Conception d'un processeur ultra basse consommation pour les nœuds de capteurs sans fil

Florent Berthier

ANNÉE 2016

UNIVERSITÉ DE RENNES 1

UNIVERSITE BRETAGNE LOIRE

THÈSE / UNIVERSITÉ DE RENNES 1

sous le sceau de l'Université Bretagne Loire

pour le grade de

DOCTEUR DE L'UNIVERSITÉ DE RENNES 1

Mention : Traitement du Signal et Télécommunications

École doctorale MATISSE

présentée par

Florent BERTHIER

préparée à l'unité de recherche IRISA (UMR 6074) - CEA LETI
Institut de Recherche en Informatique et Systèmes Aléatoires - Commissariat à l'Énergie Atomique et aux Énergies Alternatives
École Nationale Supérieure de Sciences Appliquées et de Technologie

---

Conception d'un processeur ultra basse consommation pour les nœuds de capteurs sans fil

Thèse soutenue à Grenoble le 8 décembre 2016

devant le jury composé de :

Lionel Lacassagne
Professeur HDR, Université Pierre et Marie Curie, LIP6
Rapporteur

Pascal Benoit
Maitre de Conférences HDR, Université de Montpellier, LIRMM
Rapporteur

Ian O'Connor
Professeur, Ecole Centrale de Lyon, INL
Examinateur

Jean-Luc Nagel
Chercheur, CSEM,
Examinateur

Olivier Sentieys
Directeur de recherche, Inria
Directeur de thèse

Edith Beigné
HDR, CEA LETI
Co-directrice de thèse

ANNÉE 2016

UNIVERSITÉ DE RENNES 1

UNIVERSITE BRETAGNE LOIRE

THÈSE / UNIVERSITÉ DE RENNES 1

sous le sceau de l'Université Bretagne Loire

pour le grade de

DOCTEUR DE L'UNIVERSITÉ DE RENNES 1

Mention : Traitement du Signal et Télécommunications

École doctorale MATISSE

présentée par

Florent BERTHIER

préparée à l'unité de recherche IRISA (UMR 6074) - CEA LETI
Institut de Recherche en Informatique et Systèmes Aléatoires - Commissariat à l'Énergie Atomique et aux Énergies Alternatives
École Nationale Supérieure de Sciences Appliquées et de Technologie

---

Conception d'un processeur ultra basse consommation pour les nœuds de capteurs sans fil

Thèse soutenue à Grenoble le 8 décembre 2016

devant le jury composé de :

Lionel Lacassagne
Professeur HDR, Université Pierre et Marie Curie, LIP6
Rapporteur

Pascal Benoit
Maitre de Conférences HDR, Université de Montpellier, LIRMM
Rapporteur

Ian O'Connor
Professeur, Ecole Centrale de Lyon, INL
Examinateur

Jean-Luc Nagel
Chercheur, CSEM,
Examinateur

Olivier Sentieys
Directeur de recherche, Inria
Directeur de thèse

Edith Beigné
HDR, CEA LETI
Co-directrice de thèse

Table des matières

Table des figures

Liste des tableaux

Introduction générale

Cette introduction présente le contexte du travail de cette thèse, les objectifs de la thèse et l'organisation du mémoire.

Contexte

Internet des Objets

Cette discipline consiste à vouloir connecter n'importe quel objet à Internet afin de créer de nouveaux services. Ces systèmes embarqués sont aujourd'hui très répandus grâce à la miniaturisation des composants et de leurs coûts en permanente diminution. La complexité des circuits augmente de manière exponentielle grâce à la réduction de la finesse de gravure des transistors. Dans les systèmes créés pour l'Internet des Objets ce n'est pas uniquement la puissance de calcul qui est augmentée mais aussi la co-intégration de dispositifs mécaniques, radio-fréquences, optiques et même biologiques. Ces systèmes embarqués possèdent alors des systèmes sur puce avec toujours plus de fonctions hétérogènes. Les nœuds de capteurs sans fil peuvent alors être mis en réseaux (WSN [1]) et être distribués dans l'espace afin de récupérer et traiter des mesures physiques, chimiques ou biologiques et ainsi les transférer à travers une liaison radio vers le réseau Internet. Il est alors possible d'avoir une surveillance permanente du milieu dans lequel ces nœuds de capteurs sont placés et suivant l'application, d'agir sur ce milieu grâce à des actionneurs. La consommation ainsi que la flexibilité de ces systèmes est un des enjeux principaux de recherche de part leurs utilisations dans de multiple applications qui peuvent avoir des demandes bien différentes. Cette demande en faible consommation et en flexibilité se retrouve entre autre au niveau du microcontrôleur du nœud de capteur.

