur, et réutiliser cette information plus tard dans le cadre de l'évaluation de ces éléments. Nous avons développé un moniteur d'activité composé d'un ensemble de compteurs utilisés pour capturer les événements témoins de l'activité du circuit. L'architecture de ce composant, illustrée par la Figure 3-3, est basée sur celle des moniteurs de performances (PMU, *Performance Monitoring Unit*) que l'on peut trouver sur certains micro-processeurs, mais contrairement à ces derniers nous l'utilisons pour capturer des événements sur l'ensemble du microcontrôleur.

Figure 3-3 : Architecture du moniteur d'activité



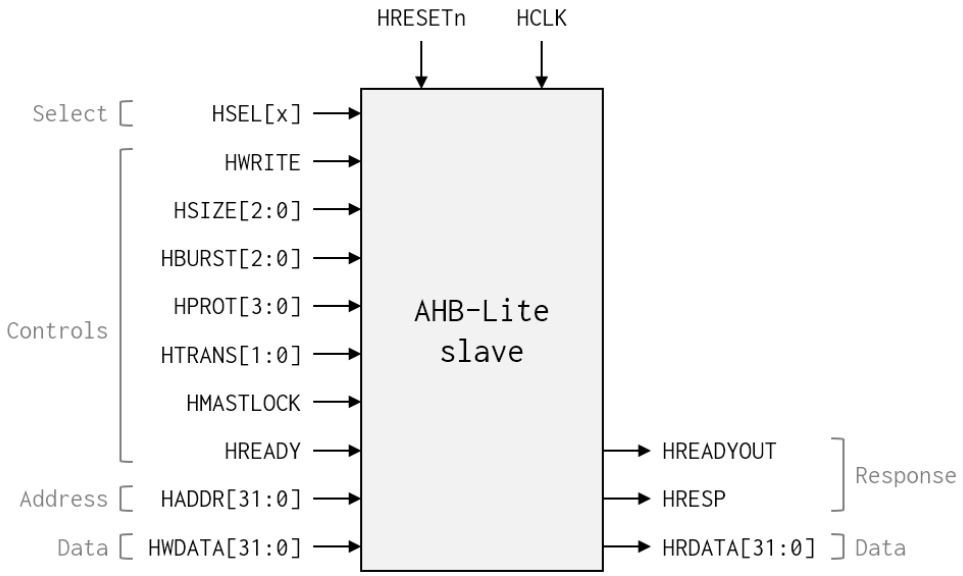
Dans notre système, le moniteur d'activité est conçu pour capturer les événements suivant :

- nombre de cycles ;
- nombre d'instructions exécutées ;
- nombre de récupérations d'instruction (*instruction fetches*) ;
- nombre d'opérations de lecture de la mémoire ;
- nombre d'opérations d'écriture de la mémoire.

Le moniteur d'activité est connecté au bus AHB-Lite du système afin d'être contrôlable par le processeur. Cela permet le démarrage, l'arrêt et la réinitialisation des compteurs par le programme grâce à un registre de contrôle. Il est donc possible de réaliser une observation des événements pour une portion de programme voulue, sans avoir recourt à du matériel additionnel.

Pour compter le nombre de cycles, un simple compteur est utilisé. Le compteur d'instructions exécutées est incrémenté chaque fois que le compteur ordinal (PC, *Program Counter*)[2] change. Les autres compteurs sont incrémentés lorsqu'une opération d'accès mémoire valide est détectée et que la condition liée à l'opération (lecture, écriture ou récupération d'instruction) est respectée. Puisque le moniteur d'activité et la mémoire sont sur le même bus, il dispose des mêmes signaux que celle-ci et peut utiliser sa propre interface pour détecter les différents accès à cette mémoire. Si le moniteur n'était pas câblé sur le même bus que la mémoire, il aurait été nécessaire de faire parvenir les signaux nécessaires depuis la mémoire jusqu'à celui-ci. La Figure 3-4 présente les signaux de l'interface esclave pour un bus type AHB-Lite (extrait de la documentation [88]).

Figure 3-4 : Interface esclave type AHB-Lite



Chaque signal de commande correspond à une fonction précise, et c'est à partir de la combinaison de ces signaux que les transactions mémoires sont réalisées. À partir de la documentation, nous avons déterminé les combinaisons correspondantes aux différents événements que nous désirons capturer.

Un accès à la mémoire est détecté lorsque la condition suivante est respectée :

$$RAMAccess = HSEL[RAM] \cdot HREADY \cdot HTRANS[1] \qquad 3\text{-}1$$

Où *HSEL[RAM]* est la ligne de sélection dédiée à la mémoire (signal actif à l'état haut), *HREADY* est le signal qui informe si le précédent transfert est terminé (lorsqu'il est à l'état haut) et la requête d'un nouveau transfert est indiqué par le signal *HTRANS[1]* (actif à l'état haut).

[2] Aussi appelé pointeur d'instruction (IP, *Instruction Pointer*)

Pour déterminer quel type de transfert mémoire est demandé, le moniteur prend en compte l'état de deux autres signaux : *HPROT[0]*, qui indique si le transfert est une récupération d'instruction (état bas) ou un transfert de donnée (état haut), et *HWRITE*, qui indique la direction du transfert (lecture si état bas, écriture si état haut).

Une récupération d'instruction est détectée lorsque la condition suivante est respectée :

$$RAMFetch = RAMAccess \cdot \overline{HPROT[0]} \cdot \overline{HWRITE} \quad 3\text{-}2$$

Une lecture mémoire est détectée lorsque la condition suivante est respectée :

$$RAMRead = RAMAccess \cdot HPROT[0] \cdot \overline{HWRITE} \quad 3\text{-}3$$

Une écriture mémoire est détectée lorsque la condition suivante est respectée :

$$RAMWrite = RAMAccess \cdot HPROT[0] \cdot HWRITE \quad 3\text{-}4$$

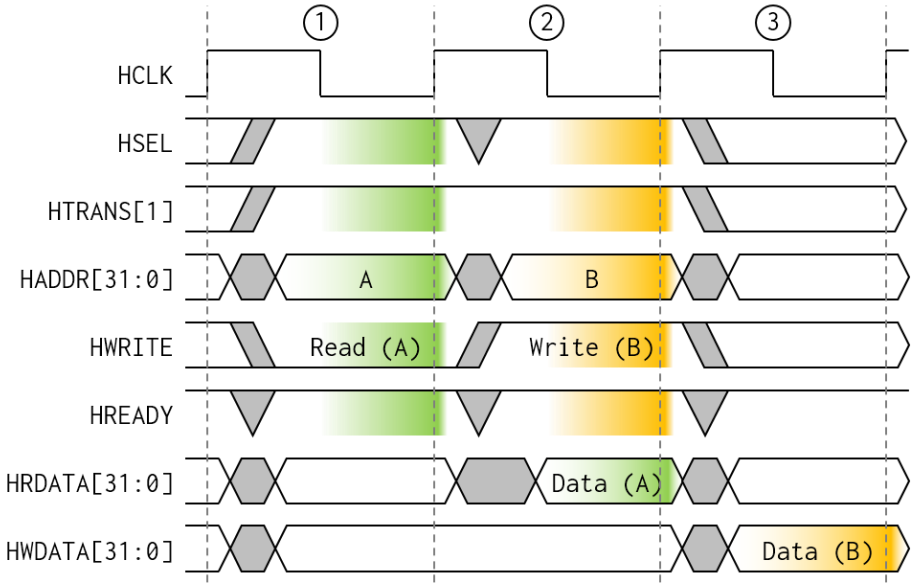
Puisque que le processeur supporte les opérations mémoires sur 8, 16 et 32 bits, trois compteurs sont utilisés pour les discriminer. Pour cela, le vecteur *HSIZE[2:0]* est utilisé :

$$RAM8bitWrite = RAMWrite \cdot (HSIZE = 000_b) \quad 3\text{-}5$$
$$RAM16bitWrite = RAMWrite \cdot (HSIZE = 001_b) \quad 3\text{-}6$$
$$RAM32bitWrite = RAMWrite \cdot (HSIZE = 010_b) \quad 3\text{-}7$$

Les autres tailles de transfert, de 64 bits (*HSIZE[2:0]* = 011_b) à 1024 bits (*HSIZE[2:0]* = 111_b), ne sont pas supportées par le système bien qu'elles existent dans le protocole AMBA 3 AHB-Lite. Si l'une de ces valeurs est détectée, la mémoire génère une erreur sur le bus, ce qui fait entrer le processeur en état d'erreur matérielle (*HardFault*). Les mémoires ne supportent que des lectures de mots de 32 bits de large. Puisque le cas où *HPROT[0]* serait à l'état bas et *HWRTE* à l'état haut n'est pas possible théoriquement (une récupération d'instruction est par définition une lecture), ce cas n'est pas traité.

Le chronogramme de la Figure 3-5 montre l'état des signaux d'un bus type AHB-Lite lors d'une lecture et d'une écriture. *HCLK* est le signal d'horloge du bus. Le maître comme les esclaves sont sensibles au front montant de ce signal. Le vecteur *HADDR[31:0]* contient l'adresse désirée, et les vecteurs *HRDATA[31:0]* et *HWDATA[31:0]* sont respectivement le bus de donnée esclave-vers-maître et le bus de donnée maître-vers-esclave. Un transfert est composé de deux phases : une phase d'adresse et une phase de donnée. La phase de donnée intervient toujours après une phase d'adresse valide, aussi bien pour la lecture que pour l'écriture. Puisque les signaux utilisés lors de la phase de donnée sont indépendants de ceux utilisés lors de la phase d'adresse, il est possible, comme illustré par la Figure 3-5, que ces deux phases s'exécutent en parallèle.

Figure 3-5 : Chronogramme d'un transfert - Cycle (1) : phase d'adresse A ; cycle (2) : phase de donnée A et phase d'adresse B ; cycle (3) : phase de donnée B



Chaque compteur est de largeur 32 bits. Puisque ARM fixe la limite en fréquence du Cortex-M0 à 50 MHz, on peut en déduire qu'un dépassement de capacité des compteurs ne sera pas atteint avant 86 secondes environ.

f) Organisation mémoire

Pour simplifier le chargement du programme applicatif dans la mémoire, un programme d'amorçage est présent dans une mémoire morte de petite taille, inclus dans le système.

Lors de l'initialisation du système (suite à son premier démarrage ou à sa réinitialisation), le processeur récupère plusieurs informations qui se trouvent à des emplacements mémoires précis : la valeur initiale du pointeur de pile (SP, *Stack Pointer*, à l'adresse 0000 0000_h), l'adresse de la routine de démarrage (*Reset Handler*, à adresse 0000 0004_h), les adresses des routines des interruptions internes au processeur (de 0000 0008_h à 0000 $003C_h$) et celles des routines du vecteur d'interruptions (de 0000 0040_h à 0000 $00BC_h$). La région mémoire démarrant à l'adresse 0000 0000_h est virtuelle, et est configurée pour pointer soit vers la mémoire morte (démarrant à l'adresse 0800 0000_h), contenant le programme d'amorçage, soit vers la mémoire principale (démarrant à l'adresse 2000 0000_h), contenant les instructions du programme applicatif.

La configuration de cette région virtuelle se fait grâce à un registre de contrôle du périphérique RSTCLK (*Reset and clocks control*). Le changement de section n'est effectif que lors d'une réinitialisation du système demandé par logiciel. Tout autre ordre de redémarrage demandé par le matériel (premier démarrage ou par l'entrée dédiée) force la mémoire virtuelle à pointer vers la mémoire morte contenant le programme d'amorçage.

3.1.3. Interface logicielle

Puisque nous avons développé notre propre architecture de microcontrôleur, il est nécessaire de l'outiller pour pouvoir la contrôler depuis la partie logicielle. Nous avons donc développé notre propre bibliothèque logicielle pour simplifier le contrôle des différents composants internes, comme les périphériques.

a) Chargement des instructions

Aucune interface pour programmateur n'étant présente, nous avons développé un programme d'amorçage destiné à récupérer le programme applicatif sur l'un des périphériques UART. Puisque ce dernier nécessite la configuration du module de sélection des broches, le PPS, une procédure d'initialisation est requise au démarrage du microcontrôleur. Lors du premier démarrage du système, les instructions sont récupérées par le processeur depuis la mémoire morte. Lors de l'initialisation du système, le programme d'amorçage envoie quelques informations par la communication série (UART) : un octet de synchronisation (55_h, qui aide à détecter la vitesse de la transmission), 8 caractères au format ASCII[3] correspondants à la date et l'heure de la compilation du programme d'amorçage encodées dans un entier de 32 bits, et un dernier caractère ASCII indiquant la raison du démarrage. Chaque partie est espacée d'un caractère ASCII « saut de ligne » (LF, *Line Feed*) pour rendre les informations plus claires si elles sont affichées à l'aide d'un terminal. Une fois prêt, le système est en attente de messages sur la liaison série.

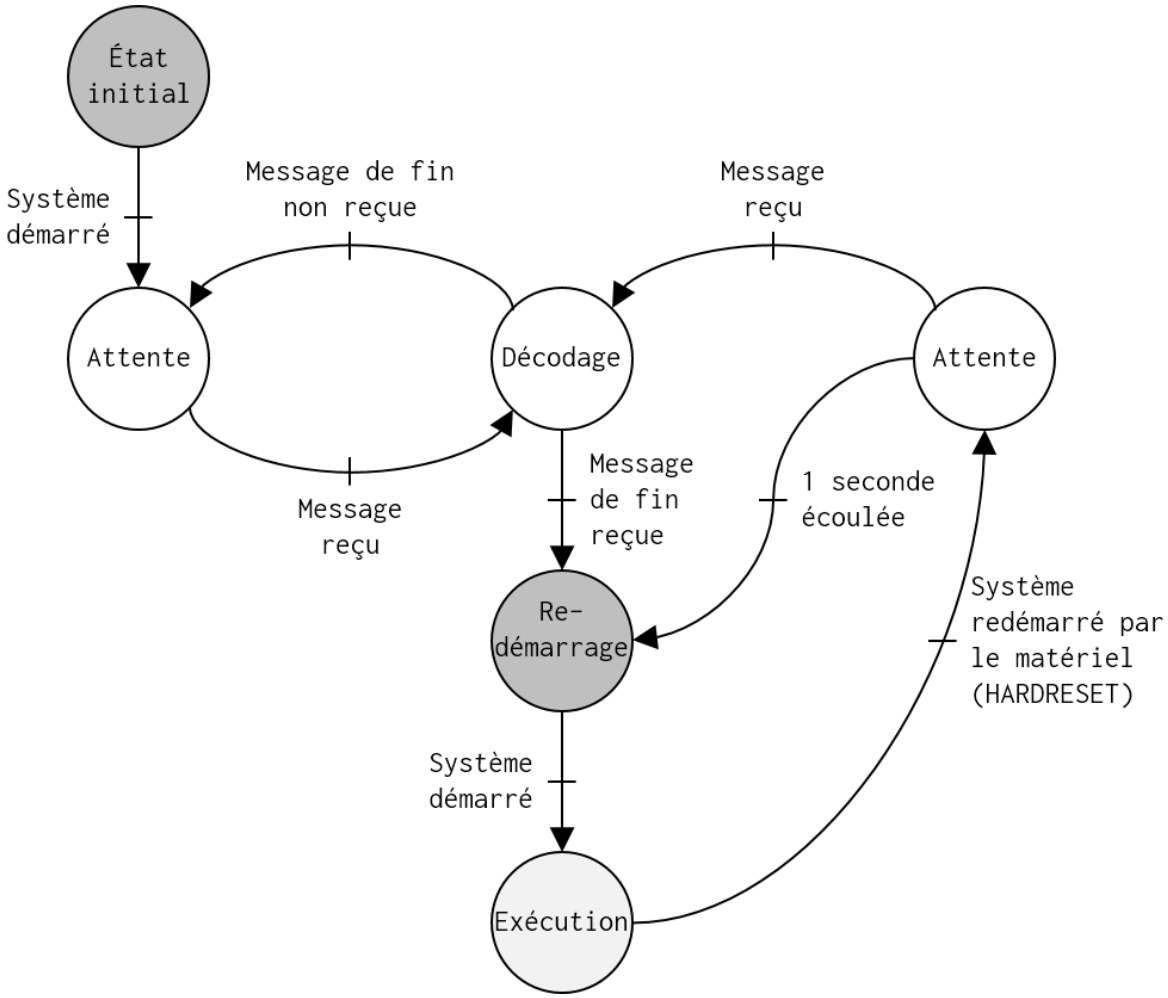
Les instructions du programme applicatif sont envoyées au programme d'amorçage par la liaison série, depuis un ordinateur. Nous avons développé sur ce dernier un logiciel qui découpe le programme applicatif compilé puis l'envoie en plusieurs messages sur la liaison série. Chaque message possède un code de détection d'erreur sur 8 bits (CRC, Contrôle de Redondance Cyclique ou *Cycle Redundancy Check*) et doit être acquitté par le programme d'amorçage pour s'assurer du bon chargement du programme applicatif. Une fois le chargement terminé, le programme d'amorçage configure la région mémoire virtuelle afin que le prochain démarrage se fasse sur la mémoire principale, puis va ordonner la réinitialisation du système. Celui-ci va alors repartir sur la mémoire principale et exécuter les instructions contenues dans celle-ci. La liaison série sert alors de canal de communication entre le microcontrôleur et l'ordinateur, notre logiciel servant d'interface.

Si, au cours du fonctionnement du programme applicatif, une réinitialisation du système est demandée par l'utilisateur en utilisant l'entrée dédiée (*HARDRESET*), le système redémarre sur la mémoire morte. Le programme d'amorçage est capable de détecter si un programme a déjà été chargé. Dans ce cas, il forcera le redémarrage sur la mémoire principale après une seconde écoulée si aucune communication série n'a été initiée au cours de ce temps d'attente. Cela permet de redémarrer le

[3] L'ASCII (*American Standard Code for Information Interchange*, code américain normalisé pour l'échange d'information) est une norme de codage de caractères sur des mots de 7 bits.

programme applicatif manuellement sans avoir à recharger ce dernier. Ce comportement est illustré par la Figure 3-6.

Figure 3-6 : Comportement du programme d'amorçage



b) Réinitialisation du système et amorçage

Il existe plusieurs sources, matérielles ou logicielles, qui peuvent être à l'origine d'une réinitialisation d'un système. La première source est tout simplement la mise sous puissance du microcontrôleur (POR, *Power-On Reset*). Un signal de réinitialisation doit être généré pour assurer qu'après cette mise sous puissance le système est bien dans son état initial. De même, lorsque le microcontrôleur est en fonctionnement, si la tension d'alimentation chute en dessous d'un certain seuil, un signal de réinitialisation est généré afin d'éviter les défaillances matérielles (BOR, *Brown-Out Reset*). L'utilisateur peut également demander une réinitialisation, de manière logicielle au travers de son application (*software reset*), ou matérielle grâce à une broche dédiée, disponible sur une grande majorité des microcontrôleurs du commerce. Un compteur nommé « chien de garde » (*watchdog*)

peut être utilisé comme source de réinitialisation du système. Ce compteur est utilisé pour remettre à l'état initial une application si celle-ci est restée inactive trop longtemps (due à une erreur d'implémentation de l'application, ou plus rare d'une défaillance matérielle ayant entraîné, volontairement ou non, un blocage de l'application), apportant une certaine sécurité de fonctionnement supplémentaire à l'application. Des variantes de ce compteur existent pour détecter des fonctionnements anormaux d'une application.

La réinitialisation du système, qu'elle soit d'origine matérielle ou logicielle, ainsi que la sélection de la mémoire utilisée lors de l'amorçage, sont gérées par le périphérique RSTCLK (*reset and clocks control*). Tout comme les autres périphériques du système, le composant RSTCLK possède une série de registres pour le contrôle et la visualisation de l'état du système après une réinitialisation, ainsi que l'activation des horloges des périphériques du microcontrôleur (*clock enable*).

Le premier registre permet de déterminer la raison du démarrage ou redémarrage du système (RSTSTATUS, *reset status register*). Pour notre microcontrôleur, trois sources produisant une réinitialisation sont implémentées : initialisation après mise sous tension (POR, le signal est nommé *PWRRESET*), broche dédiée (signal *HARDRESET*), et réinitialisation logicielle (signal *SOFTRESET*). Le compteur « chien de garde » n'a pas été implémenté, toutefois sa position existe dans le registre d'état.

Le second registre, dédié aux options d'amorçage (BOOTOPT, *boot option*), est divisé en deux champs : le premier, accessible en lecture et en écriture par le processeur, permet de sélection laquelle des mémoires sera utilisée pour le prochain amorçage ; le second, accessible uniquement en lecture par le processeur, indique quelle est la mémoire actuellement utilisée pour l'amorçage. Ce deuxième champ est remis à zéro lors d'une réinitialisation demandée par le matériel (premier démarrage, broche dédiée) et copie la valeur du premier champ (modifiée ou non par le processeur) lors d'une réinitialisation demandée par l'application (requête logicielle).

Les autres registres du composant RSTCLK sont utilisés pour l'activation des horloges des périphériques, afin de réduire la consommation d'énergie des éléments non utilisés.

c) Capture des événements

La gestion du moniteur d'activité est faite par le programme applicatif. Le démarrage et l'arrêt de la capture des événements, ainsi que la réinitialisation des compteurs d'événements se fait par l'écriture d'un registre de contrôle. Cela permet de réaliser la capture sur une partie de l'application et d'ignorer, par exemple, tous les événements dus à l'initialisation du système ou lors de l'envoi des résultats. Les registres contenant les valeurs des compteurs d'événements sont également accessibles depuis le programme applicatif. L'organisation des registres au sein du périphérique est détaillée dans le Tableau 3-4. La position 004_h et les positions allant de $02C_h$ à $3FC_h$ ne sont pas utilisées.

Tableau 3-4 : Organisation mémoire du moniteur d'activité

Offset	Register	Description
000_h	CR (on write)	Control Register
000_h	STATUS (on read)	Status register
004_h	-	-
008_h	CYCCR	Cycle Counter Register
$00C_h$	INSTCR	Instruction Counter Register
010_h	FETCHCR	Fetch Counter Register
014_h	RAMRBCR	RAM 8-bit Read Counter Register
018_h	RAMRHCR	RAM 16-bit Read Counter Register
$01C_h$	RAMRWCR	RAM 32-bit Read Counter Register
020_h	RAMWBCR	RAM 8-bit Write Counter Register
024_h	RAMWHCR	RAM 16-bit Write Counter Register
028_h	RAMWWCR	RAM 32-bit Write Counter Register

Le registre à la position 000_h est un registre spécial dont la fonction change selon l'opération effectuée sur celui-ci. Si le processeur y accède en lecture, le périphérique retournera une valeur détaillant l'état actuel du périphérique (*STATUS*). Si le processeur y accède en écriture, le comportement du moniteur d'activité sera affecté par la valeur transmise sur le bus. Chaque bit du mot correspond, si implémenté, à une action, comme détaillé dans le Tableau 3-5.

Tableau 3-5 : Registre de contrôle (CR, *Control Register*) du moniteur d'activité

Bit 31	**Bit 30**	**Bit 29**	**Bit 28**	**Bit 27**	**Bit 26**	**Bit 25**	**Bit 24**
U-0	U-0	U-0	U-0	U-0	U-0	U-0	U-0
-	-	-	-	-	-	-	-
Bit 23	**Bit 22**	**Bit 21**	**Bit 20**	**Bit 19**	**Bit 18**	**Bit 17**	**Bit 16**
U-0	U-0	U-0	U-0	U-0	U-0	U-0	S-0
-	-	-	-	-	-	-	RSTRAMWWC
Bit 15	**Bit 14**	**Bit 13**	**Bit 12**	**Bit 11**	**Bit 10**	**Bit 9**	**Bit 8**
S-0	S-0	S-0	S-0	S-0	S-0	S-0	S-0
RSTRAMWHC	RSTRAMWBC	RSTRAMRWC	RSTRAMRHC	RSTRAMRBC	RSTFETCHC	RSTINSTC	RSTCYCC
Bit 7	**Bit 6**	**Bit 5**	**Bit 4**	**Bit 3**	**Bit 2**	**Bit 1**	**Bit 0**
U-0	U-0	U-0	U-0	U-0	U-0	S-0	S-0
-	-	-	-	-	-	STOP	START

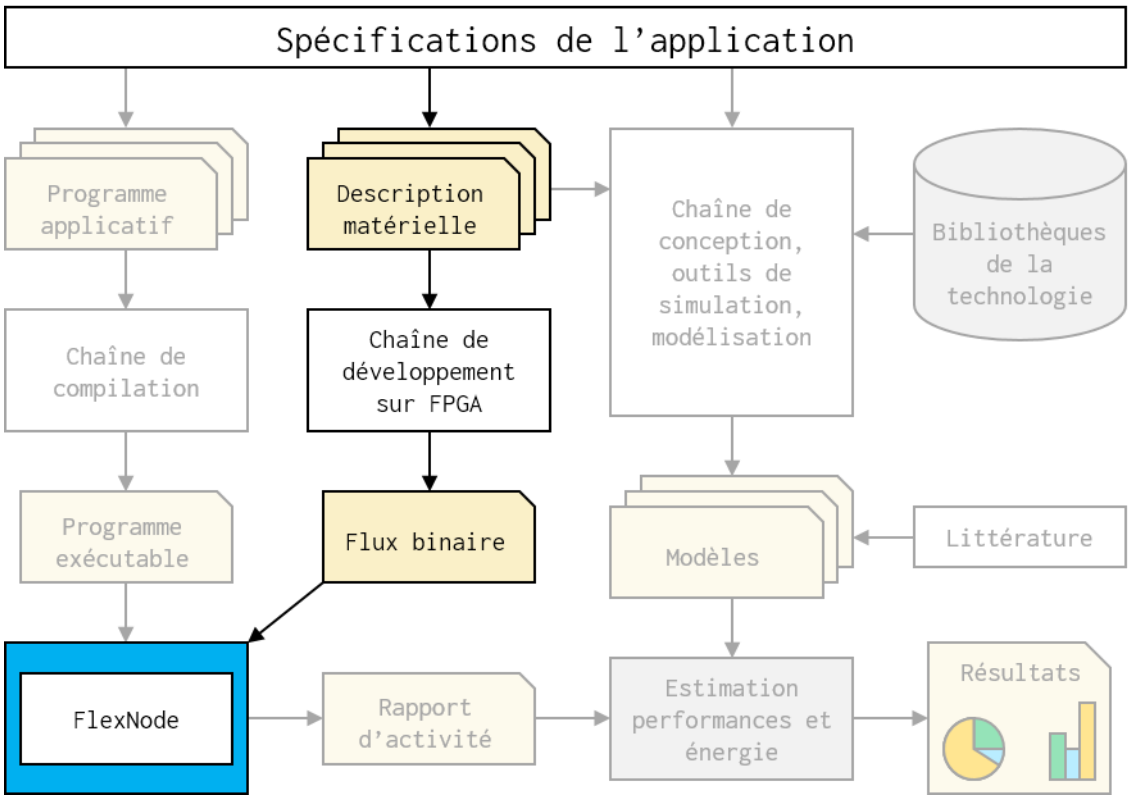
Ce registre de contrôle peut donc être « écrit » mais pas « lu ». Dans le cas où les deux commandes *START* et *STOP* sont présentes dans une même requête, bien que contraires, seule la commande *STOP* est exécutée, la commande *START* est ignorée. Les autres commandes permettent de remettre à zéro la valeur des différents compteurs du moniteur d'activité : *RSTRAM(a)(b)C* pour *reset RAM counter*, où *(a)* désigne le type d'opération (*W* pour *write* et *R* pour *read*) et *(b)* la taille

de la transaction (*B* pour *byte*, 8 bits ; *H* pour *half-word*, 16 bits ; *W* pour *word*, 32 bits) ; *RSTFETCHC* pour *reset fetches counter*, *RSTINSTC* pour *reset instructions counter* et *RSTCYCC* pour *reset cycles counter*.

3.2. Environnement de validation

Notre architecture de microcontrôleur est destinée aux applications de type nœud capteur sans fil. Les microcontrôleurs utilisés pour ces dispositifs sont bien souvent connectés à d'autres composants (capteurs, actionneurs, module radio...) et leur comportement est dépendant de ces derniers. Notre objectif étant de réaliser des évaluations d'architectures dans ce contexte applicatif, il est impératif de prendre en compte tous les aléas externes, bien souvent générés par les composants externes au microcontrôleur. Plutôt que de modéliser ces composants et de procéder à des simulations de ces derniers, nous avons choisis d'implémenter notre architecture de microcontrôleur sur un FPGA, qui fera le lien physique avec des composant réels. Cela nous permet la prise en compte les aléas externes, nécessaire pour réaliser nos évaluations dans un contexte application, sans avoir besoin de modéliser d'autres composants, un développement long et complexe.

Figure 3-7 : Flot d'évaluation - Implémentation sur FPGA

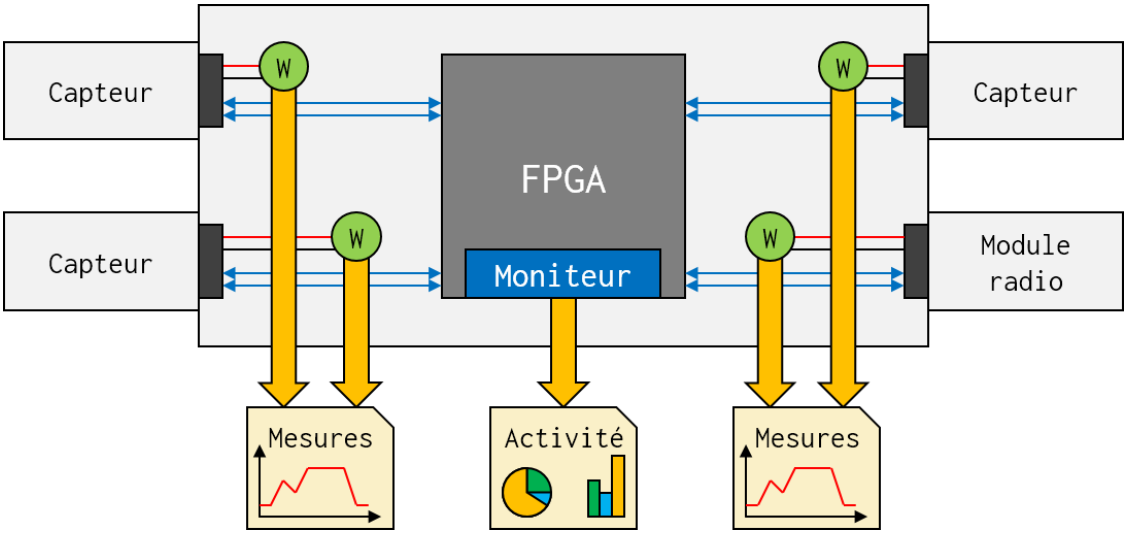


Pour réaliser l'évaluation de notre architecture, nous avons besoin d'obtenir l'activité de celle-ci lors de son fonctionnement. Cette activité est le résultat produit par le comportement de l'architecture pour une application donnée. Pour l'obtenir, nous avons choisi d'implémenter l'architecture sur un FPGA (Figure 3-7). Lui-même intégré sur une plateforme de prototypage de nœud, il est interfacé avec différents périphériques externes (module radio, capteur...). Cette plateforme permet de reproduire le comportement voulu en prenant en compte les interactions de notre architecture avec son environnement applicatif.

3.2.1. Plateforme de prototypage

Nous avons besoin une plateforme d'évaluation intégrant un FPGA, capable d'être interfacé avec des périphériques externes. Pour réaliser des évaluations au niveau applicatif, il est nécessaire d'avoir accès à la ligne d'alimentation de chaque module interfacé avec le FPGA afin de discriminer la consommation en énergie de chaque élément. S'il existe des cartes de prototypage FPGA intégrant de nombreux périphériques et disposant de connecteurs pouvant recevoir des modules externes, elles ne sont pas outillées pour réaliser les mesures de puissances individualisées. De plus, de nombreux périphériques présents par défaut sur les grandes cartes de prototypage ne sont pas intéressants pour notre usage. Nous avons donc réalisé autour d'une carte de prototypage FPGA simple une carte électronique de prototypage de nœud de capteur, dénommée FlexNode, pouvant accueillir différents modules comme des capteurs ou des moteurs radio grâce à des emplacements prévus sur celle-ci. Afin de discriminer la contribution de chaque composant sur la consommation d'énergie du nœud complet, chaque module possède sa propre ligne d'alimentation avec l'outillage nécessaire à la mesure de la puissance et donc à l'évaluation de l'énergie qu'il consomme. Cette carte électronique est dotée

Figure 3-8 : Schéma de la plateforme de prototypage de nœud capteur

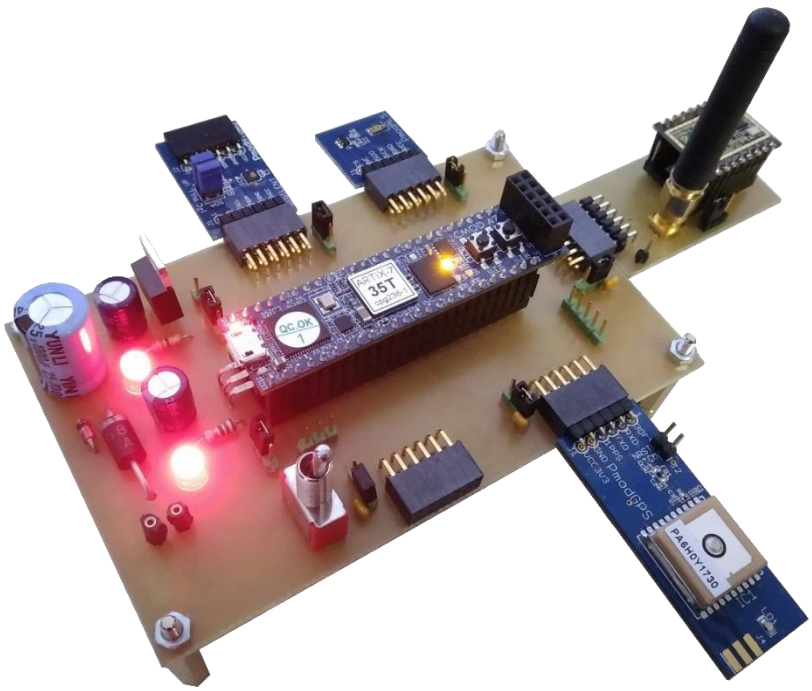


d'une batterie afin de facilement pouvoir l'intégrer dans le milieu d'évolution du nœud. Le FPGA, bien que reconfigurable à souhait, n'est pas soudé directement sur la carte mais est laissé détachable afin de pouvoir le remplacer par un autre contrôleur, par exemple par un microcontrôleur du commerce, et ainsi pouvoir comparer les performances d'une architecture avec celle d'un produit existant. Le principe est illustré par la Figure 3-8.

Pour réaliser à l'aide du FlexNode une estimation de la consommation en énergie d'une architecture ou une partie de celle-ci dans une technologie donnée, il est nécessaire de construire un modèle de consommation de la partie à étudier et d'instrumenter le FPGA pour récupérer l'activité de celle-ci, activité qui viendra alimenter le modèle. L'architecture de microcontrôleur présentée dans la partie précédente dispose d'un moniteur d'activité justement prévu pour relever n'importe quel événement interne. En sélectionnant les événements ayant une forte corrélation avec l'énergie consommée par ce contrôleur, ou une partie, et en combinant le rapport du moniteur d'activité avec le modèle énergétique dans la technologie souhaitée, on peut obtenir très rapidement une estimation de l'énergie consommée de la partie étudiée.

Le modèle énergétique d'un bloc ou d'une architecture peut être obtenu de différentes manières. Dans certains cas, il est possible d'obtenir directement ce modèle de par la littérature ou depuis un fabriquant ; ce dernier disposant généralement des modèles pour chaque cellule qu'il propose dans ses bibliothèques. Pour les éléments dont il n'existe pas de modèle, nous les réalisons à partir de

Figure 3-9 : La première version du FlexNode, avec au centre le Cmod A7



simulations plus fines du microcontrôleur dans la technologie voulue. On peut ainsi obtenir un modèle comportemental et énergétique d'un bloc numérique à partir des modèles des cellules qui le composent, disponibles dans la bibliothèque du fabriquant, et une série de simulations suffisamment complète pour couvrir le maximum des états possibles du bloc. De la même manière, il est possible d'obtenir un modèle pour un bloc analogique en faisant une évaluation au niveau transistor. Cette méthode comporte toutefois quelques limites : les estimations sont basées sur des modèles qui peuvent intégrer une erreur dans les résultats, augmentant ainsi l'erreur globale ; il n'est pas toujours possible d'évaluer tous les états que peut prendre un bloc car ils sont trop nombreux, ce qui est le cas pour des architectures complexes comme un processeur ; et les simulations à des niveaux plus fins sont bien plus lentes, ce qui complique la création de modèles pour les blocs complexes. Mais cela reste une méthode abordable pour obtenir un modèle énergétique suffisamment précis.

Pour la première version du FlexNode, les connecteurs utilisés pour les périphériques externes sont au format des Pmod de Digilent : en utilisant cette interface, il est possible de réutiliser les cartes proposées par cet intégrateur. Des points de mesures situés sur la ligne d'alimentation arrivant à chaque connecteur permettent la mesure du courant qui y circule ainsi que la mesure de la tension, pour obtenir la puissance et par extension l'énergie consommée par chaque périphérique. Deux régulateurs de tensions linéaires permettent d'obtenir une première ligne d'alimentation à 5 V pour le Cmod A7, et une seconde à 3,3 V à destination des périphériques. Un interrupteur permet de déconnecter la batterie du montage.

La Figure 3-9 illustre la première version du FlexNode, avec en son centre le Cmod A7, la carte FPGA sur laquelle est implémentée le microcontrôleur présenté précédemment. On distingue autour

Figure 3-10 : La seconde version du FlexNode, intégrant les périphériques



de la carte électronique quatre périphériques connectés aux emplacements prévus à cette intention. Dans le sens positif en partant du bas : un module GPS, un module de communication radio LoRa, un capteur de luminosité, et un capteur d'humidité et de température. La batterie qui sert à alimenter le FlexNode de manière autonome est située sous la carte (non visible sur la Figure 3-9).

Une seconde version du FlexNode, présentée par la Figure 3-10, intègre directement un capteur et un module radio sur la carte mère. Un commutateur de puissance permet de contrôler l'alimentation de ces deux périphériques par le FPGA (notre microcontrôleur).

3.2.2. Prototypage sur FPGA

Notre microcontrôleur est assez simple et doit fonctionner à des fréquences assez basses (ARM déclare une limite de 50 MHz pour le Cortex-M0) ; il est possible de l'implémenter sur FPGA.

Les Cmod A7 de Digilent sont une série de cartes de prototypage à 48 broches dotées d'un FPGA issu de la série Artix-7 de Xilinx. Le XC7A35T-1CPG236C présent sur le Cmod A7-35T possède 20800 LUT (*Look-Up Table*, table de correspondance), 41600 bascules (*D Flip-Flop*, DFF), 1800 kb de RAM et un convertisseur analogique-numérique (ADC, *Analog to Digital Converter*). La carte inclut une interface USB vers JTAG[4], un pont USB vers liaison série (UART), une source d'horloge à 12 MHz, une mémoire Flash de 4 Mo avec interface SPI et une SRAM asynchrone de 512 ko. Puisqu'un unique port USB est utilisé pour la programmation du FPGA et l'accès au pont série, il n'est pas nécessaire d'avoir recourt à du matériel externe pour communiquer avec le système.

La synthèse et l'implémentation du microcontrôleur sont réalisées avec l'outil Vivado fourni par Xilinx, dans sa version 2018.2. Les valeurs par défaut de l'outil ont été gardées pour les deux opérations. Si le cœur du microcontrôleur est entièrement décrit en langage VHDL, certains blocs, notamment les mémoires et les gestionnaires d'horloges, sont propres au FPGA et sont générés depuis l'outil Vivado. Bien qu'il soit possible de créer des modèles en langage HDL, ces blocs ne peuvent être synthétisés en utilisant des cellules standards, soit pour des soucis d'optimisation (par exemple, une banque mémoire nécessite une conception dédiée pour optimiser taille, consommation et performances), soit pour des contraintes technologiques (la gestion de l'horloge demande l'utilisation de cellules dédiées, parfois de circuits analogiques, propres à chaque technologie).

a) Blocs mémoires

Le FPGA XC7A35T présent sur le Cmod A7-35T possède 50 blocs de RAM de 36 kb, soit un total de 1 800 kb de capacité de RAM embarquée. Chaque bloc est une mémoire à deux ports qui peut être configurée en deux mémoires simple port de 18 kb (soit 100 blocs de 18 kb). Un bloc de 36 kb comporte 1024 mots de 36 bits, correspondant à 32 bits de donnée et 4 bits de parité (soit 1 par

[4] JTAG (*Join Test Action Group*) produit une interface unique pour le test, la programmation et le débogue appelé *Boundary-Scan*. Cette interface est standardisée (IEEE 1149.1) [89].

octet). L'écriture est individuelle pour chaque octet (avec bit de parité si utilisé), grâce à quatre lignes de commandes dédiées.

On utilise le générateur de Vivado pour produire trois mémoires à partir de ces blocs : une SPROM de 2 ko, correspondant à la ROM1, qui utilise un bloc de 18 kb et pré chargée avec le programme d'amorçage ; une SPRAM de 128 ko, correspondant à la RAM1, qui utilise 32 blocs de 36 kb et d'autres cellules nécessaires pour unifier la mémoire (LUT, multiplexeurs et bascules/verrous) ; et une SPRAM de 16 ko, correspondant à la RAM2, qui utilise 4 blocs de 36 kb. Chaque mémoire est configurée pour fournir la donnée demandée en un seul cycle (pas de cycle d'attente).

b) Réinitialisation

Les signaux utilisés pour la réinitialisation du système sont générés à partir de différents composants du microcontrôleur. Le signal de réinitialisation logicielle *SOFTRESET* provient directement du processeur et le signal *HARDRESET* (réinitialisation matérielle) provient d'un contrôleur de broche. Le signal *PWRRESET*, réinitialisation après la mise sous tension du microcontrôleur, représenté ici par la fin de la configuration de la partie logique du FPGA, est généré par un composant propre à l'implémentation FPGA, tout simplement nommé POR. Ce composant est conçu pour générer une requête de réinitialisation du système durant les premiers cycles, grâce à un registre dont la valeur est initialisée au chargement du FPGA.

Il s'agit du seul composant disposant de bascules avec une valeur initiale dans sa description en langage HDL. Toutes les autres bascules et registres du système sont réinitialisés au démarrage du microcontrôleur grâce au signal de réinitialisation ; elles n'ont donc pas de valeur initiale dans leur description, car pas nécessaires. De plus, les valeurs initiales sont prises en compte sur les implémentations FPGA et lors des simulations RTL, mais pas sur les implémentations ASIC ; elles sont donc une potentielle source d'erreurs lors de la conception d'un système numérique.

c) Génération et gestion du signal d'horloge

Un signal d'horloge de fréquence 12 MHz est généré par l'un des composants du Cmod A7, grâce à un oscillateur à quartz. Ce signal est récupéré par l'une des entrées du FPGA et peut être utilisé pour alimenter soit directement l'arbre d'horloge soit un gestionnaire d'horloges. Ces gestionnaires, appelés MMCM (*Mixed-Mode Clock Manager*), sont capables de produire un ou plusieurs signaux d'horloge à partir d'une entrée, de fréquence identique à celle-ci ou avec des coefficients de multiplication et de division. On retrouve dans ces gestionnaires d'horloge des diviseurs de fréquences et des boucles à verrouillage de phase (PLL, *Phase Locked Loop*), deux blocs également utilisés dans plusieurs microcontrôleurs du commerce. Pour cette première implémentation notre système sera cadencé à 12 MHz, il n'est donc pas nécessaire d'utiliser un gestionnaire d'horloge.

Pour réduire la consommation d'énergie du FPGA, il est possible de désactiver certaines parties du microcontrôleur en utilisant diverses méthodes. S'il n'est pas possible de contrôler individuellement l'alimentation des cellules inutilisées du FPGA, il est possible de réduire la puissance dynamique consommée par ces dernières en minimisant voir supprimant leurs changements d'état, grâce à des signaux d'activation d'horloge (*clock enable signals*, illustrés par la Figure 3-11 (a)) et/ou des signaux d'horloges commandés par porte (*gated clock*, illustrés par la Figure 3-12 (a)).

Figure 3-11 : (a) Exemple de circuit avec un signal d'activation d'horloge (*enable*) ; (b) Représentation de la bascule avec signal d'activation utilisée dans (a)

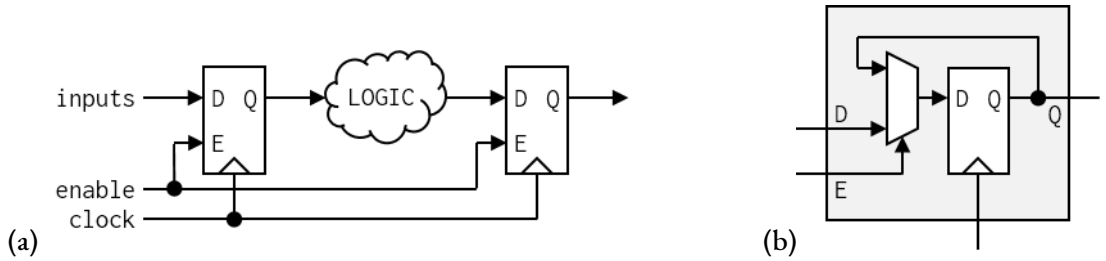
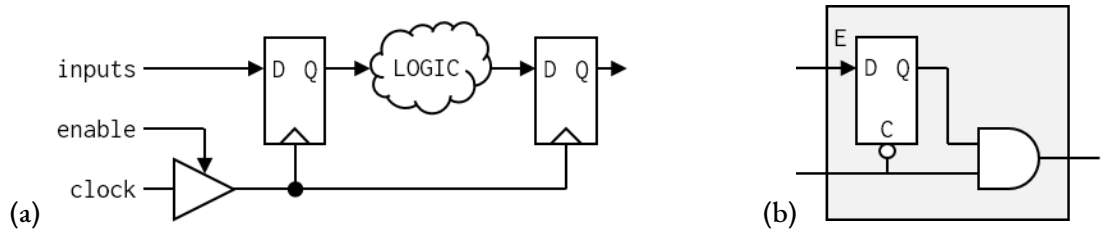


Figure 3-12 : (a) Exemple de circuit avec un signal d'horloge commandé par porte ; (b) Représentation de la cellule utilisée pour la commande du signal d'horloge du circuit (a)



Un signal d'activation d'horloge est un signal qui autorise ou non le rafraîchissement d'une bascule lorsque celle-ci est activée par un signal d'horloge (ce comportement est illustré par la Figure 3-11 (b)). Les parties logiques dont les signaux d'entrée ne proviennent que de sorties de bascules désactivées ne subissent plus de changement d'état, et ne consomment donc plus l'énergie nécessaire à ces changements. Toutefois, une bascule désactivée consomme toujours une certaine quantité d'énergie lors des changements d'état du signal d'horloge.