Nœuds de capteurs communicants

Depuis la fin du XX^e siècle, les communautés scientifiques et industrielles travaillent sur ce thème. A cause de la répartition spatiale de ces nœuds de capteurs dans des environnements très diverses, ceux-ci sont obligés d'être autonome en énergie grâce à des sources de stockage d'énergie et/ou de récupération de l'énergie dans leur milieu ambiant. Les nœuds de capteurs sont constitués de radios, de capteurs, d'une unité de calcul appelée microcontrôleur, d'un gestionnaire de puissance pour distribuer l'énergie dans les différentes entités du nœud et d'un moyen de stockage de l'énergie et/ou de récupération d'énergie. De gros efforts sont effectués par la communauté scientifique pour réduire la consommation des différentes entités du nœud afin d'une part d'augmenter considérablement leur autonomie et remplacer ces systèmes de leur environnement le moins de fois possible, et d'autre part de réduire toujours plus la taille du nœud pour les intégrer au mieux dans leur milieu. Différentes techniques sont employées pour réduire la consommation du nœud. La première est de faire fonctionner le nœud en rapport cyclique, c'est à dire qu'ils vont fonctionner sur une courte période d'activité et vont être mis dans le mode de consommation le plus bas le reste du temps. L'élément le plus consommateurs d'énergie

1. Wireless Sensor Network

dans le nœud reste la radio c'est pourquoi celle-ci est réveillée uniquement quand cela est nécessaire. Le microcontrôleur représente aussi une part non négligeable de la consommation moyenne du nœud c'est pourquoi cette thèse se concentre sur la réduction de la consommation du microcontrôleur du nœud de capteurs.

Microcontrôleurs ultra basse consommation

Le microcontrôleur est le centre de calcul et de contrôle du nœud de capteurs. Il est en charge des tâches de transfert des données provenant des capteurs vers la mémoire du microcontrôleur en passant par des périphériques de communications ou par des convertisseurs analogiques numériques. Ensuite, celui-ci doit effectuer des calculs sur ces données brutes pour les transformer en données utiles et agir sur des actionneurs suivant ces valeurs et suivant l'application. Il peut aussi envoyer les données utiles à la radio en passant par des périphériques de communication et en gérant le protocole de communication. Ceci afin de centraliser et traiter globalement l'ensemble des données provenant de tous les nœuds de capteurs sur les serveurs applicatifs. Le microcontrôleur est aussi en charge, la plupart du temps, du réveil du nœud entier grâce à l'utilisation de *timer* et d'horloges très basse consommation. Les tâches les plus courantes pour le microcontrôleur sont donc : les transferts de données de ses périphériques à sa mémoire dans le cadre des mesures sur les capteurs ou de la réception de données par la radio, les transferts de données de sa mémoire à sa mémoire pour les calculs des données brutes aux données utiles et les transferts de sa mémoire à ses périphériques dans le cadre de transmissions radio. Les tâches de réveil sont aussi des tâches courantes pour un microcontrôleur de nœud de capteurs communicant, que ce soit pour allumer ou éteindre ses modules internes ou pour mettre les radios et les capteurs dans des modes de consommation différents.

Challenges et objectifs de la thèse

Les challenges au niveau du nœud de capteurs sont multiples et s'effectuent à différents niveaux que ce soit technologiques, matériels, et logiciels. L'un des challenges est d'utiliser les technologies avancées pour réduire la surface des circuits et exploiter leur nouvelles possibilités pour réduire leurs consommations dynamiques et statiques, le tout devant être suffisamment flexible pour s'adapter à n'importe quelle application. Au niveau matériel, il faut déterminer des architectures là encore suffisamment flexibles pour être très peu consommatrices d'énergie mais aussi pour apporter toujours plus de services au sein du nœud. Le challenge s'effectue aussi au niveau de la gestion de la puissance qui doit être à une efficacité maximale à la fois pour générer de très faibles tensions et de très faibles courants pendant les phases de veille et de forts courants pendant les phases actives pour alimenter le microcontrôleur, les capteurs et les radios. Au niveau logiciel, les travaux se portent tout particulièrement sur la mise en place de protocoles radios très peu consommateur d'énergie ainsi qu'une sécurisation des données au sein du nœud. Le portage d'applications ainsi que la gestion des nœuds au cours du temps et leur maintenance est aussi un énorme challenge et est en partit possible grâce au développement de système d'exploitation pour nœud de capteurs (RTOS [2]) utilisant que très peu de ressources du microcontrôleur. On peut citer parmi eux FreeRTOS ou encore ARM RTX.