Pour réduire grandement la consommation dynamique d'une bascule, on peut utiliser un signal d'horloge commandé par porte. Une bascule ne reçoit plus de front lorsque son signal d'horloge est désactivé, et continue de maintenir son signal de sortie à la même valeur, ne stimulant alors plus les portes logiques qu'il commande. Il est également possible de lier plusieurs bascules à un même signal d'horloge commandé par porte, ce qui permet d'optimiser l'espace et la consommation d'énergie. Si l'utilisation de cette méthode permet une meilleure réduction de la consommation que l'utilisation

de signaux d'activation d'horloge [90], elle nécessite la génération d'un arbre d'horloge avec des contraintes de conception adaptées et en usant des cellules dédiées. La Figure 3-12 (b) représente une cellule de commande, composée d'une porte « ET » et d'un verrou. Ce dernier assure que le changement d'état du signal d'entrée soit reporté à l'état bas de l'horloge maître, garantissant un signal propre en sortie de la porte logique. En effet, l'insertion d'éléments logiques au sein du réseau de distribution de l'horloge peut entraîner une désynchronisation entre ses signaux, et par conséquent une possible source de violation des contraintes de temps de mise en place (*setup time*) et de maintien (*hold time*) des signaux d'entrée des bascules.

Le réseau de distribution d'horloge du XC7A35T présent sur le Cmod A7-35T est conçu pour pouvoir commander des signaux d'horloges par la logique tout en garantissant la synchronisation de ces derniers, grâce à une architecture dédiée [91]. Une première version du microcontrôleur a été réalisée avec des signaux d'horloges commandés par porte (un par périphérique, à l'exception du module *RSTCLK* qui permet la gestion de ces signaux, soit un total de 23 lignes commandées, en plus du signal d'horloge principal non commandé par la logique). Cependant, pour des soucis d'adaptation qui seront détaillés dans la section suivante, la version actuelle du microcontrôleur utilise des signaux d'activation d'horloge plutôt que des signaux d'horloges commandés par porte.

3.2.3. Utilisation des ressources

Le Tableau 3-6 présente les ressources du FPGA nécessaires pour implémenter notre microcontrôleur. 73% de la mémoire RAM embarquée disponible est nécessaire pour générer les différentes mémoires. Environ un tiers des LUT est utilisé, et plus de 90% des registres sont inutilisés. Même si le CmodA7 utilisé n'intègre pas le FPGA possédant le plus de ressources, nous constatons qu'il reste encore beaucoup de ressources disponibles, indiquant d'une part que des évolutions de l'architecture sont possibles (par exemple l'ajout d'accélérateurs matériels ou de nouveaux périphériques), et d'autre part que notre plateforme de prototypage et d'évaluation est relativement peu gourmande en terme de ressources.

La Figure 3-13 présente le schéma fonctionnel du microcontrôleur implémenté sur le FPGA. On retrouve sur celui-ci le processeur, les trois mémoires (*ROM1*, *RAM1* et *RAM2*) et leurs contrôleurs, le moniteur d'activité (*Activity Monitor*), le gestionnaire de réinitialisation et des horloges (*RSTCLK*) et les périphériques (4 compteurs, 4 UART, 4 SPI, 4 I^2C, 4 GPIO et les sélecteurs PPSIN et PPSOUT). Ces derniers permettent au microcontrôleur de communiquer avec les différents composants externes (capteurs, modules radio...) des plates-forme de prototypage, permettant ainsi d'évaluer une architecture dans son environnement applicatif. Le moniteur d'activité permet de récupérer des informations relatives à l'activité de la mémoire RAM1, informations nécessaires pour réaliser des estimations de consommation énergétique de celle-ci. L'utilisation du FPGA nous permet également d'atteindre n'importe quel signal du microcontrôleur et de le capturer avec le moniteur

d'activité. Ainsi, même si ce dernier ne prend en compte un nombre d'événements restreint, il peut être ajusté par la suite selon nos besoins.

Figure 3-13 : Schéma fonctionnel du microcontrôleur validé sur FPGA

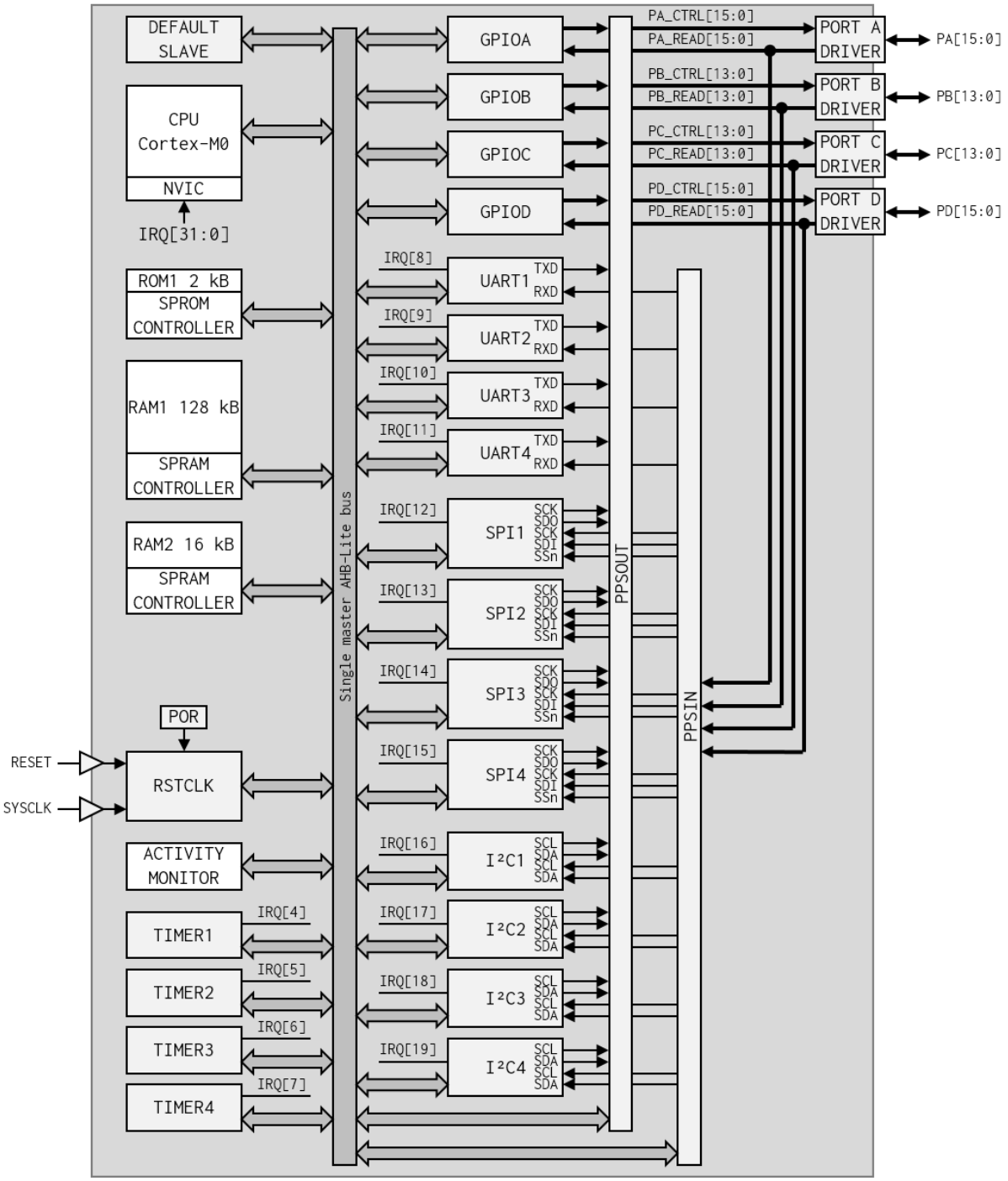


Tableau 3-6 : Resources du FPGA utilisé par le microcontrôleur

	Slice Type	Used	Available	Ratio
Full system	Slice LUTs	6 809	20 800	32.74%
	Slice Registers	4 081	41 600	9.81%
	Block RAM Tile	36.5	50	73.00%
ARM Cortx-M0	Slice LUTs	3 224	20 800	15.50%
	Slice Registers	903	41 600	2.17%
	Block RAM Tile	0	50	0.00%
Activity Monitor	Slice LUTs	296	20 800	1.42%
	Slice Registers	372	41 600	0.89%
	Block RAM Tile	0	50	0.00%

3.3. Implémentation en technologie FD-SOI 28 nm

Bien qu'il n'y a pas encore de microcontrôleur très-basse puissance commercialisé dans la technologie FD-SOI en 28 nm de STMicroelectronics, elle apporte plus de flexibilité que les technologies traditionnelles, en particulier pour l'efficacité énergétique des systèmes à laquelle nous portons un grand intérêt. Vue comme une alternative prometteuse pour les futures générations de microcontrôleurs, nous utilisons cette technologie pour notre microcontrôleur dans le but d'évaluer les possibilités qu'elle peut apporter, avec la technologie de mémoire magnétique de type STT, aux applications embarquées de l'Internet des Objets.

Notre architecture de microcontrôleur étant implémentée sur FPGA, il nous faut créer le modèle énergétique de celle-ci pour pouvoir réaliser une évaluation en terme d'énergie. Ces modèles sont réalisés non pas à partir des données technologiques du FPGA, mais à partir de celles de la technologie FD-SOI 28 nm de STMicroelectronics. La description matérielle de l'architecture est la même pour l'implémentation FPGA que pour l'implémentation ASIC d'un point de vue fonctionnel ; seules les bibliothèques utilisées sont différentes (Figure 3-14). Si les caractéristiques et les modèles nécessaires pour l'évaluation de certains composants du circuit sont disponibles dans la littérature, la maîtrise de la technologie FD-SOI nous permet d'obtenir des modèles de consommation des parties personnalisées. Pour cela, nous mettons en œuvre les outils de conceptions classiques utilisés dans la réalisation de circuits intégrés (synthèse, placement, routage...).

Nous avons eu la possibilité de faire fabriquer des circuits dans la technologie du STMicroelectronics. Pour des raisons de coût, trois projets indépendants les uns des autres ont été implémentées sur un même circuit, avec des domaines d'alimentations et d'horloges différents. La surface minimale est de 1,25 mm^2, avec une optimisation de l'implémentation de notre microcontrôleur (le projet le plus volumineux du circuit).

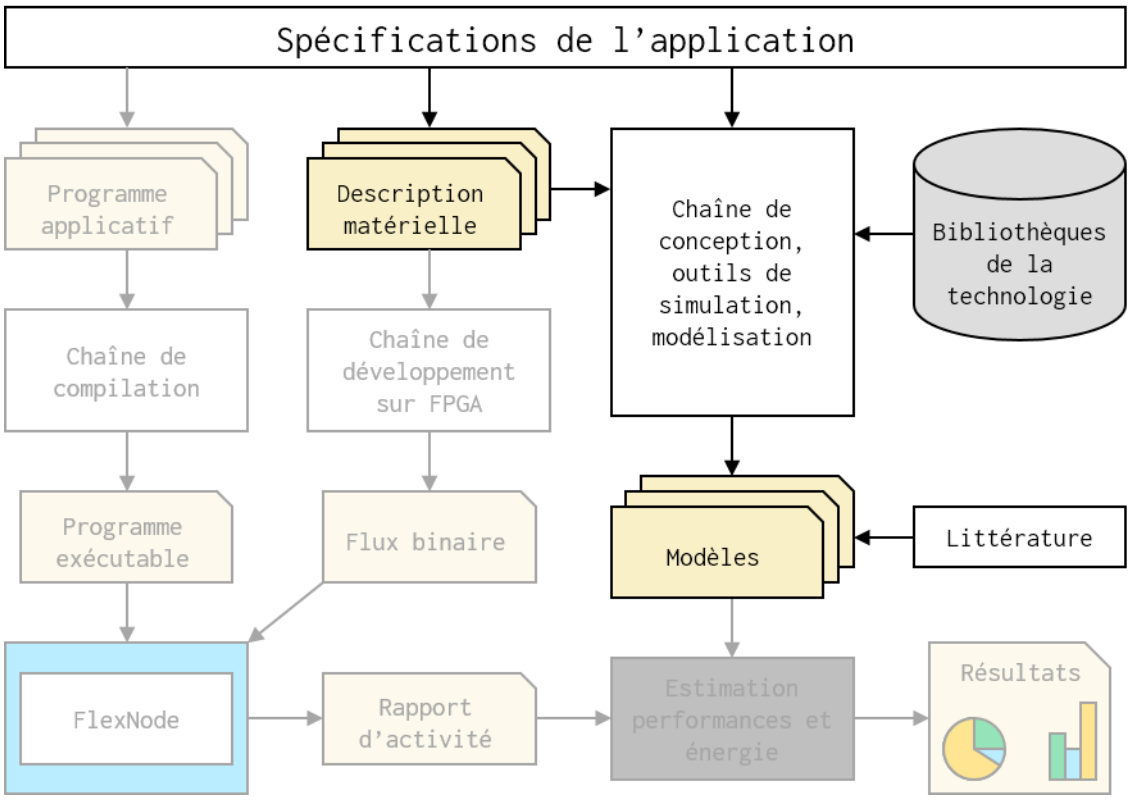
3.3.1. Cellules propres à la technologie

Quelques éléments de l'architecture ont été adaptés pour répondre aux contraintes liées à la technologie FD-SOI, en particulier pour l'utilisation des blocs propres à chaque fabricant tels que les mémoires, ou encore le circuit de réinitialisation.

a) Mémoires

Si des modèles de mémoires peuvent être décrits en langage HDL, la complexité de ces blocs fait qu'ils ne sont pas synthétisables avec les outils numériques classiques, et une implémentation basée sur des cellules standards serait bien moins optimisée en surface et en consommation d'énergie, et moins performante qu'une implémentation manuelle. Toutefois, les concepteurs de mémoires possèdent leurs propres outils pour la génération assistée de ces blocs, appelés compilateurs mémoire. Il est possible de paramétrer le compilateur pour activer ou désactiver certaines options, telles que la largeur du mot, le nombre de mots (définissant ainsi la capacité avec la largeur du mot), les options d'économie d'énergie (par exemple : mode rétention, commutateur de puissance, rails d'alimentation séparés), le nombre d'interfaces ou encore le masque d'écriture, qui seront alors inclus dans le bloc

Figure 3-14 : Flot d'évaluation - Modélisation



généré ou non. Cependant, il n'est pas toujours possible d'obtenir la capacité ou les options souhaitées en fonction de la configuration demandée.

Les blocs mémoires utilisés dans notre implémentation sont conçus et fabriqués par STMicroelectronics. La mémoire morte (ROM1), qui contient le programme d'amorçage, est une mémoire morte simple port (SPROM, *Single Port ROM*) de 2 ko (512 mots de largeur 32 bits). N'étant pas possible de générer une mémoire de 128 ko pour la RAM1, celle-ci est composée de deux banques mémoires simple port (SPRAM, *Single Port RAM*) de 64 ko (16 384 mots de 32 bits), avec rails d'alimentation séparés et masque d'écriture.

Si pour les deux premières mémoires, ROM1 et RAM1, les paramètres demandés au fabricant aboutissent à un choix unique d'implémentation, la mémoire de 16 ko nommée RAM2 peut être implémentée avec plusieurs configurations différentes : type de mémoire (SRAM ou banque de registres) et nombre de colonnes multiplexées différents. Quatre configurations sont disponibles : SRAM avec multiplexage par groupe de 8, 16 ou 32 colonnes et banque de registre (nommée SPREG) avec multiplexage par groupe de 8 colonnes. À partir des données disponibles dans les spécifications techniques de ces mémoires, une comparaison a été faite pour déterminer quelle configuration prendre. Des estimations de consommations ont été faites pour différents programmes de référence : le CoreMark, le CoreProfile, et un troisième programme plus léger réalisé par nos soins. La SRAM à multiplexeurs à 32 voies est 40% plus volumineuse que les autres banques mémoires et d'un point de vue énergétique elle n'est pas intéressante (courant de fuite en mode rétention plus grand que les autres choix, énergies d'écriture et de lecture également plus grandes). La SRAM à multiplexeurs à 16 voies est plus petite que la SPREG mais sa consommation d'énergie est plus grande que les deux choix restants. Parmi les deux solutions restantes, la SRAM est plus petite, son courant de fuite est légèrement plus faible et elle nécessite moins d'énergie pour les opérations d'écriture que la SPREG, mais elle en nécessite plus pour les opérations de lecture. Ces deux solutions sont intéressantes et la consommation globale d'énergie pour chaque application est très proche. Nous avons donc choisi d'implémenter les deux mémoires.

L'ajout d'une nouvelle mémoire dans le microcontrôleur implique une modification dans l'architecture mémoire. Cette nouvelle mémoire, nommée RAM3, est ajoutée à la suite de la RAM2 du point de vue logiciel. Elle est intégrée au bus par l'intermédiaire d'un contrôleur de mémoire (le même que pour les autres RAM) et le décodeur d'adresse est modifié pour sélectionner ce nouveau contrôleur pour la plage d'adresses de 2002 4000_h à 2002 $7FFF_h$.

b) STT-MRAM

Grâce à notre partenaire SPINTEC, nous disposons également des données d'une banque STT-MRAM de 128 ko avec une interface compatible SRAM supportant des transactions de données de largeur 32 bits. Bien qu'il ne nous est pas possible actuellement de la faire fabriquer, nous disposons d'informations sur celle-ci, en particulier des résultats de simulations qui nous permettent de la

caractériser et donc de réaliser des estimations de consommation avec. Les cellules magnétiques sont des jonctions magnétiques perpendiculaires de diamètre 40 nm et la partie logique qui les contrôle est réalisée dans la technologie FD-SOI 28 nm. Elle est compatible avec notre microcontrôleur et peut être utilisée en remplacement de la SRAM qui sert de RAM1, leurs interfaces étant similaires et leurs performances suffisantes pour nos contraintes (50 MHz). Le Tableau 3-7 résume les caractéristiques de la mémoire réalisée par SPINTEC.

Tableau 3-7 : Caractérisiques de la banque STT-MRAM de SPINTEC

Technologie CMOS	FD-SOI 28 nm
Jonction magnétique	pSTT Ø 40 nm
Architecture	1T-1MTJ
Nombre d'E/S	32
Alimentation	1,0 V
Capacité	128 ko (1 Mb)
Temps de lecture	5 ns (200 MHz)
Temps d'écriture	10 ns (100 MHz)
Énergie de lecture	0,9 pJ/bit
Énergie d'écriture	3,0 pJ/bit

c) Entrées/sorties

Pour respecter des contraintes d'espace (et indirectement de coût du projet), le nombre de broches du microcontrôleur a été défini à 18. En effet, les plots sont larges et affectent directement la forme et la surface du circuit. 16 entrées/sorties sont manipulables par les périphériques, une entrée est dédiée au signal de réinitialisation et une dernière entrée est dédiée au signal d'horloge. Un seul périphérique GPIO est suffisant pour contrôler les 16 broches. Les périphériques de communication de type SPI sont au nombre de deux, de même pour le nombre de périphériques I^2C ; quatre périphériques UART sont disponibles dans notre microcontrôleur. Les modules PPSIN et PPSOUT, le bus AHB-Lite et le décodeur d'adresse sont adaptés pour cette configuration.

Les contrôleurs d'entrée/sortie disponibles dans la bibliothèque de STMicroelectronics offrent bien plus d'options que ceux présents dans le FPGA. En dehors de l'interface de test, l'activation de résistances de tirage et de rappel, le réglage du cycle d'hystérésis et l'ajustement du courant dans l'étage de sortie sont possibles avec ces contrôleurs. Ces options sont très intéressantes et utiles, et pourraient être facilement commandées par un simple registre dans le périphérique GPIO. Cependant, nous avons décidé de ne pas les intégrer afin d'obtenir le microcontrôleur le plus proche possible de celui implémenté sur FPGA, et d'éviter l'ajout de fonctionnalités non vérifiées pour notre premier circuit dans cette technologie. Elles pourront être ajoutées dans une future version. Les options inutilisées sont donc désactivées ou forcées en fixant leur signal d'entrée au niveau logique adéquat. Certains

signaux sont fixés ou non selon la fonction désirée, entrée ou bidirectionnelle. Le Tableau 3-8 détaille les signaux de ces contrôleurs et leur configuration.

Tableau 3-8 : Configuration des contrôleurs d'entrées/sorties

Signal	Description	Utilisé	Niveau fixe
A	Entrée	Mode bidirectionnel uniquement	Bas pour les entrées simples
EN	Activation de l'entrée (actif à l'état bas)	Mode bidirectionnel uniquement	Haut pour les entrées simples
TM	Mode test (actif à l'état haut)	Non	Bas
TA	Entrée en mode test	Non	Bas
TEN	Activation de l'entrée en mode test (actif à l'état bas)	Non	Haut
PUN	Résistance de tirage (actif à l'état bas)	Non	Haut
PDN	Résistance de rappel (actif à l'état bas)	Non	Haut
PUN15K	Résistance de tirage de 15 kΩ (actif à l'état bas)	Non	Haut
HYST	Hystérésis (actif à l'état haut)	Non	Bas
LOWEMI	Limitation du courant de sortie (actif à l'état haut)	Non	Bas
ZI	Sortie	Oui	-
IO	Connection broche	-	-

d) Réinitialisation

L'initialisation du microcontrôleur après sa mise sous tension (POR) nécessite un composant analogique dédié, de même pour la détection d'une chute de la tension d'alimentation (BOR). Puisque notre conception n'est composée que de cellules standards numériques, il n'est pas possible d'implémenter en interne de telles fonctionnalités. Toutefois, un circuit analogique externe au microcontrôleur peut être utilisé pour générer une réinitialisation du système si la tension d'alimentation est trop basse.

Pour ne pas ajouter une entrée supplémentaire, la broche dédiée à la réinitialisation sera aussi utilisée pour les fonctions POR et BOR. Elle est donc câblée sur le signal *PWRRESET* ; et le signal *HARDRESET* initialement réservé pour cette entrée est fixé à l'état haut, état inactif.

e) Génération du signal l'horloge

Les microcontrôleurs du commerce possèdent typiquement une ou plusieurs sources d'horloge. Les plus communes sont : l'oscillateur à filtre RC (Résistance Condensateur), un oscillateur qui peut être entièrement intégré au microcontrôleur, consomme peu d'énergie mais peu stable en fréquence face aux variations de température ; l'oscillateur à quartz, très précis et très stable face aux variations

de température, mais plus gourmand en énergie qu'un oscillateur RC et nécessitant généralement des composants externes, notamment le quartz, les condensateurs et parfois d'autres composant passifs ; ou une source externe (à l'instar du FPGA) nécessitant un montage externe au microcontrôleur, souvent utilisé lorsque qu'une même source d'horloge peut être partagée entre plusieurs circuits intégrés ou montages pour optimiser un produit. En plus de ces sources d'horloge, la plupart des microcontrôleurs du commerce possèdent également des diviseurs de fréquences et des PLL pour adapter la fréquence du signal d'horloge aux besoins de l'application.