Cette thèse a pour objectif de réduire la consommation moyenne du microcontrôleur d'un nœud de capteurs sans fil par des procédés architecturaux et de conceptions en utilisant une technologie avancée. Elle va tenter de répondre aux questions suivantes :

2. Real Time Operating System

- Est-ce qu'une architecture de microcontrôleur partitionnée entre un processeur de réveil et un processeur généraliste permet une réduction de sa consommation moyenne ?
- Est-ce que le processeur de réveil conçu en logique asynchrone sur des technologies avancées est efficace énergétiquement ?

Organisation du mémoire

Ce manuscrit est organisé en deux parties, sept chapitres et des annexes.

État de l'art et modélisation

Cette première partie sert a positionner le sujet par rapport aux réalisations de la communauté scientifique et a répondre à la première question.

- Le chapitre 1 présente le contexte économique autour des nœuds de capteurs communicants, l'architecture des réseaux de communications ainsi que les applications dans lesquels les nœuds de capteurs sont et vont être utilisés. Il présente aussi un état de l'art sur les nœuds de capteurs complet actuels,
- Le chapitre 2 traite de la partie microcontrôleur du nœud de capteurs. Il présente son architecture et les différentes réalisations dans les communautés scientifiques et industrielles,
- Le chapitre 3 présente toutes les techniques de réduction de la consommation dans un circuit au niveau architectural, de la gestion de l'énergie, technologique et au niveau du *design*,
- Le chapitre 4 est une analyse de la consommation d'un nœud de capteurs dans les différentes phases applicatives et plus particulièrement du microcontrôleur. En effet ce chapitre présente les gains potentiels sur la consommation moyenne d'un microcontrôleur qui peuvent être obtenus en partitionnant l'architecture entre une partie contenant le processeur de réveil et une partie contenant le processeur principal.

Conception d'un processeur de réveil basse consommation

Cette partie traite de l'implémentation de la plateforme de réveil et tout particulièrement du processeur de réveil et va tenter de répondre à la seconde question lors de la présentation des résultats.

- Le chapitre 5 présente les spécifications et l'architecture de la plateforme de réveil. Il présente tout particulièrement le jeu d'instructions du processeur, l'architecture de la plateforme de réveil et du processeur ainsi que son principe de fonctionnement et la manière dont le debugage a été mis en place dans le processeur,
- Le chapitre 6 traite de la méthode de conception asynchrone ainsi que de la microarchitecture de la plateforme de réveil et du processeur de réveil. Chaque module interne est présenté dans ce chapitre,
- Le chapitre 7 présente quant à lui les résultats en performances et consommations du circuit pour des simulations post-backend du circuit. Les résultats silicium du circuit seront présentés à la soutenance.

Publications liées au manuscrit

Article de journal

Florent Berthier, Edith Beigne, Frédéric Heitzmann, Olivier Debicki, Jean-Frédéric Christmann, Alexandre Valentian, Olivier Billoint, Esteve Amat, Dominique Morche, Soundous Chairat, and Olivier Sentieys. UTBB FDSOI suitability for iot applications : Investigations at device, design and architectural levels. *Solid-State Electronics*, pages –, 2016

Articles de conférences

F. Berthier, E. Beigne, P. Vivet, and O. Sentieys. Power gain estimation of an event-driven wake-up controller dedicated to wsn's microcontroller. In *IEEE 13th International New Circuits and Systems Conference (NEWCAS)*, pages 1–4, June 2015

F. Berthier, E. Beigne, P. Vivet, and O. Sentieys. Fresh ideas: Asynchronous wake up controller for wsn's microcontroller: Power simulation and specifications. In *IEEE 21th International Asynchronous Circuits and Systems (ASYNC)*, May 2015

Poster

Session poster à l'école d'hiver francophone FETCH 2016

Brevet

F. Berthier, E. Beigne, F. Heitzmann, O. Debicki, and O. Sentieys. Coeur de processeur asynchrone et microcontrôleur de noeud de capteur communicant comportant un tel coeur de processeur. Brevet, 2016

Première partie

État de l'art et modélisation

Chapitre 1

Marché actuel de l'IoT, architecture des réseaux de communication, applications et nœuds de capteurs communicants

C'est en 1998 que fut créé le premier microsystème de taille millimétrique capable de mesurer des informations grâce à des capteurs et de les communiquer vers l'extérieur. Ce projet, nommé Smart Dust [38] [169] [170], a été réalisé par l'équipe des professeurs Pister et Kahn de l'Université de Berkeley en Californie. Les nœuds de capteurs communicants et les réseaux de capteurs sans fils sont alors nés, ce qui donnera naissance ensuite à l'Internet des objets (IoT [1]) et même maintenant à l'Internet de tout (IoE [2]). Nous allons analyser dans un premier temps le marché actuel de l'Internet des objets, ainsi que les estimations dans les années à venir pour comprendre l'importance de ces systèmes dans les installations futures. Ensuite, une explication de l'architecture du réseau de l'Internet des objets sera faite, suivie des réseaux et moyens de communications existants pour l'Internet des objets. Enfin une analyse des différentes applications visées par l'Internet des objets sera proposée pour bien comprendre à quel point cela impacte déjà la société.

1.1 Marché actuel et prédictions autour de l'Internet des objets

Le marché de l'Internet des objets est conséquent. Il représente 698,6 milliards de dollars en 2015 et pourrait atteindre les 1700 milliards de dollars en 2019 [3] avec un taux de croissance annuel composé (CAGR) de 35%, sur la période de 2014 à 2019. Comme montré sur le graphique de la figure 1.1, le nombre d'objets connectés était de 2 milliards en 2013 et est estimé à 24 milliards en 2019 soit un nombre d'objets connectés 12 fois plus important que celui d'aujourd'hui. Cisco estime même qu'il y aura 50 milliards d'unités connectés à Internet en 2020 comparé aux 15 milliards d'aujourd'hui. C'est donc un marché en plein essor qui fait beaucoup parler de lui étant donné l'argent qu'il va générer et permettre d'économiser aux entreprises.

Les entreprises adopteront massivement les solutions IoT dont le but est de mettre en réseau un ensemble de nœuds afin de baisser leurs coûts d'exploitation, d'augmenter leur productivité, de s'étendre à de nouveaux marchés ou encore de développer de nouveaux produits. Une courte introduction sur la représentation des réseaux de capteurs