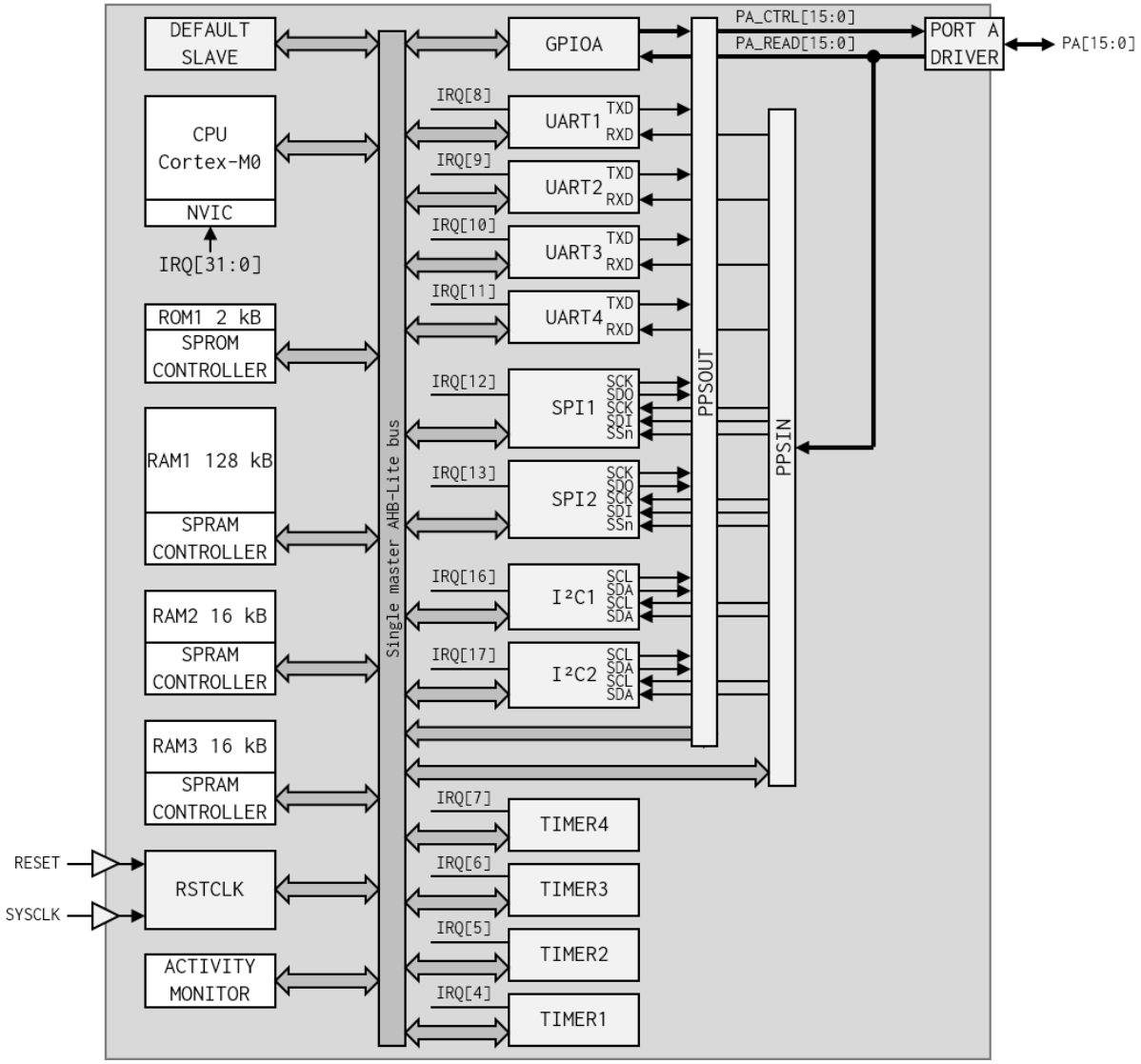
S'il existe de nombreuses solutions pour générer (oscillateur à quartz, à filtre RC, en anneau…) ou modifier (diviseur de fréquence, PLL…) un signal d'horloge, ces blocs nécessitent l'intégration d'une partie analogique ou d'une technologie particulière dans le microcontrôleur or, pour des raisons de simplicité et de temps, nous avons choisi de ne pas intégrer ces types de blocs et d'utiliser plutôt une source d'horloge externe, qui, par exemple, peut être fournie par un microcontrôleur du commerce, par un oscillateur intégré ou par un générateur de fonctions. La fréquence nominale, c'est-à-dire la fréquence attendue par le programme d'amorçage, est fixée à 32 MHz.

Le Tableau 3-9 résume les caractéristiques des deux implémentations du microcontrôleur, l'une sur FPGA et l'autre dans la technologie FD-SOI 28 nm de STMicroelectronics, dont le schéma fonctionnel est présenté par la Figure 3-15.

Tableau 3-9 : Caractéristiques des implémentations FPGA et ASIC du microcontrôleur

	FPGA	**ASIC**
Technologie	28 nm HKMG	FD-SOI 28 nm
Architecture mémoire	SPROM 32×512 (2 ko ROM1) SPRAM 32×32768 (128 ko RAM1) SPRAM 32×4096 (16 ko RAM2)	SPROM 32×512 (2 ko ROM1) SPRAM 2×32×16384 (128 ko RAM1) SPRAM 32×4096 (16 ko RAM2) SPREG 32×4096 (16 ko RAM3)
Entrées/Sorties	60	16
Périphériques	Compteur ×4 UART ×4 SPI ×4 I^2C ×4 PPS Moniteur d'activité	Compteur ×4 UART ×4 SPI ×2 I^2C ×2 PPS Moniteur d'activité
Signaux de réinitialisation	POR HARDRESET SOFTRESET	HARDRESET (implique POR) SOFTRESET
Gestion du signal d'horloge	Source externe, gestion interne (MMCM avec PLL)	Externe

Figure 3-15 : Schéma fonctionnel du microcontrôleur implémenté



3.3.2. Réalisation du circuit

Pour la synthèse et l'implémentation du microcontrôleur, une version offusquée de la bibliothèque FD-SOI 28 nm de STMicroelectronics est utilisée (version 10a). En effet, elle a été obtenue depuis un centre multi-projets qui délivre pour des raisons de confidentialité une version limitée de la bibliothèque du fabricant, où certaines cellules ne sont pas disponibles et où les couches les plus basses des transistors (RX et PC) ont été retirées. Bien qu'il soit toujours possible de réaliser le circuit et d'effecteur des simulations temporelles et des estimations de consommation énergétique, une partie des outils de vérification d'erreurs ne peuvent fonctionner normalement avec cette bibliothèque, et il devient difficile dans les dernières étapes de la conception de faire le tri entre des erreurs réelles de conception et des erreurs dues à l'offuscation.

La synthèse, le placement et le routage sont réalisés avec les outils de Cadence Design Systems : la synthèse avec Genus version 18.11, le placement et le routage avec Innovus 18.11. Les ajustements après routage sont réalisés avec Virtuoso, un autre outil de Cadence, dans sa version 6.1.7. Une contrainte en fréquence de 50 MHz est appliquée pour correspondre aux limitations du Cortex-M0 données par ARM.

a) Choix des cellules

Sont présentes dans la bibliothèque 28 nm FD-SOI de STMicroelectronics plusieurs catégories de cellules standards, distinguées en particulier par le type de transistor utilisé [92] : transistor à tension de seuil bas (LVT, *Low Voltage Threshold*) et transistor à tension de seuil régulière (RVT, *Regular Voltage Threshold*, parfois SVT, *Standard Voltage Threshold*). Les cellules de la bibliothèque LVT sont plus rapides mais consomment plus d'énergie à chaque transition que celles provenant de la bibliothèque RVT. Pour la réalisation d'un circuit orienté basse consommation, le choix se portera naturellement vers les cellules de la bibliothèque RVT. Dans certaines technologies, une autre catégorie de cellules standards utilisant des transistors à tension de seuil élevée (HVT, *High Voltage Threshold*) est disponible, mais ce n'est pas le cas de la bibliothèque que nous utilisons. La largeur du canal des transistors (*Poly-Biasing*) fait partie des paramètres de transistor disponibles, mais nous gardons toutefois la largeur nominal (PB0) pour les cellules standards de notre microcontrôleur.

Les blocs en dehors de la bibliothèque de cellules standards, en particulier les plots et les mémoires, ne sont implémentés qu'avec un seul type de transistor et ne sont pas alimentés de la même manière que le reste du circuit. Une alimentation séparée ainsi qu'une isolation du substrat sont nécessaires pour faire cohabiter les différents éléments.

b) Ajustements après synthèse

Les lignes réalisant la connexion entre les entrées et les sorties des contrôleurs de plots et les cellules logiques de la partie numérique sont très longues, certaines traversent presque la totalité du circuit. Les étapes de placement et de routage du microcontrôleur sont réalisées sans les plots ; ces lignes ne sont donc pas complètes à ces étapes là et l'outil n'en est pas informé. Des erreurs d'antennes sont alors relevées durant l'étape de vérification, une fois le microcontrôleur intégré dans le circuit final. Pour limiter au mieux ces problèmes d'antennes, on choisit d'insérer une diode de protection contre les effets d'antenne pour chaque ligne.

Les contrôleurs de plots doivent être entièrement configurés et un niveau logique doit être attribué pour chacune de leurs entrées. Puisque les fonctionnalités disponibles des plots ne sont pas toutes utilisées, il est nécessaire d'attribuer un niveau logique constant sur les entrées inutilisées. Ces niveaux logiques sont fixés avec des cellules d'attache (*tie cells*), soit au niveau haut (*tie high*), soit au niveau bas (*tie low*) en fonction de la configuration désirée.

Une routine écrite en langage Python sert à construire une liste d'interconnexions à partir de celle générée par l'outil de synthèse. Cette nouvelle liste, intégrant le microcontrôleur décrit dans la première ainsi que toutes les cellules nécessaires à la configuration des contrôleurs de plots et les diodes de protection, est ensuite fournie à l'outil réalisant les étapes de placement et de routage. La configuration des plots peut être modifiée de manière générale ou individuellement grâce à un fichier de configuration donné à la routine Python.

Par plot, une seule cellule d'attache au niveau haut et une seule cellule d'attache au niveau bas sont générées, avec une diode de protection pour chaque niveau. Une diode de protection est également ajoutée pour chaque ligne liant la logique du microcontrôleur aux plots : une si le plot est fixé en entrée (ligne de lecture) ; deux s'il est fixé en sortie (lignes d'écriture et de lecture) ; ou trois s'il est mode bidirectionnel (lignes de commande, d'écriture et de lecture).

c) Placement

La routine Python génère également le fichier de placement des entrées et sorties du composant « microcontrôleur » en suivant des contraintes spécifiées dans un second fichier de configuration (largeur des connexions, couche de métal désirée, écartement entre les connexions d'un même groupe, écartement entre les groupes, positionnement des groupes autour du circuit...). Cette routine génère aussi les ressources nécessaires à la simulation du microcontrôleur après son routage, prenant en compte les modifications apportées par la nouvelle liste d'interconnexions.

Bien que l'outil de placement soit capable de positionner automatiquement toutes les cellules déclarées dans les listes d'interconnexions, les cellules mémoires sont placées manuellement pour optimiser l'espace mais aussi pour assurer que les contraintes liées à l'alimentation et à l'isolation de ces mémoires soient respectées. Leur placement est visible dans la Figure 3-16. Les deux banques de la RAM1 ont été regroupées ensemble ; la ROM1 est située au plus proche de la partie numérique car partageant la même alimentation.

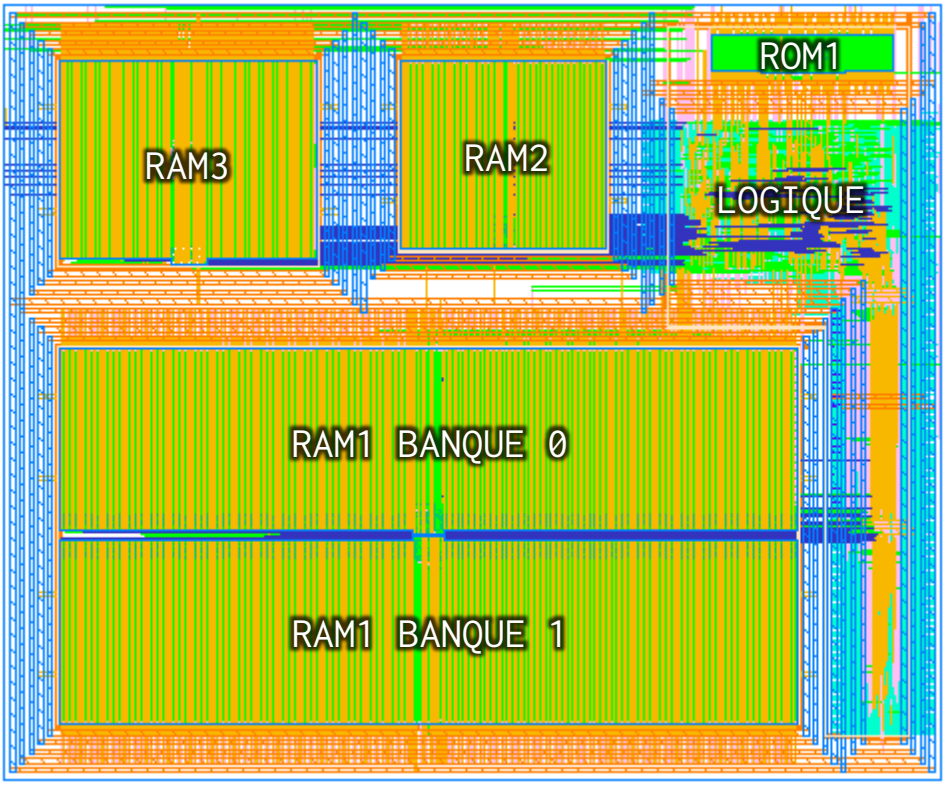
d) Routage

Chaque mémoire est entourée par des rails d'alimentations, situés sur les couches de métal les plus hautes (car les plus larges et donc les moins résistantes, pour limiter au mieux les chutes de tension d'alimentation). Si chaque mémoire possède sa propre alimentation, deux lignes sont quand même partagées avec le reste du circuit : les rails VDD et GND. Ils sont situés sur l'extérieur du circuit et entre chaque bloc (mémoire ou partie numérique).

L'outil de placement et de routage n'étant pas capable de réaliser toutes les étapes nécessaires pour compléter le circuit, en particulier l'ajout de puits enterrés pour isoler les mémoires du reste du circuit, il est nécessaire de procéder à des ajustements après le routage, qui sont réalisés avec un troisième outil. Il faut toutefois prévoir l'espace nécessaire à l'ajout de pistes et de vias lors du routage automatique pour éviter une modification manuelle longue et source potentielle d'erreurs. Pour cela,

avant le routage, on définit des zones de restrictions pour limiter aux mieux l'apparition de pistes autour des mémoires.

Figure 3-16 : Représentation du microcontrôleur avant son intégration dans le circuit final



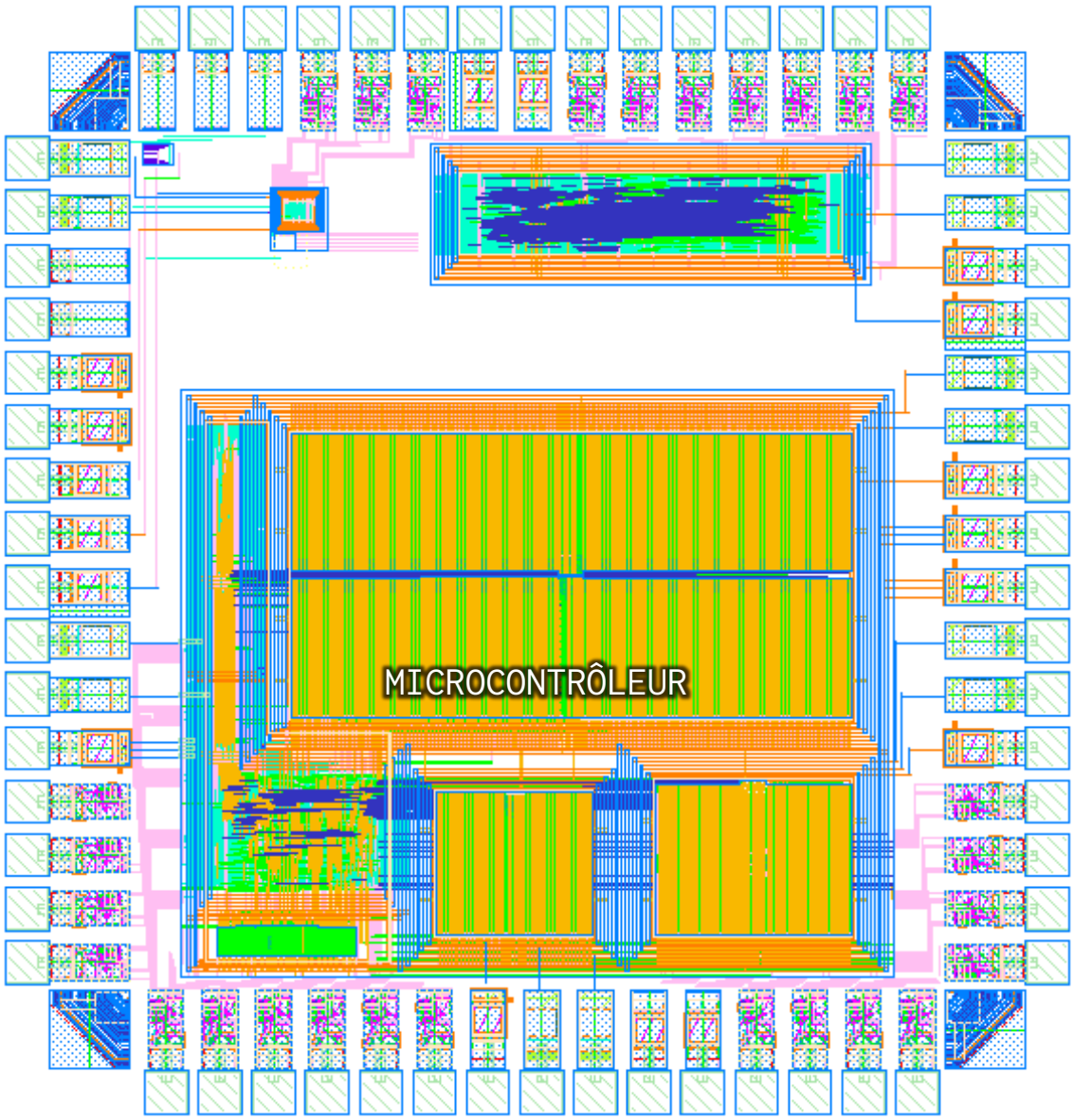
La Figure 3-16 présente notre microcontrôleur après les étapes de placement, de routage et de finition. Il est composé de 5 banques mémoires, 86 diodes de protection contre les effets d'antenne, 63 cellules d'attache (37 *tie low*, 26 *tie high*) et 17249 cellules logiques ou séquentielles. Parmi ces cellules on retrouve 3047 bascules, 163 verrous et 14039 cellules logiques diverses. Parmi toutes ces bascules 899 attribuées par l'outil au processeur Cortex-M0.

e) Intégration

Afin de réduire les coûts de prototypages, le microcontrôleur n'est pas implémenté seul dans un circuit intégré ; d'autres projets nécessitant la réalisation d'un circuit en technologie FD-SOI 28 nm sont également présents sur celui-ci. Afin de limiter la propagation d'un défaut de conception aux autres projets, chaque composant implémenté dispose de sa propre alimentation, non seulement celle du cœur, mais aussi celle des plots. Concernant notre microcontrôleur, chaque mémoire dispose de sa propre alimentation, provenant de broches dédiées ; seule l'alimentation des contrôleurs internes aux mémoires est partagée avec le restant du circuit pour assurer des niveaux logiques identiques, et ainsi éviter de dégrader le circuit. Au total, 34 plots sont utilisés pour le microcontrôleur : 18 plots

numériques (16 entrées/sorties, 2 entrées simples), 5 pour l'alimentation de la partie logique, 3 plots d'alimentation pour chaque mémoire (9 au total), 2 pour l'alimentation de la couronne de plots. La Figure 3-17 présente le circuit envoyé en fabrication, intégrant la couronne de plots et le microcontrôleur (en bas, tourné de 180° par rapport à la Figure 3-16).

Figure 3-17 : Circuit complet en technologie FD-SOI 28 nm



3.4. Conclusion

La simulation, bien que très utilisée aujourd'hui pour réaliser l'évaluation d'une architecture, pose de nombreuses contraintes pour l'exploration de nouvelles solutions pour les microcontrôleurs. Premièrement, plus la simulation est précise, plus elle est lente. Puisque nous réalisons des

modifications à un niveau très bas dans l'architecture (jusqu'au niveau cellule), nous avons besoin de précision dans nos évaluations, il nous faut alors plus de temps pour réaliser une exploration. Deuxièmement, les outils d'évaluation existants capables d'obtenir des estimations de consommation d'énergie à partir d'une simulation au niveau système ne possèdent pas de modèle satisfaisant nos intentions. Il existe peu de modèles d'architectures similaires à celles que l'on retrouve dans les microcontrôleurs, et lorsqu'ils existent, ces modèles sont souvent des boîtes noires que nous pouvons difficilement exploiter et modifier. Enfin, pour réaliser une évaluation au niveau applicatif, il est nécessaire de modéliser l'environnement du système étudié, dans notre cas un nœud de capteur complet incluant capteurs, moteur radio et autres composants externes pouvant influencer le comportement de notre microcontrôleur. Il faut alors développer les modèles de ces différents éléments, une tâche additionnelle qui demande du temps et peut apporter des erreurs supplémentaires.

Nous avons proposé une méthodologie d'évaluation afin de répondre à différents problèmes posés par les flots d'évaluation plus classiques faisant appel uniquement à de la simulation. Plutôt que de nous baser uniquement sur la simulation, nous l'utilisons à bas niveau pour créer des modèles de consommation d'énergie d'un composant ou d'un système, modèles qui sont ensuite combinés avec l'activité du circuit pour obtenir une estimation de la consommation d'énergie de celle-ci. Ces informations sur l'activité du circuit sont quant à elles obtenues à l'aide d'une plateforme de prototypage FPGA, et plus particulièrement grâce à un outil implémenté avec le microcontrôleur dans le FPGA de cette plateforme, le moniteur d'activité. Autour de cette plateforme, nous avons développé une carte de prototypage de nœud de capteur pouvant accueillir capteurs et moteurs radio, nous permettant alors de réaliser des évaluations en tenant compte des éléments externes au microcontrôleur et ainsi garder le contexte applicatif.

Afin de pouvoir comparer les résultats obtenus par la simulation et par notre outil d'évaluation, une implémentation de notre microcontrôleur dans la technologie FD-SOI 28 nm a été envoyée en fabrication auprès de STMicroelectronics. Nous disposerons alors d'une version matérielle du microcontrôleur, qui sera utilisée pour affiner nos modèles de consommation d'énergie.