1. Internet of Things
2. Internet of Everything

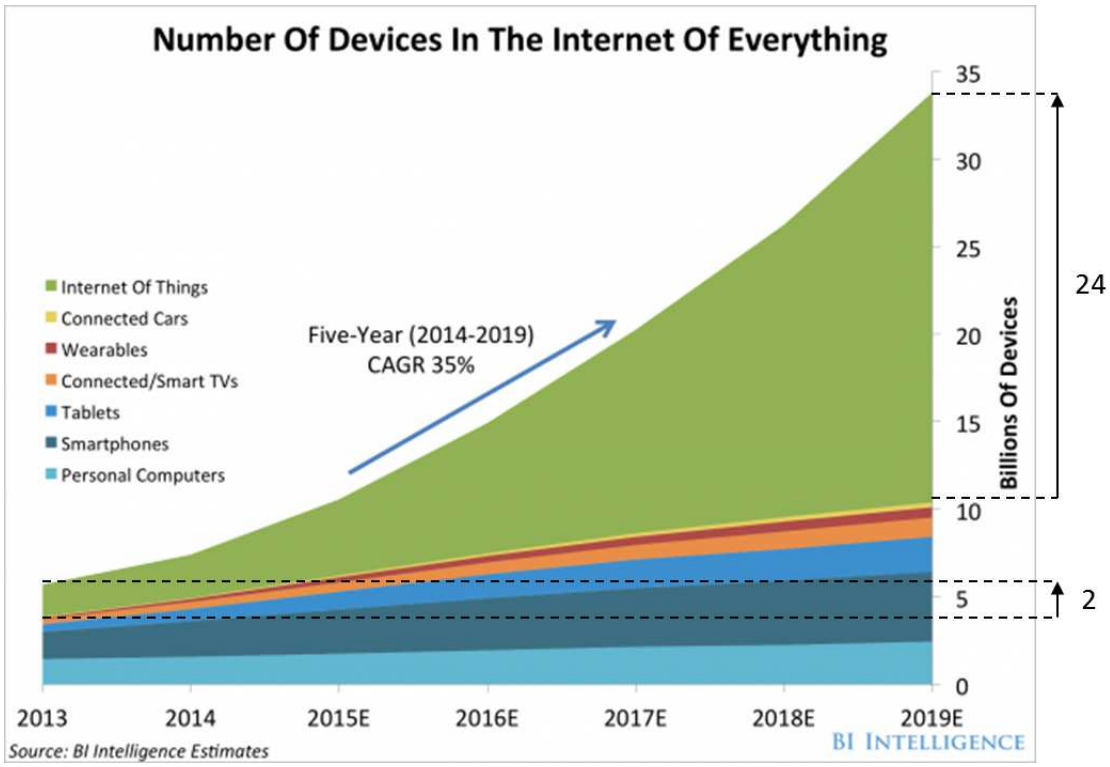


FIGURE 1.1 – Nombre d'objets connectés dans le monde actuellement et estimation jusqu'à 2019 [2]

communicants dans les réseaux de communication actuels ainsi que les différentes couches constituant les réseaux de l'IoT va maintenant être présenté.

1.2 Représentation globale des réseaux de communication

Les réseaux de communication sont séparés en plusieurs couches. L'architecture du réseau de l'IoT peut être séparée en 5 couches comme présenté figure 1.2. La première couche est la couche de perception constituée de tous les nœuds de capteurs effectuant des mesures dans leur environnement et transférant de l'information à la couche suivante. Il peut y avoir aussi des actionneurs pouvant agir sur cet environnement. C'est à l'intérieur de la couche de perception que les travaux de cette thèse se focalisent puisqu'ils se concentrent sur la partie microcontrôleur du nœud de capteurs. La couche d'accès et de sécurité est la couche qui permet de faire la passerelle entre les réseaux internet ou GSM avec les autres types de sous réseaux. C'est la porte d'accès vers les réseaux de capteurs. Il faut donc qu'elle soit suffisamment sécurisée pour filtrer les requêtes afin d'éviter d'être piratée. Ensuite la couche réseau et transport est la couche qui permet de transférer les données vers la couche de stockage. Cette couche de stockage est ce que l'on peut appeler le *Cloud* ou l'ensemble des serveurs sur lesquels vont être traitées toutes les données. La dernière couche est la couche applicative constituée de tous les algorithmes de traitement de données. C'est la couche de service, qui fait le lien direct avec l'utilisateur final. C'est dans cette couche que la valeur ajoutée est la plus forte car elle décide de la manière de stocker et traiter ces données.

Ces couches sont un peu plus détaillées dans la figure 1.3. Dans la partie *collecte des données* les différents nœuds de capteurs peuvent se connecter non seulement entre eux