4. Exploration et résultats

Afin d'améliorer l'efficacité énergétique des dispositifs à faible puissance utilisés pour les applications de l'Internet des Objets, différentes stratégies et technologies émergentes sont proposées pour répondre aux limitations rencontrées avec les solutions traditionnelles. Il est cependant nécessaire de les évaluer et de déterminer si elles peuvent répondre aux contraintes imposées par ces applications, comment optimiser leur intégration, et avec quelles limitations. Il faut étudier l'apport de la non-volatilité des technologies telles que les STT-MRAM pour définir comment en tirer parti, où l'utiliser et quelles stratégies à adopter pour augmenter l'efficacité énergétique des circuits intégrés dans le cadre des applications embarquées. Pour réaliser ces évaluations, nous devons disposer des outils nécessaires à cette tâche ; cependant, les solutions actuelles ne sont pas adaptées pour réaliser ce travail d'exploration de façon optimale, avec une granularité et une prise en compte du contexte applicatif suffisantes. Pour cela, nous avons développé un flot d'exploration basé sur une plateforme de prototypage FPGA, le FlexNode, sur laquelle nous avons conçu une architecture de microcontrôleur de référence. Une première évaluation a pour objectif de valider cette référence, afin de s'assurer que, d'un point de vue comportemental, elle est semblable à un microcontrôleur classique. À partir de cette architecture validée, équipée d'un moniteur d'activité, nous sommes capables de relever durant l'exécution d'un programme applicatif différents évènements liés à la consommation des mémoires. Ces résultats sont ensuite utilisés pour estimer et évaluer la consommation énergétique du microcontrôleur avec différentes configurations d'architectures mémoire, utilisant différentes technologies, et ainsi déterminer une combinaison optimale pour une application.

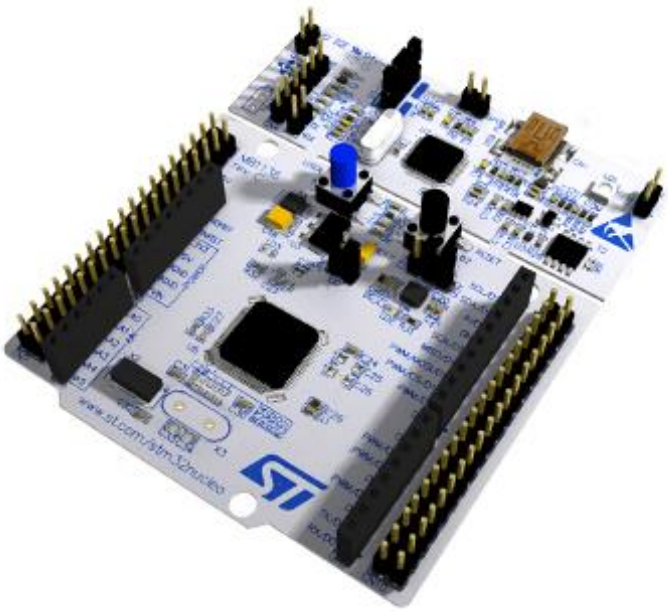
4.1. Validation

Nous avons conçu notre propre architecture de microcontrôleur afin de pouvoir la modifier à souhait, dans le but d'évaluer l'intégration de technologies mémoires émergentes et différentes configurations de celle-ci. Afin de valider le bon fonctionnement du microcontrôleur réalisé, il est important de passer par une étape de validation. Le travail d'exploration étant basé sur notre architecture, cela nous permet aussi de valider une partie de notre méthodologie. Puisque le processeur choisi est utilisé dans des microcontrôleurs vendus dans le commerce, et que l'architecture mémoire est sensiblement la même, il est possible d'effectuer une comparaison en terme de performances avec l'un de ces produits. En terme de puissance consommée, nous ne disposons pas des éléments analogiques qui composent traditionnellement un microcontrôleur (génération et gestion des signaux d'horloge, régulateurs de tension...), mais puisque nous maîtrisons la partie numérique, nous pourrons aussi faire des hypothèses sur ces éléments non connus à partir de mesures effectuées sur des architectures équivalentes et complètes.

4.1.1. Microcontrôleur de référence

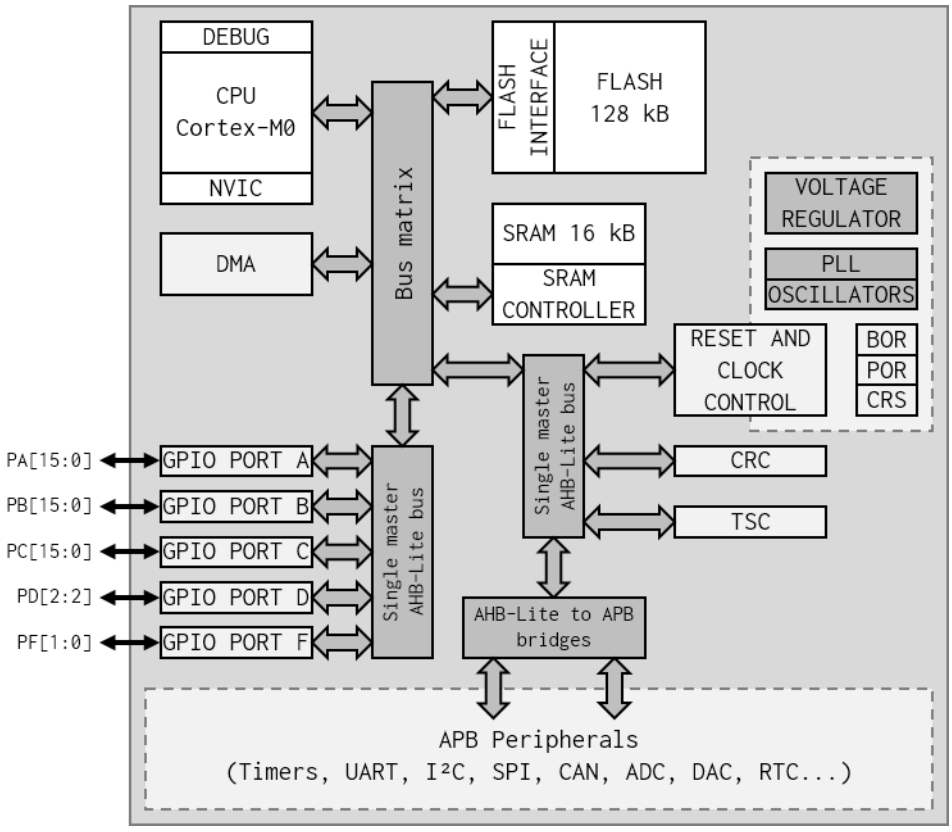
Nous avons conçu notre architecture en partant des caractéristiques de microcontrôleurs existants, nous permettant alors de comparer notre réalisation avec ces derniers. Pour cette vérification, nous avons retenu l'un des microcontrôleurs dont nous nous sommes inspirés pour créer le nôtre, possédant alors des caractéristiques proches. Le microcontrôleur présent sur la carte de démonstration STM32F072 Nucleo-64 (Figure 4-1) est un STM32F072RBT6 de STMicroelectronics. Il possède un Cortex-M0, 128 ko de mémoire non-volatile de type Flash (mémoire principale dont la fonction première est de contenir le programme applicatif) et 16 ko de mémoire volatile de type SRAM. La Figure 4-2 présente une vue simplifiée du schéma fonctionnel du microcontrôleur STM32F072RBT6.

Figure 4-1 : Carte d'évaluation STM32F072 Nucleo-64



Pour que la comparaison entre les deux microcontrôleurs soit valide, il est important de prendre en compte les différences matérielles entre les deux composants et de faire fonctionner ces derniers dans les conditions où ces différences n'ont pas de conséquence sur les comportements des deux architectures, afin de ne pas affecter l'analyse. Si le processeur de chacun est le même (à l'exception du module de débogue, mais ce dernier n'est pas utilisé), ce n'est pas le cas pour le reste du système.

Figure 4-2 : Schéma fonctionnel simplifié de la STM32F072RBT6



4.1.2. Différences entre les architectures

a) Contraintes de la hiérarchie mémoire

Pour commencer, la mémoire principale du microcontrôleur de référence est une mémoire de type Flash embarquée. Or, ces dernières présentent plusieurs limitations techniques.

Les mémoires Flash embarquées (de type NOR) peuvent être lues par accès direct (ou accès aléatoire, comme pour les RAM), mais il n'est pas possible d'effectuer directement des opérations d'écriture. Puisque cette mémoire principale est utilisée uniquement pour contenir le programme applicatif et ses données statiques, cette différence au niveau de l'écriture est ignorée.

Ces mémoires Flash sont limitées en vitesse de lecture. Dans le cas du STM32F072RBT6, la mémoire Flash peut être utilisé jusqu'à 24 MHz. Au-delà de cette fréquence, il est nécessaire d'ajouter un cycle d'attente (*wait state*) pour que la mémoire puisse fournir la donnée demandée dans les temps. Ce cycle d'attente impose donc un cycle supplémentaire pour récupérer chaque donnée de la mémoire, et impacte ainsi les performances du système. S'il existe des accélérateurs matériels (lecture de lignes de 64 bits au lieu de 32 bits et utilisation d'un registre tampon de récupération, *prefetch buffer*), ces derniers ne permettent pas de masquer tous les cycles d'attente (notamment lors des branchements). On choisit alors de prendre une fréquence de fonctionnement inférieure à 24 MHz

pour s'assurer que la mémoire Flash du microcontrôleur de STMicroelectronics soit capable de réaliser des opérations de lecture en un seul cycle, à l'image des mémoires type SRAM.

Ensuite, le bus principal du STM32F072RBT6 est différent de celui utilisé dans notre système. Le microcontrôleur issu du commerce possède une architecture multi-maître, où le processeur n'est pas le seul à pouvoir réaliser des opérations de lecture ou d'écriture dans les mémoires. En effet, on retrouve dans ce microcontrôleur des modules DMA (*Direct Memory Access*) capables de réaliser des transactions entre les mémoires et les périphériques du système, permettant ainsi au processeur d'automatiser le transfert de données en parallèle de son fonctionnement (ou lors sa phase de sommeil) afin de gagner du temps (et de l'énergie). Par exemple, ces modules peuvent être utilisés pour gérer l'envoi et la réception de paquets de données lors de l'utilisation des périphériques de communication, qui ne disposent généralement pas de mémoire tampon de taille suffisante pour tous les cas d'applications. Si plusieurs maîtres tentent d'accéder à la même mémoire ou au même périphérique au même cycle, l'un d'eux se voit pénalisé d'un cycle d'attente (ou de plusieurs cycles selon la cible de la transaction). L'arbitrage des accès est réalisé par le bus interne. Dans le cas de notre microcontrôleur, il n'y a qu'un seul maître, le processeur, donc aucune pénalité d'arbitrage possible. Pour obtenir un fonctionnement identique, on choisit de ne pas utiliser les DMA du microcontrôleur de STMicroelectronics et par conséquent de garder le processeur comme seul maître du bus interne.

b) Contraintes du bus périphérique

Sur le microcontrôleur de STMicroelectronics, la plupart des périphériques (compteurs, UART, I^2C, SPI...) sont câblés sur des bus de périphériques implémentant le protocole AMBA 3 APB. Des ponts sont alors utilisés pour convertir les transactions en provenance du bus principal (protocole AHB-Lite) vers les bus périphériques (protocole APB).

Contrairement au protocole AHB-Lite, chaque transaction réalisée avec un bus implémentant le protocole APB prend au minimum deux cycles (à l'image du protocole AHB, des cycles d'attentes peuvent aussi être requis). Le protocole APB impose que, lors d'une transaction d'écriture, la valeur à écrire doit être stable lors de la transition entre le premier cycle (appelé cycle de mise en place, SETUP) et le second cycle (appelé cycle d'accès, ACCESS).

Extrait de la documentation du protocole AMBA 3 APB [87] :

> « The address, write, select, and write data signals must remain stable during the transition from the SETUP to ACCESS state. »

Cela implique que la donnée à écrire doit être fournie dès le premier cycle de la transaction. Or, dans le protocole AHB, cette donnée n'est présente qu'au cycle suivant la requête de transaction. Afin de respecter cette condition, les ponts faisant l'intermédiaire entre les bus principaux (AHB-

Lite) et les bus pour périphériques (APB) ajoutent un cycle tampon supplémentaire. Il faut donc au minimum trois cycles pour que le processeur du STM32F072RBT6 puisse réaliser une écriture de registre d'un périphérique se trouvant sur un bus APB.

Dans le cas de notre microcontrôleur, tous les périphériques sont câblés sur le bus principal. Aucun bus de périphérique utilisant le protocole APB n'a été implémenté. De ce fait, tous les périphériques du système, disposant d'une interface AHB-Lite, sont accessibles en seulement deux cycles (plus exactement, un cycle de contrôle et un cycle de donnée en différé, mais d'un point de vue logiciel, une instruction d'écriture ou de lecture prend toujours deux cycles au minimum).

Pour calculer le nombre de cycles nécessaires au microcontrôleur de STMicroelectronics pour exécuter le programme de référence, on utilise l'un des périphériques du système, un compteur, paramétré tel que sa valeur s'incrémente à chaque cycle. Puisque ce composant est câblé sur le bus dédié aux périphériques, son activation et son arrêt sont retardés d'un cycle. Toutefois, puisque le décalage est présent au démarrage comme à l'arrêt, le compte est bon. Aucun autre périphérique câblé sur ce bus n'est utilisé lors de l'exécution du programme de référence afin de s'assurer de la validité des résultats.

4.1.3. Résultats

Plusieurs programmes applicatifs de référence ont été utilisés pour vérifier que notre microcontrôleur soit, d'un point de vue comportemental, comparable à l'existant. Le premier est le CoreMark de EEMBC [58], présenté au Chapitre 2 ; le second est une procédure de chiffrement de type AES implémentée en langage C. Pour chacune des deux plateformes, les programmes de référence ont été compilés avec le compilateur GCC (*GNU C Compiler*) pour les processeurs Cortex-M et Cortex-R de ARM [93], dans sa version 6.3.1. Il est configuré pour ne pas optimiser le programme (paramètre *-O0*). Les paramètres de l'évaluation et les résultats obtenus sont résumés dans le Tableau 4-1.

Le nombre d'itérations de chaque programme applicatif correspond au nombre de fois que le programme est exécuté. On constate que les deux microcontrôleurs ont pu exécuter chaque programme applicatif avec exactement le même nombre de cycles, et avec le même temps d'exécution. Si notre architecture de microcontrôleur nous permet de comparer différentes configurations et technologies, ces résultats montrent qu'elle garde un comportement similaire à celui d'un microcontrôleur classique. Nous pouvons alors évaluer des solutions faisant appel à des stratégies et des technologies émergentes et les comparer aux solutions existantes.

Tableau 4-1 : Comparaison des caractéristiques et des résultats obtenus entre le microcontrôleur implémenté sur FPGA et le STM32F072 Nucleo-64

Settings		
Platform	Digilent Cmod A7-35T	STM32F072 Nucleo-64
Hardware	XC7A35T-1CPG236C	STM32F072RBT6
CPU	DesignStart Cortex-M0 r1p0	Cortex-M0 r1
Frequency	12 MHz	12 MHz
Compiler	GCC 6.3.1 20170620	GCC 6.3.1 20170620
Compiler flags	-O0	-O0
Program location	RAM	Flash
CoreMark results		
Iterations	170	170
Runtime (ms)	31 793	31 793
Total cycles	381 526 501	381 526 501
AES results		
Iterations	5 000	5 000
Runtime (ms)	20 803	20 803
Total cycles	249 636 047	249 636 047

4.2. Hybridation des mémoires principales

L'architecture étant validée, nous pouvons réaliser un travail d'exploration pour l'intégration de mémoires magnétiques au sein de celle-ci. Cette section présente l'évaluation d'une banque mémoire STT, celle réalisée par SPINTEC, en remplacement de la mémoire Flash traditionnelle. Pour cela, nous utilisons les informations fournies par le moniteur d'activité, combinées avec les caractéristiques des mémoires dont nous disposons. Différentes configurations d'organisation (une ou deux mémoires) et de technologies (Flash, SRAM, STT-MRAM) sont évaluées.

4.2.1. Modification de la hiérarchie mémoire

La plupart des microcontrôleurs possède au moins deux mémoires : une mémoire non-volatile contenant les instructions et les données statiques du programme applicatif, typiquement une mémoire Flash, et une mémoire volatile pour les données dynamiques de l'application, typiquement

une SRAM. Il existe des microcontrôleurs qui incluent d'autres types de mémoire, tels que des ROM, des EEPROM (*Electrically Erasable Programmable Read-Only Memory*, mémoire morte effaçable électriquement et programmable) ou des banques de registres. Toutefois, cette configuration mémoire typique (mémoire Flash et SRAM) apporte des contraintes et des limitations pour la conception, la fabrication et l'utilisation de ces systèmes.

- Les mémoires Flash sont limitées en vitesse de lecture (typiquement quelques dizaines de mégahertz). Des cycles d'attentes sont nécessaires pour opérer à des cadences plus élevées, entraînent une dégradation des performances (due à une réduction du nombre d'instructions par cycle).
- Les mémoires Flash ne supportent pas l'écriture directe. Il n'est pas possible d'effacer une valeur isolée sans affecter une partie de la mémoire. Pour effacer une valeur, il faut effacer la page où elle se situe, et donc potentiellement effacer d'autres données. Cette opération est coûteuse en temps et en énergie, en plus de la programmation ou reprogrammation de chaque donnée.
- Les SRAM perdent leur contenu si on coupe leur alimentation. Il est nécessaire de garder une partie du circuit alimentée pour préserver les données des mémoires volatiles. Cela implique une consommation résiduelle des mémoires volatiles, même en mode rétention, et de maintenir un régulateur de tension (ou autre fournisseur d'énergie) opérationnel.
- Les SRAM sont peu denses par rapport à d'autres technologies mémoires. Leur capacité est en partie limitée par la taille du microcontrôleur. Pour obtenir des capacités de mémoire plus grandes, il faut se tourner vers des mémoires externes ou des microcontrôleurs prenant plus d'espace, ce qui peut entrer en conflit avec les contraintes d'intégration de l'application.
- La taille de chaque mémoire est fixée lors de la conception par le fabriquant et n'est alors pas modulable.

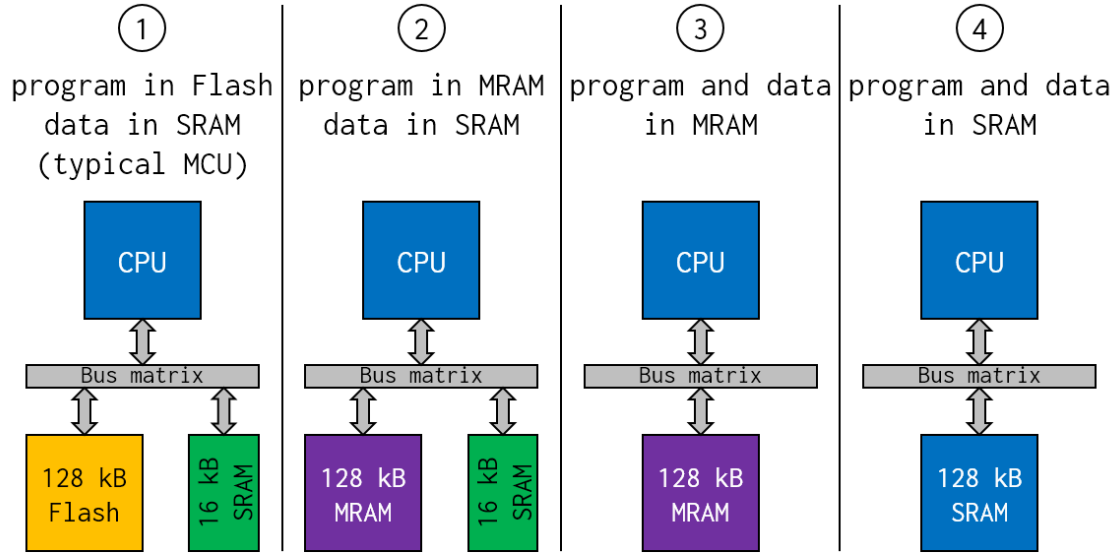
On propose d'étudier l'utilisation de la STT-MRAM dans ce contexte, afin d'évaluer son potentiel de non-volatilité, ses accès directs en lecture et en écriture et sa cadence de fonctionnement suffisamment élevée pour ce type de microcontrôleur.

4.2.2. Étude de cas

Dans cette étude, nous proposons une analyse de différentes configurations de l'architecture mémoire pour un système à un seul maître, qui est le Cortex-M0 et son interface unique. Puisque la STT-MRAM est non-volatile et à accès direct, il est parfaitement envisageable de remplacer la mémoire programme et la mémoire dynamique par cette technologie. On peut aussi n'utiliser qu'une seule STT-MRAM pour le programme et les données dynamiques.

Pour évaluer les gains potentiels de l'intégration de STT-MRAM dans notre système, on compare différentes configurations possibles, illustrées par la Figure 4-3 : (1) instructions et données statiques dans une mémoire Flash et données dynamiques dans une SRAM (configuration typique d'un microcontrôleur du commerce) ; (2) remplacement de la mémoire Flash par une STT-MRAM (la SRAM est toujours utilisée pour les données dynamiques) ; (3) unification des mémoires (instructions et données) en une seule STT-MRAM ; et (4) unification des mémoires (instructions et données) en une seule SRAM.

Figure 4-3 : Représentation de l'architecture du système pour les quatre configurations



On garde le système le plus simple possible afin de concentrer l'évaluation sur les mémoires. Le processeur et les mémoires communiquent à l'aide d'un bus type AMBA3 AHB-Lite simple maître. Les hypothèses suivantes sont faites :

- la fréquence de fonctionnement du système global est choisie de manière à assurer que chaque opération mémoire ne dure pas plus d'un cycle ;
- les SRAM et les STT-MRAM supportent des opérations d'écriture de mots de largeur 8, 16 et 32 bits ;
- toutes les opérations de lecture (y compris les récupérations d'instruction) sont des transactions de largeur 32 bits ;
- aucune interruption, aucune exception et aucun événement qui viendrait modifier le comportement du processeur n'intervient durant l'exécution de l'application de référence ;
- aucune transaction mémoire cachée, puisqu'il s'agit d'une architecture simple maître.

L'énergie consommée par le bus, les contrôleurs mémoire et le processeur n'est pas prise en compte dans cette évaluation. Le programme applicatif de référence utilisé pour cette évaluation est le CoreProfile de EEMBC [59], présenté au Chapitre 2.

Le Tableau 4-2 résume le coût de chaque opération mémoire (lecture et écriture) pour les différentes mémoires implémentées. Les valeurs présentées pour la SRAM sont issues de l'implémentation d'une SRAM de 128 ko avec la technologie FD-SOI 28 nm de STMicroelectronics ; celles de la STT-MRAM sont issues d'une implémentation et simulation faites par SPINTEC [94] ; l'énergie de lecture de la mémoire Flash (nœud technologique 28 nm) a été extrapolée des valeurs présentées dans [95]. Puisque nous manquons d'information sur les courants de fuite (*leakage*), l'évaluation n'est faite que pour la partie dynamique de la consommation d'énergie.