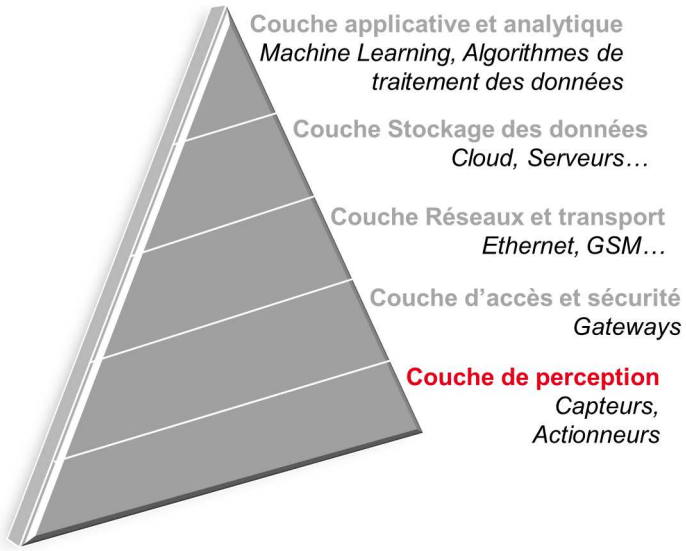


FIGURE 1.2 – 5 couches de l'architecture de l'internet des objets

mais aussi avec une passerelle de la partie *accès au réseau* grâce à des radios supportant les protocoles Wi-Fi, basé sur la norme IEEE 802.11 [76], ZigBee, basé sur la norme IEEE 802.15.4 [78], Bluetooth, basé sur la norme IEEE 802.15.1 [77], ou encore, plus récemment, LoRa, basé sur les normes définies par la LoRa Alliance [15]. L'annexe A.1 résume les radios existantes pour nœuds de capteurs et leurs caractéristiques principales. Le réseau satellite peut aussi être une passerelle vers le réseau Internet. Le *cœur du réseau* Internet lui est constitué de ligne de transmission telle que la fibre optique, des répétiteurs ou routeurs pour interconnecter le monde entier. Ensuite vient la partie serveur qui va gérer le stockage des données, la gestion de recherches d'informations, la gestion des applications, la résolution des adresses et des identifiants, etc. L'application effectue des requêtes aux serveurs pour récupérer les données traitées et afficher les informations utiles à l'utilisateur et/ou agir sur des actionneurs automatiquement.

1.3 Quelles applications pour les nœuds de capteurs ?

Comme présenté dans la section 1.1, le secteur de l'Internet des objets est en pleine expansion. Aujourd'hui, beaucoup d'applications pour nœuds de capteurs sont en cours de déploiement ou sont en développement dans les laboratoires ou les industries. Dans cette partie nous allons analyser une partie de ces applications. La liste n'est pas exhaustive mais donne une bonne idée sur ce que permettent de faire les réseaux de capteurs communicants aujourd'hui et ce qu'ils nous permettront de faire plus tard. Il existe huit secteurs de développement pour les nœuds de capteurs communicants qui sont la domotique, le transport, la santé et les dispositifs portatifs sur le corps, le secteur de la construction (villes intelligentes et environnement), le secteur de la gestion de l'énergie, le secteur de la sécurité et la gestion des désastres et catastrophes naturelles, le secteur de l'industrie, et enfin le secteur de l'agriculture. Les détails de ces applications ainsi que de nombreuses références sont présentés en annexe A.2.

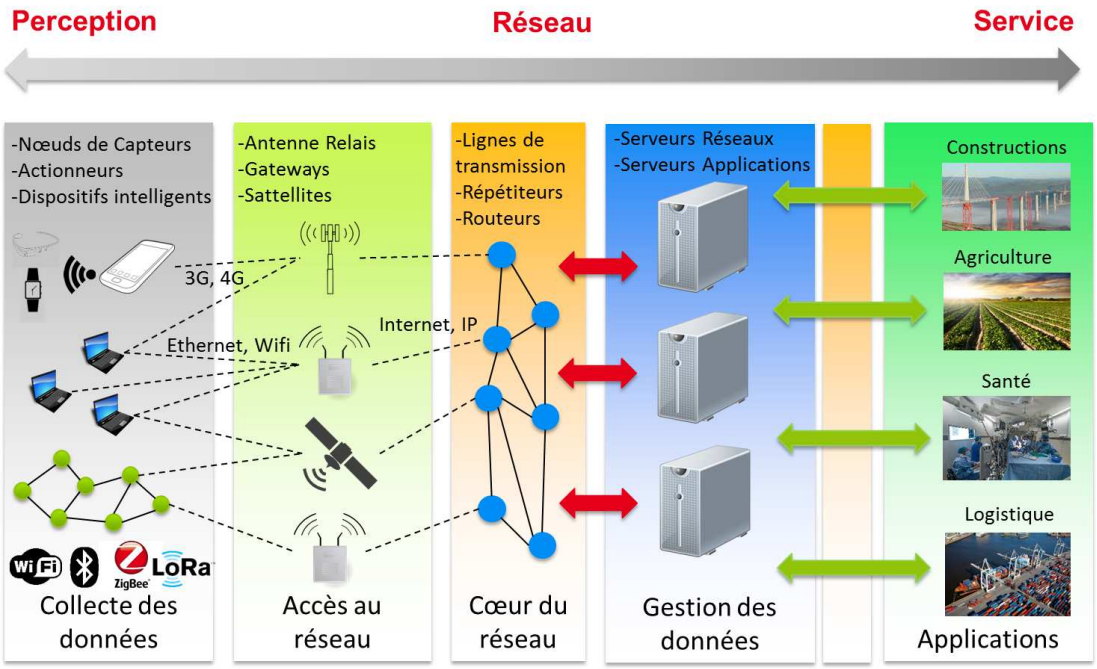


FIGURE 1.3 – Architecture de l'internet des objets

1.4 Synthèse

Le marché des nœuds de capteurs communicants est en pleine expansion et concerne des milliers d'acteurs de tout horizon. Ils vont être utilisés dans une multitude d'applications comme présenté plus haut et vont réellement changer notre manière d'interagir avec l'environnement, et notre manière de se déplacer. Ils vont nous permettre aussi de gaspiller beaucoup moins d'énergie et de matières premières, ce qui est aujourd'hui un enjeu considérable étant donné le contexte écologique et économique. Toutefois, beaucoup d'améliorations restent à faire à tous les étages de l'architecture du réseau de l'IoT (figure 1.3). Dans certaines applications comme la domotique, le biomédical, les dispositifs portables sur le corps, l'agriculture ou la surveillance de l'environnement, les contraintes en énergie sont fortes pour avoir une autonomie la plus grande possible. Ainsi la réduction de la consommation de chacune des entités qui constitue le nœud est une nécessité pour réduire à la fois les consommations moyennes des nœuds et leur courant pic et ainsi réduire la taille de la batterie et donc la taille totale du nœud. Dans cette thèse, nous nous sommes intéressés à la réduction de la consommation du nœud de capteurs dans la couche de perception de la figure 1.2. Nous allons donc voir dans un premier temps ce qui constitue un nœud de capteurs pour ensuite nous concentrer sur la partie microcontrôleur du nœud. Nous étudierons ensuite les différentes pistes de réduction de la consommation de la partie microcontrôleur.

1.5 Nœuds de capteurs sans fil

Les nœuds de capteurs communicants ont bénéficié des avancées considérables de la micro-électronique telles que :

- la réduction de la longueur de canal des transistors ainsi que l'intégration de micro-

systèmes électromécaniques (MEMS [3]) et de Systèmes sur Puce (SoC [4]) contenant le microcontrôleur et la radio par exemple ;
- la réduction de la consommation statique et dynamique des circuits par diverses méthodes qui seront présentées au chapitre 3 ;
- la réduction des coûts de production des différentes puces électroniques grâce à une production de masse et à des procédés de fabrication permettant des rendements toujours plus importants.

Comme présenté au chapitre 1, l'objectif des nœuds de capteurs est d'intégrer de l'intelligence partout où cela est possible afin d'augmenter le confort et la sécurité des personnes, de récupérer des paramètres environnementaux ou physiques afin d'agir sur cet environnement le plus rapidement possible en fonction de l'évolution de ses paramètres, ou encore de faire des économies d'énergie et donc d'argent. Les contraintes sur les nœuds de capteurs communicants peuvent changer d'une application à une autre. Cependant la plupart du temps le nœud de capteurs devra respecter les contraintes décrites ci dessous :

- Consommation : l'application va déterminer l'importance de cette contrainte. Pour certaines applications le nœud de capteurs va être alimenté par le réseau électrique, ainsi sa contrainte en consommation sera nettement moins importante que pour un nœud disposé en pleine nature et dont l'autonomie devra atteindre plusieurs années sans aucune maintenance. Cependant, même dans le cas d'un nœud de capteur alimenté par le réseau électrique, la consommation devra rester toujours très faible dans un esprit écologique car, si dans 5 à 10 ans des milliards de nœuds de capteurs sont ajoutés sur le réseau électrique, il faudra alors construire des centrales électriques supplémentaires afin d'honorer la demande en électricité.
- Coût : le but est d'avoir un réseau très dense de capteurs communicants c'est pourquoi le coût unitaire des nœuds du réseau doit rester assez faible pour atteindre cet objectif.
- Taille : les contraintes de taille vont être vraiment différentes entre une application biomédicale et une application environnementale par exemple. En effet dans la première, le nœud de capteur doit être le plus petit possible ou bien atteindre des tailles millimétriques pour être implanté dans le corps contrairement à la seconde application pour laquelle les contraintes en taille seront fonction du prix du nœud et donc de la matière première utilisée.
- Services : le nœud doit pouvoir fournir de plus en plus de services dans certaines applications et gérer de nombreuses tâches. C'est pourquoi des systèmes d'exploitation (OS [5]) ont été créé pour les nœuds de capteurs communicants ayant une capacité mémoire très limitée afin de faciliter la programmation du microcontrôleur et le portage des applications sur différentes cibles.
- Adaptation à l'environnement : ces nœuds de capteurs vont potentiellement faire face à une très large gamme de température et de variations de tension de la batterie. C'est pourquoi ces circuits intégrés devront rester fonctionnels quelles que soient les conditions environnementales pour garantir un réseau stable.
- Sécurité : l'un des points les plus critiques dans l'IoT est la sécurité des données. Il faut pouvoir garantir que les données échangées n'ont pas été corrompues pendant le transfert du nœud vers les serveurs d'applications. Pour cela différentes techniques existent qui seront vues dans la section 2.1.