Tableau 4-2 : Coût énergétique par opération pour chaque mémoire

Operation	Flash 128 kB	STT-MRAM 128 kB	SRAM 128 kB	SRAM 16 kB
32-bit read	39.4 pJ	29 pJ	30.6 pJ	11.7 pJ
8-bit write	-	24 pJ	5.85 pJ	2.68 pJ
16-bit write	-	48 pJ	11.7 pJ	5.36 pJ
32-bit write	-	96 pJ	23.4 pJ	10.7 pJ

On observe à partir de ces données que, pour une même capacité (128 ko), la STT-MRAM consomme moins d'énergie que les autres mémoires, Flash et SRAM, pour les opérations de lecture. Toutefois, l'énergie demandée pour les différentes opérations d'écriture est plus importante pour la STT-MRAM (3,0 pJ/bit) que pour la SRAM (0,73 pJ/bit). En fonction de la charge de travail demandée par l'application, les résultats peuvent varier. Quant à la mémoire Flash, cette solution semble la moins intéressante à cause d'une énergie de lecture plus grande que les autres technologies. Ses seuls avantages sur la SRAM sont sa non-volatilité et sa densité.

Concernant la taille des mémoires, la STT-MRAM de 128 ko occupe une surface de 58 000 μm^2, ce qui est environ 3,5 fois plus petit qu'une SRAM de capacité identique (autour de 204 000 μm^2).

4.2.3. Évaluation

Pour chaque configuration, la phase dite active de l'application s'exécute en 49 767 cycles, avec 32 773 instructions exécutées. Le Tableau 4-3 présente le détail des différentes opérations mémoires pour la mémoire principale, contenant les instructions et les données statiques de l'application, et la mémoire secondaire, utilisée pour les données dynamiques. Le nombre de cycles d'inactivité (*idle*

cycles) correspond au nombre de cycles où aucune opération mémoire n'a été initiée. Chaque récupération d'instruction correspond, du point de vue des mémoires, à une lecture d'un mot de 32 bits. Puisque les instructions sont situées dans la mémoire principale, on observe qu'aucune récupération d'instruction n'est faite dans la mémoire secondaire. Aucune opération d'écriture n'est exécutée dans la mémoire principale.

Tableau 4-3 : Détails des opérations mémoires lors de la phase active de l'application de référence

	Program memory	Data memory
Idle cycles	29532	38280
Instruction fetches	18693	0
Total reads	1542	8151
Total writes	0	3336
8-bit writes	0	1000
16-bit writes	0	516
32-bit writes	0	1820

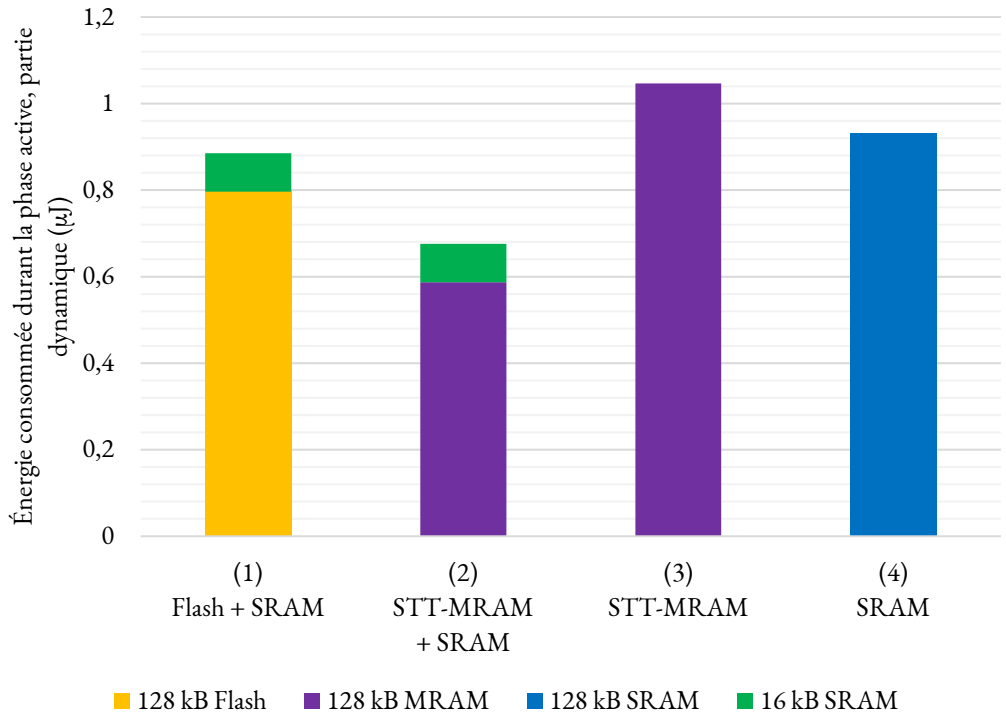
Dans ces résultats, on constate que le nombre d'instructions est inférieur au nombre de cycles d'horloge : cela est dû aux instructions de lecture et d'écriture, nécessitant deux cycles ; et aux branchements nécessitant trois cycles pour récupérer l'instruction suivante de la mémoire et recharger les différents étages du processeur. Le nombre de récupérations d'instructions est quant à lui plus grand que la moitié du nombre d'instruction exécutées. Dans le jeu d'instructions ARMv6-M implémenté par le Cortex-M0, très peu d'instructions ont une taille de 32 bits, la presque totalité faisant 16 bits de largeur ; deux instructions peuvent être récupérées en une seule opération de lecture. Toutefois, des instructions de 32 bits de largeur sont quand même présentes lors de l'exécution du programme, et les instructions récupérées en avance sont perdues lors d'un branchement.

Les résultats contenus dans le Tableau 4-3 montrent également que les récupérations d'instructions (*instruction fetches*) sont les opérations les plus nombreuses effectuées sur les mémoires. En effet, pour le CoreMark, 59% des opérations effectuées sur les mémoires (programme et données confondues) sont des récupérations d'instructions ; 31% sont des lectures de mots de 32 bits de largeur et 10% sont des écritures. En dehors des récupérations qui sont faites dans la mémoire programme, presque toutes les autres opérations de lecture et d'écriture sont effectuées dans la mémoire de données.

À partir des données présentées dans le Tableau 4-2 et le Tableau 4-3, il est possible, pour chaque configuration (Figure 4-3), d'estimer la consommation d'énergie de chaque mémoire pour une

exécution de la phase active de l'application de référence. La Figure 4-4 présente la partie dynamique de l'énergie consommée par les mémoires. Chaque mémoire est représentée par une couleur différente afin de visualiser sa contribution énergétique.

Figure 4-4 : Partie dynamique de l'énergie consommée par les mémoires durant la phase active

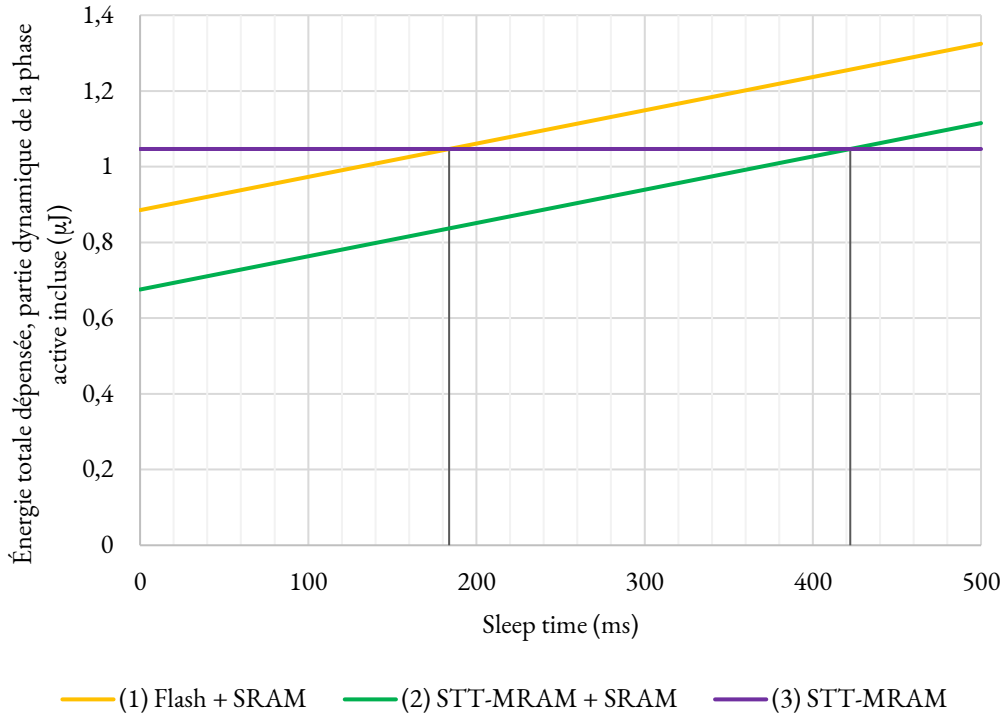


Pour les configurations (1) et (2), où chaque architecture intègre deux mémoires distinctes, on observe pour la mémoire principale que la partie dynamique de la consommation d'énergie de la STT-MRAM est plus faible que celle de la mémoire Flash (environ 26%). Puisque la mémoire secondaire utilisée est la même pour ces deux configurations (une SRAM de 16 ko), leur contribution est identique dans les deux cas. Au total, l'énergie consommée par les mémoires est plus faible dans la seconde configuration que dans la première (d'environ 24%). Pour les configurations (3) et (4), où une seule mémoire est utilisée pour les instructions, les données statiques et les données dynamiques de l'application, on observe que l'architecture ne comportant qu'une seule SRAM de 128 ko nécessite moins d'énergie que l'architecture basée sur une seule STT-MRAM de 128 ko (environ 12%). Cela est dû à une énergie demandée pour les opérations d'écriture plus importante pour la STT-MRAM, même si l'énergie nécessaire pour les opérations lecture est plus faible pour la STT-MRAM que pour la SRAM, pour une capacité identique. Cependant, les opérations mémoires effectuées dans une SRAM de 16 ko (contenant les données dynamiques de l'application) nécessitent bien moins d'énergie que les mémoires de 128 ko, SRAM comme STT-MRAM. C'est pourquoi les

configurations (3) et (4) ont une consommation d'énergie (partie dynamique) plus importante que celle de (2), disposant d'une architecture à deux mémoires.

En considérant maintenant un comportement périodique, comportant une phase active et une phase de sommeil, on peut estimer le temps de sommeil minimum nécessaire pour que la surconsommation d'énergie en phase active de la configuration (3) soit compensée par son courant de fuite faible par rapport aux autres configurations. En effet, la STT-MRAM peut être éteinte lors de la phase de sommeil car elle est non-volatile, ce qui n'est pas le cas des SRAM qui seront au mieux placées en mode rétention. Dans ce mode, la SRAM de 16 ko utilisée pour les configurations (1) et (2) a une consommation en puissance de 879 nW ; et celle de la SRAM de 128 ko utilisée dans la configuration (4) est de 13,3 µW. On considère alors que le courant de fuite de la STT-MRAM est nul lorsque le microcontrôleur est en veille. Il faut donc attendre 184 ms pour que l'énergie « sauvée » dans la configuration (3) lors de la phase de sommeil compense la surconsommation de 162 nJ par rapport à la configuration (1) durant la phase active. Comme indiqué par la Figure 4-5, c'est après 422 ms de sommeil que l'architecture à une seule STT-MRAM de la configuration (3) devient plus efficace d'un point de vue énergétique que l'architecture à deux mémoire, STT-MRAM et SRAM, de la configuration (2).

Figure 4-5 : Énergie perdue par les mémoires durant la phase de sommeil



En termes de flexibilité, les configurations (3) et (4) sont les plus intéressantes car ces solutions offrent la possibilité d'ajuster les partitions mémoires allouées pour les instructions du programme applicatif, ses données statiques et les données dynamiques. De plus, ces architectures à une seule mémoire sont plus simples que celles à deux mémoires proposées dans les configurations (1) et (2).

4.3. Simulation du microcontrôleur FD-SOI 28 nm

La plateforme de prototypage FPGA nous permet actuellement d'obtenir rapidement l'activité des mémoires grâce au moniteur d'activité. Cela est possible car nous disposons des informations nécessaires relatives aux mémoires pour réaliser une telle évaluation. Cependant, ce n'est pas le cas du reste du microcontrôleur, dont nous n'avons aucun modèle. Toutefois, ce microcontrôleur étant synthétisé dans la technologie FD-SOI 28 nm, il est possible de réaliser des simulations au niveau portes afin d'obtenir des estimations de consommation en terme de puissance, et par extension en terme d'énergie.

4.3.1. Extraction de l'activité du circuit

Les simulations sont effectuées avec l'outil ModelSim SE version 2019.1 de Mentor Graphics. Elles sont réalisées à différentes étapes de la conception : simulation RTL ; après la synthèse ; après le routage. Pour la simulation après routage, l'aspect temporel est pris en compte pour s'assurer du bon respect des contraintes spécifiées dans les bibliothèques de la technologie. Lors de cette simulation, les changements de valeur des signaux logiques sont enregistrés dans un fichier prévu à cet effet (au format VCD, *Value Change Dump*). L'extraction des changements de valeurs n'est réalisée que sur la partie intéressée, c'est-à-dire durant l'exécution du programme de référence CoreProfile d'une durée de 1,555 ms environ. Les changements qui ont lieu durant la phase d'amorçage, le transfert des instructions et l'initialisation du système ne sont pas enregistrés. Cette manipulation permet de gagner du temps de simulation, qui est ralentie par la sauvegarde des changements de valeur. Cette simulation, réalisée sur un processeur Intel Xeon (Skylake IBRS) x64, nécessite en effet près de 14 heures et 17 minutes pour simuler environ 114,7 ms de temps d'exécution du microcontrôleur, soit 7 minutes et 28 secondes pour une milliseconde. La simulation est très lente comparée à la plateforme FPGA, cette dernière étant en temps réel. Cela représente un facteur proche de 448000.

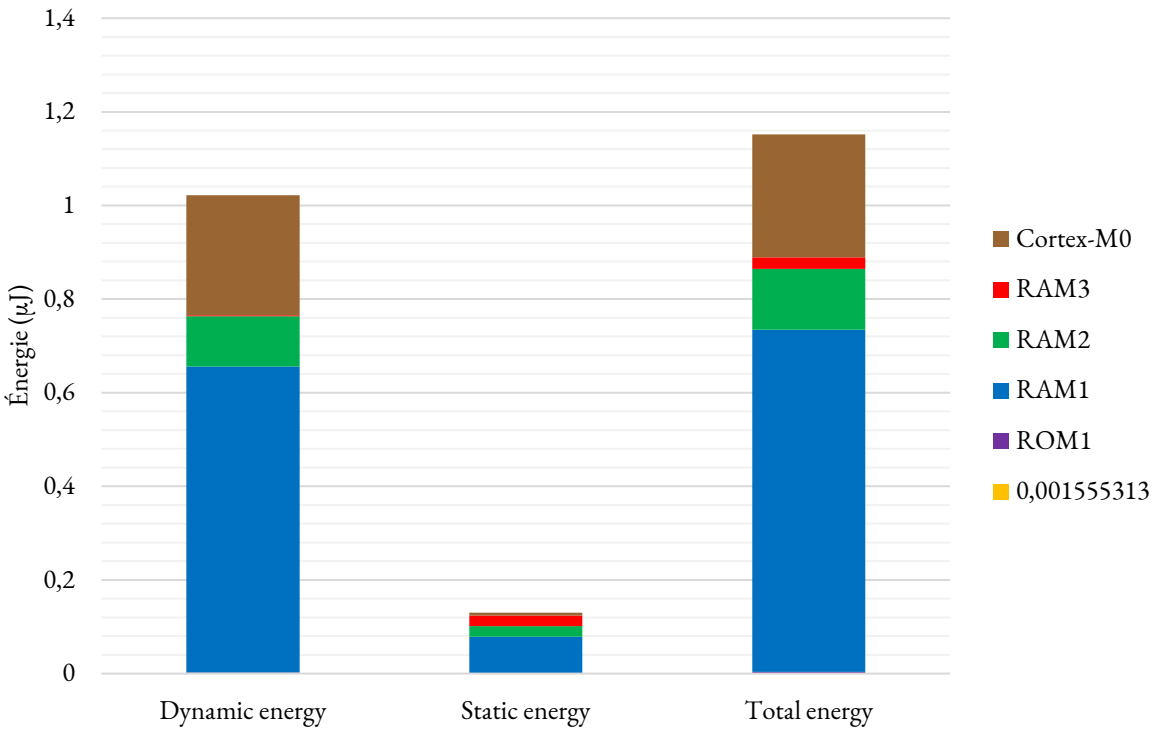
4.3.2. Estimation

Les estimations de consommation en termes de puissance sont obtenues à l'aide de l'outil PrimeTime PX version O-2018.06-SP5. Ce dernier prend comme entrées la liste d'interconnexions générée après l'étape de routage, les changements d'état obtenus après la simulation réalisée avec ModelSim, et les bibliothèques de la technologie utilisée, la FD-SOI 28 nm de STMicroelectronics.

Différentes bibliothèques sont disponibles pour représenter différentes conditions (température, pression, alimentation, polarisation du substrat, variation des paramètres...). Nous utilisons dans un premier temps les paramètres nominaux (alimentation à 0,90 V, pas de polarisation de substrat, température ambiante à 25°C, effets capacitifs et résistifs nominaux). L'outil PrimeTime nous offre en sortie une estimation de la puissance moyenne consommée par chaque cellule. Les plots et leurs contrôleurs ne sont donc pas pris en compte dans cette évaluation.

À partir de la puissance moyenne de chaque élément durant l'exécution du CoreProfile, nous en déduisons l'énergie consommée par notre microcontrôleur, présentée par la Figure 4-6. Cette figure présente également la contribution énergétique des éléments suivant : les mémoires ROM1 (de 2 ko et dédiée aux instructions du programme d'amorçage), RAM1 (de 128 ko et dédiée aux instructions du programme applicatif), RAM2 et RAM3 (toutes les deux de 16 ko et utilisées pour les données dynamiques, la première est une SRAM et la seconde une banque de registres) ; le processeur Cortex-M0 ; le reste de l'architecture est confondu (périphériques, bus de communication interne, arbre d'horloge, contrôleurs mémoire...). L'énergie attribuée à la RAM1 comprend la contribution des deux banques mémoires et de leurs adaptateurs respectifs (la partie adaptateur de la RAM1 représentant à peine plus de 1% de la contribution de celle-ci).

Figure 4-6 : Estimation de la consommation d'énergie obtenue par simulation



Durant la phase active du programme de référence CoreProfile, notre microcontrôleur a consommé 1,68 µJ, dont 1,54 µJ en dynamique et 0,14 µJ en pertes statiques. Plus de 57% de l'énergie totale consommée par le circuit est due aux mémoires. Si la ROM1 a un contribution négligeable, grâce à sa petite taille et au fait qu'elle n'est pas utilisée durant l'exécution de l'application, ce n'est pas le cas des autres mémoires. La RAM3 n'étant pas utilisée par l'application, sa contribution dynamique est très faible (ce qui représente environ 0,63% du total) ; cependant, sa contribution statique l'est moins (16,5%). La RAM1 est l'élément le plus énergivore de ce microcontrôleur : elle représente 42,4% de la consommation dynamique et près de 57% des pertes statiques. Pour ce circuit, il y a un grand intérêt à optimiser l'efficacité énergétique de cette mémoire.

Concernant la partie logique, si le processeur, le Cortex-M0, représente 16,6% de l'énergie dépensée de manière dynamique et que 4,5% des pertes statiques, le reste du circuit (périphériques, bus, horloge...) consomme plus (27% en dynamique, 4% en pertes statiques) bien que la majorité des périphériques ne soit pas utilisée. L'utilisation de signaux d'horloges commandés en remplacement des signaux d'activation d'horloge aiderait à réduire cette contribution, malheureusement nous ne disposons actuellement pas des cellules nécessaires.

4.3.3. Observations

On compare maintenant les résultats obtenus par la simulation aux précédentes estimations réalisées dans la Section 4.2. La RAM1 est composée de deux banques de SRAM de 64 ko et a une consommation totale de 740 nJ. La STT-MRAM de 128 ko conçue par SPINTEC consomme 912 nJ, soit 172 nJ de plus que la solution SRAM pour la même application. Si la mémoire magnétique était intégrée dans notre microcontrôleur comme mémoire principale (RAM1), l'énergie dépensée par le circuit pour le CoreProfile augmenterait donc de 172 nJ, soit 11% de l'énergie totale.

On observe une différence de résultats entre la simulation et l'estimation faite dans la section précédente. En effet, pour la RAM2 nous obtenons une contribution énergétique de 107,5 nJ avec la simulation contre 88,7 nJ avec notre flot d'évaluation. Cette différence de près de 17,5% peut être due à la différence des protocoles et des données de consommation utilisées. D'un côté, la simulation est réalisée en utilisant une suite d'outils standards, à savoir Cadence Genus et Cadence Innovus pour la réalisation du circuit en FD-SOI 28 nm, puis ModelSim et PrimeTime PX pour l'estimation de la puissance moyenne consommée durant l'exécution de l'application de référence, avec les données de la technologie situées dans une bibliothèque fournie par STMicroelectronics. De l'autre, l'estimation de la consommation d'énergie de la mémoire est obtenue en réalisant la somme des énergies consommées par la mémoire pour chaque type de transaction (lecture et écriture de mots de 8, 16 et 32 bits), elle-même obtenue en multipliant le nombre de transactions (effectuées au cours de l'exécution de l'application) par l'énergie consommée par la mémoire pour le type de transaction correspondant. Pour chaque type de transaction, le nombre d'opérations mémoires a été récupéré

depuis le FlexNode, c'est-à-dire grâce au moniteur d'activité présent dans notre microcontrôleur, lui-même implémenté sur un FPGA. Ces premières informations (les nombres de transactions) ont pu être comparées avec les résultats de la simulation, plus particulièrement à partir des changements d'état des signaux enregistrés, qui correspondent parfaitement (les résultats correspondent aussi aux résultats affichés par le moniteur simulé). Les données concernant l'énergie consommée pour chaque opération ont quant à elles été récupérées sur la documentation technique de la mémoire, pour les mêmes conditions (d'alimentation, de température et de variation de procédé de fabrication) que celles de la bibliothèque utilisée avec PrimeTime. L'estimation de la consommation d'énergie obtenue avec notre méthode est moins importante que le résultat obtenu par la simulation. Cette différence peut être due : à des modèles de consommations différents (les valeurs déduites de la documentation ne sont pas correctes, ou les conditions de températures, d'alimentation et de variation de procédé de fabrication ne correspondent pas avec la bibliothèque utilisée avec PrimeTime) ; à un mauvais paramétrage des outils d'évaluation (PrimeTime) ; à une méthode de calcul utilisée dans notre méthodologie (somme des énergies pour chaque opération) pas suffisamment précise ou incomplète. Nous prévoyons de rechercher puis corriger les erreurs qui ont pu s'introduire dans nos modèles et nos évaluations, et d'affiner notre méthodologie si nécessaire.

4.4. Conclusion

Afin de confirmer que l'architecture de microcontrôleur proposée au Chapitre 3 est comparable à celle d'un microcontrôleur classique, nous avons évalué les performances de notre architecture, implémentée sur un FPGA, avec celle d'un microcontrôleur de STMicroelectronics possédant le même processeur, un Cortex-M0 de ARM. Nous avons exécuté sur chaque plateforme des programmes de référence (le CoreMark de EEMBC et un programme de chiffrement AES) et obtenu des temps d'exécution et un nombre de cycles identiques pour chaque programme. Nous avons ainsi démontré que l'architecture est fonctionnelle, et que son comportement est similaire et comparable à celui d'un microcontrôleur classique issu du commerce.

Nous avons ensuite effectué un premier travail d'exploration sur les mémoires principales du système, en comparant différentes configurations de la hiérarchie mémoire et différentes technologies. La première configuration correspond au cas typique que l'on retrouve dans nombre de microcontrôleurs, avec une mémoire programme en technologie Flash (non-volatile, dans notre étude de 128 ko) associée à une mémoire de données de type SRAM de plus petite capacité (volatile, dans notre étude de 16 ko). Nous avons étudié une seconde configuration en remplaçant la mémoire Flash par une STT-MRAM de capacité équivalente (128 ko, jonctions magnétiques de diamètre 40 nm) pour la mémoire principale ; la mémoire de données est identique (une SRAM de 16 ko). Une troisième configuration consiste à utiliser une STT-MRAM de 128 ko comme mémoire unique, servant à l'entreposage des instructions du programme applicatif et de toutes ses données, statiques

et dynamiques. Enfin la dernière configuration emploie une SRAM de 128 ko comme mémoire unique. Toutes les mémoires utilisées ont été réalisées avec la technologie FD-SOI 28 nm. La partie dynamique de la consommation d'énergie est comparée avec les différentes configurations. Pour le programme de référence CoreProfile de EEMBC, durant la partie active, la configuration la plus économe en énergie est la seconde, où la mémoire traditionnelle Flash a été remplacée par une STT-MRAM avec une réduction de l'énergie dynamique consommée de près de 23%. Ce résultat montre qu'il y a un avantage d'un point de vue énergétique à remplacer la technologie Flash par la technologie STT pour la mémoire programme des microcontrôleurs. La troisième configuration, n'utilisant qu'une seule mémoire de type STT-MRAM, mène à une plus grande consommation d'énergie dynamique. Celle-ci s'explique par le fait que les déplacements des données dynamiques sont réalisés dans une petite mémoire dans les deux premières configurations, la SRAM de 16 ko, contrairement aux deux dernières configurations où les transactions sont effectuées dans des mémoires de grande capacité qui nécessitent plus d'énergie pour chaque opération. Cependant, si l'on considère que lors des phases d'inactivité de l'application, l'alimentation des mémoires non-volatiles peut être coupée, cette configuration à une seule mémoire de type STT est celle qui permet d'économiser le plus d'énergie lors de ces phases d'inactivité grâce un courant de fuite nul. Pour l'application CoreProfile, cette configuration à une mémoire est plus économe en énergie que la configuration traditionnelle Flash-SRAM pour une durée de veille supérieure à 148 ms et plus efficace que toutes les autres configurations pour une durée de veille supérieure à 386 ms.

Afin d'effectuer une évaluation à l'échelle du microcontrôleur, nous avons réalisé une simulation de notre microcontrôleur implémenté en technologie FD-SOI 28 nm au niveau cellule (RTL). Après avoir constaté une différence significative entre le temps d'exécution du programme de référence CoreProfile avec le microcontrôleur implémenté sur le FPGA et le temps de simulation, nous avons observé que, pour une configuration mémoire n'utilisant que des SRAM (mémoire programme de 128 ko et mémoires de données de 16 ko), ces dernières représentaient près de 57% de la consommation d'énergie de la partie numérique du circuit. Si les mémoires sont responsables de plus de 90% des pertes statiques du circuit, c'est principalement la mémoire principale (une SRAM de 128 ko) qui est responsable de la majorité de ces pertes. Si cette dernière était remplacée par une mémoire de type STT, la consommation d'énergie totale du circuit augmenterait de 11% (augmentation due à des pertes statiques plus grandes). Toutefois, si l'on considère le remplacement des mémoires volatiles par des technologies non-volatiles dont l'alimentation peut être coupée durant les phases d'inactivité de l'application, il serait alors possible d'économiser 90% des pertes d'énergie durant ces phases d'inactivité grâce à ces technologies.

5. Conclusion et perspectives

Pour beaucoup d'applications de l'Internet des Objets, et plus particulièrement les réseaux de capteurs sans fil, la gestion de l'énergie est un enjeu critique. Différentes stratégies sont utilisées pour optimiser la consommation d'énergie des dispositifs sans fil, en particulier l'organisation des différentes tâches du nœud en phases d'activité et d'inactivité. Grâce à un contrôle de la tension d'alimentation et de la fréquence de travail des différents composants d'un nœud, il est possible d'optimiser la puissance moyenne consommée lors de ces différentes phases, et plus particulièrement lors des phases d'inactivité où les pertes résiduelles d'énergie ont un effet significatif sur la durée de vie du nœud. Si les stratégies les plus agressives permettent de supprimer une grande partie de ces pertes, elles allongent le temps nécessaire aux différents composants du nœud pour un retour à leur état nominal, ce qui n'est pas adapté pour les applications nécessitant des temps de réaction courts. Les technologies traditionnelles posent aujourd'hui de nombreuses limitations quant à la recherche d'un système plus économique énergétiquement. Toutefois, nous avons vu que de nouvelles solutions technologiques sont étudiées pour compléter ou remplacer les solutions actuelles afin d'outrepasser ces limitations et apporter de nouvelles façons de gérer l'énergie dans les systèmes embarqués. Parmi les solutions les plus prometteuses, nous nous sommes intéressés à deux d'entre elles : la technologie FD-SOI et les mémoires magnétiques, plus particulièrement de type STT.

Une analyse de la littérature sur le sujet nous a permis de recenser les différents outils disponibles pour réaliser des évaluations d'architectures numériques, selon plusieurs critères : précision, flexibilité, vitesse d'évaluation, temps de mise en place... Cependant, la plupart de ces outils ciblent des architectures différentes de celles des microcontrôleurs basse puissance, qui ne sont pas toujours adaptées et nécessitent des modifications pour satisfaire nos besoins. Beaucoup de ces solutions font appel à la simulation, étant certes très flexible mais malheureusement trop lente lorsque la précision est recherchée. Enfin, peu d'entre elles prennent en compte l'environnement applicatif du sujet étudié, or cet environnement est très important dans le cadre des applications embarquées de type Internet des Objet où il affecte le comportement des microcontrôleurs. Nous avons donc développé notre propre outil capable de réaliser des évaluations avec rapidité, précision et tenant compte du contexte applicatif. À partir d'un modèle de consommation d'énergie d'un composant, obtenu grâce à des simulations bas niveau, des mesures sur des produits existants ou trouvé dans la littérature, nous avons réalisé des estimations de la consommation d'énergie de ce composant en combinant son modèle à son activité interne. Ces informations sur l'activité du circuit ont été obtenues grâce à un outil intégré à notre microcontrôleur, le moniteur d'activité. Implémenté sur une plateforme FPGA, le microcontrôleur peut opérer en temps réel, ce qui a permis d'accélérer le temps nécessaire à l'évaluation. Nous avons développé une carte de prototypage de nœud de capteur autour de ce FPGA nous permettant de connecter à ce dernier des périphériques externes tels que des capteurs et des

moteurs radio. Cet outil permet de réaliser des évaluations rapides, qui tiennent compte des événements externes au microcontrôleur.

L'accès à une architecture de microcontrôleur est primordial afin de réaliser des travaux d'hybridation de celle-ci. Cependant, n'ayant pas accès à des architectures avancées satisfaisant nos besoins, nous avons décidé de réaliser notre propre microcontrôleur à partir du processeur Cortex-M0 de ARM. Bien que nous ne disposons que d'une version offusquée, le niveau de détail est suffisant pour être implémenté sur FPGA et créer une version ASIC dans la technologie FD-SOI 28 nm de STMicroelectronics. En dehors du processeur, nous avons conçu l'ensemble du microcontrôleur. Nous y avons intégré le moniteur d'activité, périphérique dédié à la récupération d'informations relatives à l'activité de la mémoire, informations nécessaires pour effectuer des évaluations de celle-ci. Après validation, nous avons démontré que notre microcontrôleur est fonctionnel et que son comportement est similaire à un microcontrôleur du commerce.

Grâce à notre outil, nous avons pu réaliser une évaluation des mémoires principales de notre microcontrôleur. Nous avons comparé différentes configurations de la hiérarchie mémoire et avec différentes technologies. Par rapport à la configuration traditionnelle, mettant en œuvre une mémoire programme de type Flash et une SRAM pour les données dynamiques (souvent de capacité plus faible), nous avons observé que le simple remplacement de la mémoire Flash par une STT-MRAM permet de réduire de 23% la consommation dynamique en énergie des mémoires, pour l'application de référence CoreProfile de EEMBC. Nous avons également constaté que l'utilisation d'une STT-MRAM unique pour les instructions et les données de l'application nécessite plus d'énergie lors des phases actives de la même application, mais que lors des phases d'inactivité, la non-volatilité de cette technologie permettait d'économiser plus d'énergie que les autres configurations. Ainsi, pour cette application alternant entre phase d'activité et phase de sommeil, cette solution de mémoire unique est rentable du point de vue énergétique lorsque la durée de la phase de sommeil est supérieure à 386 ms.

Enfin, nous avons réalisé une simulation de notre microcontrôleur implémenté en technologie FD-SOI 28 nm pour réaliser une évaluation à l'échelle du microcontrôleur. Nous avons observé pour une configuration mémoire n'utilisant que des SRAM (mémoire programme de 128 ko et mémoires de données de 16 ko) que ces dernières représentent plus de la moitié de la consommation d'énergie de la partie numérique du microcontrôleur (57%), et sont responsables de plus de 90% des pertes statiques du circuit. La mémoire principale, une SRAM de 128 ko, est responsable de la majorité de ces pertes statiques. Nous avons constaté que si cette dernière était remplacée par une STT-MRAM équivalente, la consommation d'énergie totale du circuit augmenterait de 11%. Si les SRAM sont remplacées par des mémoires non-volatiles, il serait possible de réduire de 90% les pertes d'énergie durant les phases de sommeil de l'application en coupant leur alimentation.

Nous avons pu comparer une partie des résultats obtenus grâce à notre outil d'évaluation avec ceux obtenus par la simulation, et constaté une différence de 22% sur une partie d'entre eux. La

précision de notre méthode peut être améliorée, les résultats obtenus entre les deux méthodologies restent proches ; notre outil est beaucoup plus rapide que les autres outils d'évaluation, et la possibilité d'interconnecter notre plateforme à des composants externes reste un atout majeur pour réaliser des évaluations complètes.

Plusieurs améliorations sont envisagées afin d'améliorer et compléter notre étude. La prochaine étape consistera en la création d'un modèle énergétique du processeur, puis du reste du microcontrôleur afin d'obtenir une plateforme d'évaluation complète. Dans le but d'obtenir un système normalement éteint, nous voulons évaluer une hybridation plus avancée du microcontrôleur, avec l'étude de l'intégration de bascules hybrides magnétiques dans le processeur mais aussi dans le reste du circuit (périphériques). Plusieurs technologies sont à évaluer pour la création d'une architecte non-volatile, les mémoires magnétiques de type STT pour commencer puis de type SOT par la suite. L'unification de la mémoire n'apparaissant pas comme une solution adéquate dans tous les cas, l'utilisation d'une mémoire de données dans la technologie SOT est envisagée, voir plus simplement utiliser la technologie SOT à la place de la STT. En effet, cette solution semble mieux adaptée pour les mémoires de petites tailles, et sa non-volatilité reste un atout indispensable pour la réalisation de microcontrôleurs normalement éteints. La contribution énergétique des périphériques du microcontrôleur étant importante, il nous faut étudier l'utilisation de signaux d'horloges commandés par portes pour la réduire, ainsi que l'utilisation de bascules hybrides au sein de ces derniers.

Pour réaliser une évaluation plus complète de l'intégration des technologies émergentes dans les microcontrôleurs, il est important de prendre en compte le contexte applicatif, c'est-à-dire d'évaluer les apports de ces technologies à l'échelle du nœud capteur complet, en prenant en compte tous les éléments qui le composent (les capteurs, le module radio, l'accumulateur d'énergie...), mais aussi à l'échelle d'un réseau de nœuds capteurs dans le but d'obtenir une vision plus large d'une telle transition technologique.

Des architectures plus complexes sont aussi à l'étude : le nombre de services demandés par les applications augmente, nécessitant alors des architectures plus performantes ou bien dotés d'accélérateurs matériel.

Bibliographie

[1] David Mercer, « Global Connected and IoT Device Forecast Update », *Strategy Analytics Consumer Electronics*, page 6, mai 2019.

[2] Kay Soon Low, W. N. N. Win et Meng Joo Er, « Wireless Sensor Networks for Industrial Environments », dans *Proceedings of the International Conference on Computational Intelligence for Modelling, Control and Automation and International Conference on Intelligent Agents, Web Technologies and Internet Commerce (CIMCA-IAWTIC'06)*, Vienna, Austria, 2005, volume 2, pages 271-276, DOI : 10.1109/CIMCA.2005.1631480.

[3] B. Atwood, B. Warneke et K. S. J. Pister, « Preliminary circuits for Smart Dust », dans *Proceedings of the 2000 Southwest Symposium on Mixed-Signal Design*, 2000, pages 87-92, DOI : 10.1109/SSMSD.2000.836452.

[4] B. Warneke, M. Last, B. Liebowitz et K. S. J. Pister, « Smart Dust: communicating with a cubic-millimeter computer », *Computer*, volume 34, numéro 1, pages 44-51, janvier 2001, DOI : 10.1109/2.895117.

[5] S. Vardhan, M. Wilczynski, G. J. Portie et W. J. Kaiser, « Wireless integrated network sensors (WINS): distributed in situ sensing for mission and flight systems », dans *Proceedings of the 2000 IEEE Aerospace Conference*, Big Sky, MT, USA, 2000, volume 7, pages 459-463 vol.7, DOI : 10.1109/AERO.2000.879313.

[6] « Termes et définitions applicables à l'Internet des objets ». International Telecommunication Union (ITU), 29 juillet 2012.

[7] Partha Pratim Ray, Mithun Mukherjee et Lei Shu, « Internet of Things for Disaster Management: State-of-the-Art and Prospects », *IEEE Access*, volume 5, pages 18818-18835, 2017, DOI : 10.1109/ACCESS.2017.2752174.

[8] Mahammad Shareef Mekala et P. Viswanathan, « A Survey: Smart agriculture IoT with cloud computing », dans *Proceedings of the 2017 International conference on Microelectronic Devices, Circuits and Systems (ICMDCS)*, Vellore, India, 2017, pages 1-7, DOI : 10.1109/ICMDCS.2017.8211551.

[9] Camilo Alejandro Medina, Manuel Ricardo Pérez et Luis Carlos Trujillo, « IoT Paradigm into the Smart City Vision: A Survey », dans *Proceedings of the 2017 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*, Exeter, UK, 2017, pages 695-704, DOI : 10.1109/iThings-GreenCom-CPSCom-SmartData.2017.109.

[10] Bahar Farahani, Farshad Firouzi, Victor Chang, Mustafa Badaroglu, Nicholas Constant et Kunal Mankodiya, « Towards fog-driven IoT eHealth: Promises and challenges of IoT in medicine and healthcare », *Future Generation Computer Systems*, volume 78, pages 659-676, janvier 2018, DOI : 10.1016/j.future.2017.04.036.

[11] Yang Lu, « Industry 4.0: A survey on technologies, applications and open research issues », *Journal of Industrial Information Integration*, volume 6, pages 1-10, juin 2017, DOI : 10.1016/j.jii.2017.04.005.

[12] Milica Pejanović Đurišić, Zhilbert Tafa, Goran Dimić et Veljko Milutinović, « A survey of military applications of wireless sensor networks », dans *Proceedings of the Mediterranean Conference on Embedded Computing (MECO)*, Bar, Montenegro, 2012, pages 196-199.

[13] Rodrigo Román-Castro, Javier López et Stefanos Gritzalis, « Evolution and Trends in IoT Security », *Computer*, volume 51, numéro 7, pages 16-25, juillet 2018, DOI : 10.1109/MC.2018.3011051.

[14] Stephen Brown et Cormac J. Sreenan, « Software Updating in Wireless Sensor Networks: A Survey and Lacunae », *Journal of Sensor and Actuator Networks*, volume 2, numéro 4, pages 717-760, décembre 2013, DOI : 10.3390/jsan2040717.

[15] Adrian M. Caulfield, Eric S. Chung, Andrew Putnam, Hari Angepat, Jeremy Fowers, Michael Haselman, Stephen Heil, Matt Humphrey, Puneet Kaur, Joo-Young Kim, Daniel Lo, Todd Massengill, Kalin Ovtcharov, Michael Papamichael, Lisa Woods, Sitaram Lanka, Derek Chiou et Doug Burger, « A cloud-scale acceleration architecture », dans *Proceedings of the 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, Taipei, Taiwan, 2016, pages 1-13, DOI : 10.1109/MICRO.2016.7783710.

[16] Borja Martinez, Màrius Montón, Ignasi Vilajosana et Joan Daniel Prades, « The Power of Models: Modeling Power Consumption for IoT Devices », *IEEE Sensors Journal*, volume 15, numéro 10, pages 5777-5789, octobre 2015, DOI : 10.1109/JSEN.2015.2445094.

[17] Anders Frøytlog et Linga Reddy Cenkeramaddi, « Design and Implementation of an Ultra-Low Power Wake-up Radio for Wireless IoT Devices », dans *2018 IEEE International Conference on Advanced Networks and Telecommunications Systems (ANTS)*, 2018, pages 1-4, DOI : 10.1109/ANTS.2018.8710086.

[18] Goran Panic, Zoran Stamenković et Rolf Kraemer, « Power gating in wireless sensor networks », dans *Proceedings of the 3rd International Symposium on Wireless Pervasive Computing*, Santorini, Greece, 2008, pages 499-503, DOI : 10.1109/ISWPC.2008.4556258.

[19] Emanuel Popovici, Michele Magno et Stevan Marinkovic, « Power management techniques for Wireless Sensor Networks: A review », dans *Proceedings of the 5th IEEE International Workshop on Advances in Sensors and Interfaces IWASI*, Bari, Italy, 2013, pages 194-198, DOI : 10.1109/IWASI.2013.6576090.

[20] Masanori Hayashikoshi, Yohei Sato, Hiroshi Ueki, Hiroyuki Kawai et Toru Shimizu, « Normally-off MCU architecture for low-power sensor node », dans *Proceedings of the 19th Asia and South Pacific Design Automation Conference (ASP-DAC)*, Singapore, Singapore, 2014, pages 12-16, DOI : 10.1109/ASPDAC.2014.6742852.

[21] Ridha Soua et Pascale Minet, « A survey on energy efficient techniques in wireless sensor networks », dans *Proceedings of the 4th Joint IFIP Wireless and Mobile Networking Conference (WMNC 2011)*, Toulouse, France, 2011, pages 1-9, DOI : 10.1109/WMNC.2011.6097244.

[22] STMicroelectronics, « Optimizing power and performance with STM32L4 Series microcontrollers ». STMicroelectronics, décembre 2019.

[23] NXP Semiconductors, « Power Management for S32K1xx ». NXP Semiconductors, mai 2018.

[24] Atmel, « SAM AT03782: Using Low Power Mode in SAM4N Microcontroller ». Atmel, juillet 2013.

[25] Mohsin Raza, Nauman Aslam, Hoa Le-Minh, Sajjad Hussain, Yue Cao et Noor Muhammad Khan, « A Critical Analysis of Research Potential, Challenges, and Future Directives in Industrial Wireless Sensor Networks », *IEEE Communications Surveys Tutorials*, volume 20, numéro 1, pages 39-95, Firstquarter 2018, DOI : 10.1109/COMST.2017.2759725.

[26] Michael Zwerg, Sudhanshu Khanna et Steven Bartling, « Non-volatile logic SoC with software-hardware co-design and integrated supply supervisor for energy harvesting applications », dans *Proceedings of the IEEE 60th International Midwest Symposium on Circuits and Systems (MWSCAS)*, Boston, MA, USA, 2017, pages 887-889, DOI : 10.1109/MWSCAS.2017.8053066.

[27] Sophiane Senni, Lionel Torres, Pascal Benoit, Abdoulaye Gamatie et Gilles Sassatelli, « Normally-Off Computing and Checkpoint/Rollback for Fast, Low-Power, and Reliable Devices », *IEEE Magnetics Letters*, volume 8, pages 1-5, juin 2017, DOI : 10.1109/LMAG.2017.2712780.

[28] Rajendra Bishnoi, Fabian Oboril et Mehdi Baradaran Tahoori, « Design of Defect and Fault-Tolerant Nonvolatile Spintronic Flip-Flops », *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, volume 25, numéro 4, pages 1421-1432, avril 2017, DOI : 10.1109/TVLSI.2016.2630315.

[29] Sophiane Senni, Lionel Torres, Guillaume Patrigeon, Pascal Benoit, Frederic Ouattara, Jad Modad, Pascal Nouet, Kaan Sevin, François Duhem, Gregory Di Pendina et Guillaume Prenat, « Spintronics: a Way Towards Normally-Off Computing », volume 4, page 11, 2016.

[30] Eric Pop, « Energy dissipation and transport in nanoscale devices », *Nano Research*, volume 3, numéro 3, pages 147-169, mars 2010, DOI : 10.1007/s12274-010-1019-z.

[31] Nor Zaidi Haron et Said Hamdioui, « Why is CMOS scaling coming to an END? », dans *Proceedings of the 3rd International Design and Test Workshop*, Monastir, Tunisia, 2008, pages 98-103, DOI : 10.1109/IDT.2008.4802475.

[32] Philippe Magarshack, Philippe Flatresse et Giorgio Cesana, « UTBB FD-SOI: A process/design symbiosis for breakthrough energy-efficiency », dans *Proceedings of the 2013 Design, Automation Test in Europe Conference Exhibition (DATE)*, Grenoble, France, 2013, pages 952-957, DOI : 10.7873/DATE.2013.200.

[33] Kiyoo Itoh, « Embedded Memories: Progress and a Look into the Future », *IEEE Design Test of Computers*, volume 28, numéro 1, pages 10-13, janvier 2011, DOI : 10.1109/MDT.2011.14.

[34] Hideto Hidaka, « Evolution of embedded flash memory technology for MCU », dans *Proceedings of the 2011 IEEE International Conference on IC Design Technology*, Kaohsiung, Taiwan, 2011, pages 1-4, DOI : 10.1109/ICICDT.2011.5783209.

[35] Paulo Cappelletti, Carla Golla, Piero Olivo et Enrico Zanoni, *Flash Memories*. Springer Science & Business Media, 2013.

[36] Jagan Singh Meena, Simon Min Sze, Umesh Chand et Tseung-Yuen Tseng, « Overview of emerging nonvolatile memory technologies », *Nanoscale Research Letters*, volume 9, numéro 1, page 526, septembre 2014, DOI : 10.1186/1556-276X-9-526.

[37] Luís Cargnini, Lionel Torres, Raphael Brum, Sophiane Senni et Gilles Sassatelli, « Embedded Memory Hierarchy Exploration Based on Magnetic Random Access Memory », *Journal of Low Power Electronics and Applications*, volume 4, numéro 3, pages 214-230, août 2014, DOI : 10.3390/jlpea4030214.

[38] Hidemi Takasu, « The Ferroelectric Memory and its Applications », *Journal of Electroceramics*, volume 4, numéro 2, pages 327-338, juin 2000, DOI : 10.1023/A:1009910525462.

[39] Dudley Allen Buck, « Ferroelectrics for Digital Information Storage and Switching », MIT Digital Computer Laboratory, Technical Report, juin 1952.

[40] M. Trentzsch, S. Flachowsky, R. Richter, J. Paul, B. Reimer, D. Utess, S. Jansen, H. Mulaosmanovic, S. Müller, S. Slesazeck, J. Ocker, M. Noack, J. Müller, P. Polakowski, J. Schreiter, S. Beyer, T. Mikolajick et B. Rice, « A 28nm HKMG super low power embedded NVM technology based on ferroelectric FETs », dans *Proceedings of the 2016 IEEE International Electron Devices Meeting (IEDM)*, San Francisco, CA, USA, 2016, page 11.5.1-11.5.4, DOI : 10.1109/IEDM.2016.7838397.

[41] S. Dünkel, M. Trentzsch, R. Richter, P. Moll, C. Fuchs, O. Gehring, M. Majer, S. Wittek, B. Müller, T. Melde, H. Mulaosmanovic, S. Slesazeck, S. Müller, J. Ocker, M. Noack, D.-A. Löhr, P. Polakowski, J. Müller, T. Mikolajick, J. Höntschel, B. Rice, J. Pellerin et S. Beyer, « A FeFET based super-low-power ultra-fast embedded NVM technology for 22nm FDSOI and beyond », dans *2017 IEEE International Electron Devices Meeting (IEDM)*, 2017, page 19.7.1-19.7.4, DOI : 10.1109/IEDM.2017.8268425.

[42] T. Mikolajick, U. Schroeder et S. Slesazeck, « The Past, the Present, and the Future of Ferroelectric Memories », *IEEE Transactions on Electron Devices*, volume 67, numéro 4, pages 1434-1443, avril 2020, DOI : 10.1109/TED.2020.2976148.

[43] « MSP430 Ultra-Low-Power MCUs | Overview | Microcontrollers (MCU) | TI.com ». (En ligne, consulté le 25/03/2020). Disponible sur http://www.ti.com/microcontrollers/msp430-ultra-low-power-mcus/overview.html#FRAM.

[44] H. S. Philip Wong, Simone Raoux, SangBum Kim, Jiale Liang, John P. Reifenberg, Bipin Rajendran, Mehdi Asheghi et Kenneth E. Goodson, « Phase Change Memory », *Proceedings of the IEEE*, volume 98, numéro 12, pages 2201-2227, décembre 2010, DOI : 10.1109/JPROC.2010.2070050.

[45] Albert Fert et Peter Grünberg, « The Discovery of Giant Magnetoresistance », The Nobel Prize in Physics 2007, 2007.

[46] « GMR Sensor Catalog ». NVE Corporation.

[47] P. M. Tedrow et R. Meservey, « Spin-Dependent Tunneling into Ferromagnetic Nickel », *Physical Review Letters*, volume 26, numéro 4, pages 192-195, janvier 1971, DOI : 10.1103/PhysRevLett.26.192.

[48] M. Julliere, « Tunneling between ferromagnetic films », *Physics Letters A*, volume 54, numéro 3, pages 225-226, septembre 1975, DOI : 10.1016/0375-9601(75)90174-7.

[49] S. A. Wolf, D. D. Awschalom, R. A. Buhrman, J. M. Daughton, S. von Molnár, M. L. Roukes, A. Y. Chtchelkanova et D. M. Treger, « Spintronics: A Spin-Based Electronics Vision for the Future », *Science*, volume 294, numéro 5546, pages 1488-1495, novembre 2001, DOI : 10.1126/science.1065389.

[50] T. W. Andre, J. J. Nahas, C. K. Subramanian, B. J. Garni, H. S. Lin, A. Omair et W. L. Martino, « A 4-Mb 0.18-/spl mu/m 1T1MTJ toggle MRAM with balanced three input sensing scheme and locally mirrored unidirectional write drivers », *IEEE Journal of Solid-State Circuits*, volume 40, numéro 1, pages 301-309, janvier 2005, DOI : 10.1109/JSSC.2004.837962.

[51] I. L. Prejbeanu, M. Kerekes, R. C. Sousa, H. Sibuet, O. Redon, B. Dieny et J. P. Nozières, « Thermally assisted MRAM », *Journal of Physics: Condensed Matter*, volume 19, numéro 16, page 165218, avril 2007, DOI : 10.1088/0953-8984/19/16/165218.

[52] D. C. Ralph et M. D. Stiles, « Spin transfer torques », *Journal of Magnetism and Magnetic Materials*, volume 320, numéro 7, pages 1190-1216, avril 2008, DOI : 10.1016/j.jmmm.2007.12.019.

[53] Yu-Jen Wang, Denny Tang et Hsu-Chen Cheng, « Spin torque transfer MRAM device », US7573736B2, 11 août 2009.

[54] Rajagopalan Ramaswamy, Jong Min Lee, Kaiming Cai et Hyunsoo Yang, « Recent advances in spin-orbit torques: Moving towards device applications », *Applied Physics Reviews*, volume 5, numéro 3, page 031107, septembre 2018, DOI : 10.1063/1.5041793.

[55] Sophiane Senni, Lionel Torres, Gilles Sassatelli, Abdoulaye Gamatie et Bruno Mussard, « Exploring MRAM Technologies for Energy Efficient Systems-On-Chip », *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, volume 6, numéro 3, pages 279-292, septembre 2016, DOI : 10.1109/JETCAS.2016.2547680.

[56] Shimeng Yu et Pai-Yu Chen, « Emerging Memory Technologies: Recent Trends and Prospects », *IEEE Solid-State Circuits Magazine*, volume 8, numéro 2, pages 43-56, Spring 2016, DOI : 10.1109/MSSC.2016.2546199.

[57] « Embedded Microprocessor Benchmark Consortium ». (En ligne, consulté le 03/03/2020). Disponible sur https://www.eembc.org/.

[58] « CPU Benchmark – MCU Benchmark – CoreMark – EEMBC Embedded Microprocessor Benchmark Consortium ». (En ligne, consulté le 03/03/2020). Disponible sur https://www.eembc.org/coremark/.

[59] « CPU Energy Benchmark – MCU Energy Benchmark – ULPMark – EEMBC Embedded Microprocessor Benchmark Consortium ». (En ligne, consulté le 22/10/2018). Disponible sur https://www.eembc.org/ulpmark/.

[60] Zhibo Wang, Yongpan Liu, Albert Lee, Fang Su, Chieh-Pu Lo, Zhe Yuan, Jinyang Li, Chien-Chen Lin, Wei-Hao Chen, Hsiao-Yun Chiu, Wei-En Lin, Ya-Chin King, Chrong-Jung Lin, Pedram Khalili Amiri, Kang-Lung Wang, Meng-Fan Chang et Huazhong Yang, « A 65-nm ReRAM-Enabled Nonvolatile Processor With Time-Space Domain Adaption and Self-Write-Termination Achieving > 4x Faster Clock Frequency and > 6x Higher Restore Speed », *IEEE Journal of Solid-State Circuits*, volume 52, numéro 10, pages 2769-2785, octobre 2017, DOI : 10.1109/JSSC.2017.2724024.

[61] Fang Su, Yongpan Liu, Yiqun Wang et Huazhong Yang, « A Ferroelectric Nonvolatile Processor with 46 μs System-Level Wake-up Time and 14 μs Sleep Time for Energy Harvesting Applications », *IEEE Transactions on Circuits and Systems I: Regular Papers*, volume 64, numéro 3, pages 596-607, mars 2017, DOI : 10.1109/TCSI.2016.2616905.

[62] Shintaro Izumi, Ken Yamashita, Masanao Nakano, Shusuke Yoshimoto, Tomoki Nakagawa, Yozaburo Nakai, Hiroshi Kawaguchi, Hiromitsu Kimura, Kyoji Marumoto, Takaaki Fuchikami, Yoshikazu Fujimori, Hiroshi Nakajima, Toshikazu Shiga et Masahiko Yoshimoto, « Normally Off ECG SoC With Non-Volatile MCU and Noise Tolerant Heartbeat Detector », *IEEE Transactions on Biomedical Circuits and Systems*, volume 9, numéro 5, pages 641-651, octobre 2015, DOI : 10.1109/TBCAS.2015.2452906.

[63] Guenole Lallement, Fady Abouzeid, Martin Cochet, Jean-Marc Daveau, Philippe Roche et Jean-Luc Autran, « A 2.7 pJ/cycle 16 MHz, 0.7 μW Deep Sleep Power ARM Cortex-M0+ Core SoC in 28 nm FD-SOI », *IEEE Journal of Solid-State Circuits*, volume 53, numéro 7, pages 2088-2100, juillet 2018, DOI : 10.1109/JSSC.2018.2821167.

[64] Vipul Kumar Singhal, Vinod Menezes, Srinivasa Chakravarthy et Mahesh Mehendale, « A 10.5μA/MHz at 16MHz single-cycle non-volatile memory access microcontroller with full state retention at 108nA in a 90nm process », dans *Proceedings of the 2015 IEEE International Solid-State Circuits Conference - (ISSCC) Digest of Technical Papers*, San Francisco, CA, USA, 2015, pages 1-3, DOI : 10.1109/ISSCC.2015.7062969.

[65] Johannes Bauer et Felix Freiling, « Towards Cycle-Accurate Emulation of Cortex-M Code to Detect Timing Side Channels », dans *Proceedings of the 11th International Conference on Availability, Reliability and Security (ARES)*, Salzburg, Austria, 2016, pages 49-58, DOI : 10.1109/ARES.2016.94.

[66] Mohamad El Ahmad, Mohamad Najem, Pascal Benoit, Gilles Sassatelli et Lionel Torres, « Adaptive Power monitoring for self-aware embedded systems », dans *Proceedings of the 2015 Nordic Circuits and Systems Conference (NORCAS): NORCHIP International Symposium on System-on-Chip (SoC)*, Oslo, Norway, 2015, pages 1-4, DOI : 10.1109/NORCHIP.2015.7364364.

[67] Brinkley Sprunt, « The basics of performance-monitoring hardware », *IEEE Micro*, volume 22, numéro 4, pages 64-71, juillet 2002, DOI : 10.1109/MM.2002.1028477.

[68] Sebastien Fontaine, Luc Filion et Guy Bois, « Exploring ISS Abstractions for Embedded Software Design », dans *Proceedings of the 11th EUROMICRO Conference on Digital System Design Architectures, Methods and Tools*, Parma, Italy, 2008, pages 651-655, DOI : 10.1109/DSD.2008.59.

[69] Tero Rissa, Adam Donlin et Wayne Luk, « Evaluation of SystemC modelling of reconfigurable embedded systems », dans *Proceedings of the 2015 Design, Automation and Test in Europe Conference*, Munich, Germany, 2005, pages 253-258 Vol. 3, DOI : 10.1109/DATE.2005.143.

[70] Derek Chiou, Huzefa Sunjeliwala, Dam Sunwoo, John Xu et Nikhil Patil, « Fpga-based fast, cycle-accurate, full-system simulators », dans *Proceedings of the second Workshop on Architecture Research using FPGA Platforms*, Austin, TX, USA, 2006.

[71] Derek Chiou, Dam Sunwoo, Joonsoo Kim, Nikhil A. Patil, William Reinhart, Darrel Eric Johnson, Jebediah Keefe et Hari Angepat, « FPGA-Accelerated Simulation Technologies (FAST): Fast, Full-System, Cycle-Accurate Simulators », dans *Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO 2007)*, Chicago, IL, USA, 2007, pages 249-261, DOI : 10.1109/MICRO.2007.36.

[72] Donggyu Kim, Adam Izraelevitz, Christopher Celio, Hokeun Kim, Brian Zimmer, Yunsup Lee, Jonathan Bachrach et Krste Asanovicc, « Strober: Fast and Accurate Sample-Based Energy Simulation for Arbitrary RTL », dans *Proceedings of the ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, Seoul, South Korea, 2016, pages 128-139, DOI : 10.1109/ISCA.2016.21.

[73] « Low power short-range wireless products - Nordic Semiconductor ». (En ligne, consulté le 15/04/2019). Disponible sur https://www.nordicsemi.com/en/Products/Low%20power%20short-range%20wireless.

[74] « The McClean Report 2020 | Report Contents and Summaries | IC Insights ». (En ligne, consulté le 06/02/2020). Disponible sur http://www.icinsights.com/services/mcclean-report/report-contents/.

[75] Somnath Paul, Vinayak Honkote, Ryan Gary Kim, Turbo Majumder, Paolo A. Aseron, Vaughn Grossnickle, Robert Sankman, Debendra Mallik, Tao Wang, Sriram Vangal, James W. Tschanz et Vivek De, « A Sub-cm3 Energy-Harvesting Stacked Wireless Sensor Node Featuring a Near-Threshold Voltage IA-32 Microcontroller in 14-nm Tri-Gate CMOS for Always-ON Always-Sensing Applications », *IEEE Journal of Solid-State Circuits*, volume 52, numéro 4, pages 961-971, avril 2017, DOI : 10.1109/JSSC.2016.2638465.

[76] James Myers et Ben Conrad, « M0N0: An Ultra Low Power Sub-threshold Microcontroller ». DEFENSE TECHNICAL INFORMATION CENTER, 25 mars 2019.

[77] « STM32L0 Series », *STMicroelectronics*. (En ligne, consulté le 06/02/2020). Disponible sur https://www.st.com/en/microcontrollers-microprocessors/stm32l0-series.html.

[78] « STM32F0 Series », *STMicroelectronics*. (En ligne, consulté le 06/02/2020). Disponible sur https://www.st.com/en/microcontrollers-microprocessors/stm32f0-series.html.

[79] « LPC800 Series: Low-Cost Microcontrollers (MCUs) based on Arm® Cortex®-M0+ Cores | NXP ». (En ligne, consulté le 06/02/2020). Disponible sur https://www.nxp.com/products/processors-and-microcontrollers/arm-microcontrollers/general-purpose-mcus/lpc800-cortex-m0-plus-:MC_71785.

[80] « LPC1100 Series: Scalable Entry-level Microcontrollers (MCUs) based on Arm® Cortex®-M0+/M0 Cores | NXP ». (En ligne, consulté le 06/02/2020). Disponible sur https://www.nxp.com/products/processors-and-microcontrollers/arm-microcontrollers/general-purpose-mcus/lpc1100-cortex-m0-plus-m0:MC_1392389687150.

[81] « 32-bit Arm® Cortex®-M0 PSoC® 4 ». (En ligne, consulté le 06/02/2020). Disponible sur https://www.cypress.com/products/32-bit-arm-cortex-m0-psoc-4.

[82] « RX Family of 32-bit High Power Efficiency MCUs », *Renesas Electronics*. (En ligne, consulté le 12/02/2020). Disponible sur https://www.renesas.com/eu/en/products/microcontrollers-microprocessors/rx.html.

[83] « PIC32MM Family | Microchip Technology ». (En ligne, consulté le 12/02/2020). Disponible sur https://www.microchip.com/design-centers/32-bit/pic-32-bit-mcus/pic32mm-family.

[84] « SAM L MCUs | Microchip Technology ». (En ligne, consulté le 12/02/2020). Disponible sur https://www.microchip.com/design-centers/32-bit/sam-32-bit-mcus/sam-l-mcus.

[85] Jonathan K. Brown, David Abdallah, Jim Boley, Nicholas Collins, Kyle Craig, Greg Glennon, Kuo-Ken Huang, Christopher J. Lukas, William Moore, Richard K. Sawyer, Yousef Shakhsheer, Farah B. Yahya, Alice Wang, Nathan E. Roberts, David D. Wentzloff et Benton H. Calhoun, « A 65nm Energy-Harvesting ULP SoC with 256kB Cortex-M0 Enabling an 89.1µW Continuous Machine Health Monitoring Wireless Self-Powered System », dans *2020 IEEE International Solid- State Circuits Conference - (ISSCC)*, 2020, pages 420-422, DOI : 10.1109/ISSCC19947.2020.9063067.

[86] « RX Family Features », *Renesas Electronics*. (En ligne, consulté le 13/02/2020). Disponible sur https://www.renesas.com/us/en/products/microcontrollers-microprocessors/rx/rx-features.html.

[87] « AMBA 3 APB Protocol Specification ». ARM Limited, 2003.

[88] « AMBA® 3 AHB-Lite Protocol v1.0 ». ARM Limited, 06 juin 2006.

[89] « IEEE 1149.1-2013 - IEEE Standard for Test Access Port and Boundary-Scan Architecture ». (En ligne, consulté le 07/08/2019). Disponible sur https://standards.ieee.org/standard/1149_1-2013.html.

[90] Juan P. Oliver, Juan Curto, Diego Bouvier, Manuela Ramos et Eduardo Boemo, « Clock gating and clock enable for FPGA power reduction », dans *Proceedings of the VIII Southern Conference on Programmable Logic*, Bento Goncalves, Spain, 2012, pages 1-5, DOI : 10.1109/SPL.2012.6211782.

[91] « 7 Series FPGAs Clocking Resources User Guide ». XILINX, 2018.

[92] « 28nm FD-SOI Technology Catalog ». STMicroelectronics, juin 2016.

[93] Arm Ltd, « GNU Toolchain | GNU Arm Embedded Toolchain », *ARM Developer*. (En ligne, consulté le 13/08/2019). Disponible sur https://developer.arm.com/tools-and-software/open-source-software/developer-tools/gnu-toolchain/gnu-rm.

[94] Kotb Jabeur et Guillaume Prenat, « Design of a full 1Mb STT-MRAM based on advanced FDSOI technology », *MATEC Web of Conferences*, volume 125, juillet 2017, DOI : 10.1051/matecconf/201712501003.

[95] Igor Kouznetsov, « Embedded 28-nm Charge-Trap NVM Technology », présenté à Flash Memory Summit 2017, Santa Clara, CA, 07 août 2017.

Liste des publications

Guillaume Patrigeon, Paul Leloup, Pascal Benoit et Lionel Torres, « FlexNode: a reconfigurable Internet of Things node for design evaluation », dans *Proceedings of the 2019 IEEE Sensors Applications Symposium (SAS)*, Sophia Antipolis, France, 2019, DOI : 10.1109/SAS.2019.8706095, HAL : lirmm-02079509.

Guillaume Patrigeon, Pascal Benoit, Lionel Torres, Sophiane Senni, Guillaume Prenat et Gregory Di Pendina, « Design and Evaluation of a 28-nm FD-SOI STT-MRAM for Ultra-Low Power Microcontrollers », *IEEE Access*, volume 7, pages 58085-58093, mars 2019, DOI : 10.1109/ACCESS.2019.2906942, HAL : lirmm-02079679.

Guillaume Patrigeon, Pascal Benoit et Lionel Torres, « FPGA-Based Platform for Fast Accurate Evaluation of Ultra Low Power SoC », dans *Proceedings of the 28th International Symposium on Power and Timing Modeling, Optimization and Simulation (PATMOS)*, Platja d'Aro, Spain, 2018, pages 123-128, DOI : 10.1109/PATMOS.2018.8464173, HAL : lirmm-01890567.

Mehdi Baradaran Tahoori, Sarath Mohanachandran Nair, Rajendra Bishnoi, Lionel Torres, Sophiane Senni, **Guillaume Patrigeon**, Pascal Benoit, Gregory Di Pendina et Guillaume Prenat, « A Universal Spintronic Technology based on Multifunctional Standardized Stack », dans *Proceedings of the 2020 Design, Automation Test in Europe Conference Exhibition (DATE)*, Grenoble, France, 2020, pages 394-399, DOI : 10.23919/DATE48585.2020.9116321.

Pascal Benoit, Loic Dalmasso, **Guillaume Patrigeon**, Thierry Gil, Florent Bruguier et Lionel Torres, « Edge-Computing Perspectives with Reconfigurable Hardware », dans *Proceedings of the 14th International Symposium on Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC)*, York, United Kingdom, 2019, pages 51-58, DOI : 10.1109/ReCoSoC48741.2019.9034961, HAL : lirmm-02499157.

Sophiane Senni, Frederic Ouattara, Jad Modad, Kaan Sevin, **Guillaume Patrigeon**, Pascal Benoit, Pascal Nouet, Lionel Torres, François Duhem, Gregory Di Pendina et Guillaume Prenat, « From Spintronic Devices to Hybrid CMOS/Magnetic System On Chip », dans *Proceedings of the 2018 IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC)*, Verona, Italy, 2018, pages 188-191, DOI : 10.1109/VLSI-SoC.2018.8644875, HAL : hal-01982791.

Odilia Coi, **Guillaume Patrigeon**, Sophiane Senni, Lionel Torres et Pascal Benoit, « A novel SRAM — STT-MRAM hybrid cache implementation improving cache performance », dans *Proceedings of the 2017 IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH)*, Newport, RI, USA, 2017, pages 39-44, DOI : 10.1109/NANOARCH.2017.8053704, HAL : lirmm-01548938.

L'Internet des Objets est une infrastructure permettant d'interconnecter des objets pour la réalisation de services évolués. Utilisée pour des applications très variées, les solutions engagées sont très diverses. On retrouve cependant une architecture typique décomposée en trois couches : la couche de perception, la couche de transport et la couche de services. Les dispositifs de la couche de perception, appelés « nœuds capteurs », répondent à des contraintes de taille, de sécurité, de fiabilité, d'autonomie et de longue durée de vie. L'efficacité énergétique des nœuds est cependant la contrainte majeure à laquelle les solutions technologiques actuelles trouvent leurs limites. De nouvelles solutions et stratégies sont proposées pour répondre à ce défi, mais comment les évaluer, avec quels outils et à quelle échelle ? Comment utiliser efficacement les technologies émergentes et optimiser leur intégration dans les microcontrôleurs pour les applications de l'Internet des Objets ? Quelles nouvelles stratégies de gestion de l'énergie nous apportent des technologies telles que la FD-SOI 28 nm et les mémoires non-volatiles, et quelles sont leurs limites ? Sont-elles suffisantes et adaptées ?

Pour étudier l'intégration de technologies émergentes dans les microcontrôleurs, nous avons mis en place une méthodologie d'évaluation, à partir d'une plateforme de prototypage de nœud capteur réalisée autour d'un FPGA. Capable d'opérer dans des réseaux déjà déployés, elle nous permet une évaluation rapide, fine, dans un contexte applicatif. Nous avons étudié le remplacement de l'architecture mémoire traditionnelle par différentes solutions intégrant des mémoires magnétiques non-volatiles de type STT, et constaté en intégrant cette technologie une amélioration significative de l'efficacité énergétique du microcontrôleur pour les applications embarquées.

Mots-clés : *Internet des Objets, systèmes intégrés, MRAM, ultra basse puissance*

The Internet of Things is an infrastructure enabling advanced services by interconnecting things. Although the large variety of Internet of Things applications involve many kinds of technical solutions, many of those are based on a typical architecture that can be divided in three layers: the perception layer, the transport layer and the services layer. The dispositive that composed the perception layer, called "sensor nodes", are subject to technical requirements: size, security, reliability, autonomy, and long lifetime. Sensor nodes' energy efficiency is the most critical point where traditional technologies show their limitations. New strategies and solutions are proposed to overcome this technical challenge; however, how can those be evaluated, with which tools and at which level? How emerging technologies can be optimized and integrated inside microcontrollers for Internet of Things applications? Which are the new strategies for energy management to adopt with technologies such as 28 nm FD-SOI and non-volatiles memories? What are their limitations? Will they be sufficient?

To evaluate the integration of emerging technologies inside low power microcontrollers, we propose a new methodology using an FPGA-based sensor node prototyping platform. Able to operate in already deployed wireless sensor networks, we use it to perform fast and precise evaluations, taking account of the application context. We studied and evaluated multiple memory architecture configurations based on STT magnetic memories as a replacement of traditional solutions, and showed that the non-volatile STT memory technology can improve a microcontroller's energy efficiency for embedded applications.

Keywords: *Internet of Things, integrated systems, MRAM, Ultra-Low Power*