Dans l'optique de respecter ces contraintes, le développement du microcontrôleur de

3. Micro ElectroMechanical System
4. System on Chip
5. Operating System

cette thèse est tout particulièrement dirigé par la faible consommation en utilisant différentes techniques de réduction de la consommation, décrites au chapitre 3, par l'ajout de services avec des procédés architecturaux et par l'adaptation à l'environnement avec l'utilisation de la logique asynchrone.

Un nœud de capteurs doit être capable d'effectuer des mesures de paramètres physiques dans son environnement proche, d'effectuer un traitement local sur ces données brutes et d'envoyer les données traitées localement par un lien radio, optique ou filaire à un concentrateur pour être envoyées ensuite vers les serveurs applicatifs. Le nœud peut aussi récupérer de l'énergie de son environnement pour alimenter directement les différentes entités du nœud ou bien recharger sa batterie afin d'augmenter la durée de vie du nœud. L'architecture classique d'un nœud de capteurs communicant est présentée figure 1.4.

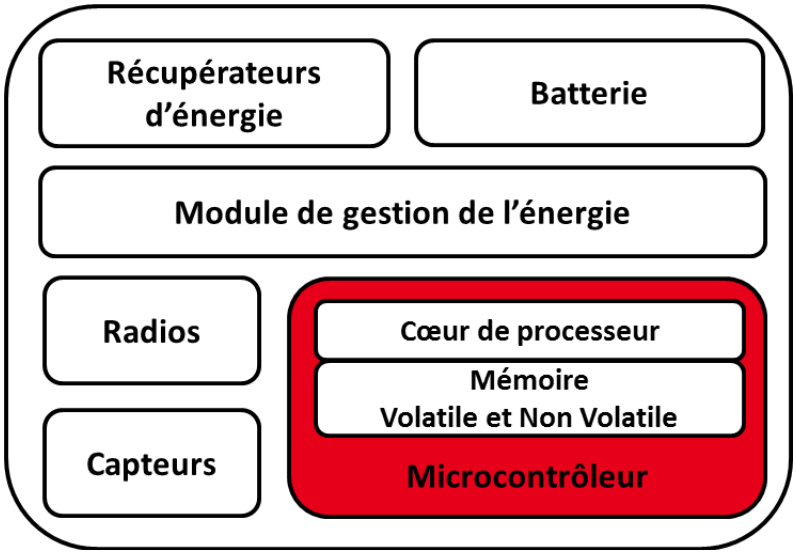


FIGURE 1.4 – Architecture d'un nœud de capteur sans fil

Elle comporte six blocs principaux :

- radios : communication avec le concentrateur ou un autre nœud pour échanger des données ou des informations de contrôle ;
- capteurs : extraction des données physiques de l'environnement ;
- microcontrôleur : gestion de la récupération des données provenant des capteurs ou des radios et gestion du scénario du nœud ;
- récupérateurs d'énergie : transformation d'un certain type d'énergie en énergie électrique ;
- batterie : stockage de l'énergie électrique ;
- module de gestion de l'énergie : conversion et distribution de l'énergie provenant des récupérateurs d'énergie et batterie vers les radios, capteurs et microcontrôleur et recharge de la batterie par conversion de l'énergie provenant des récupérateurs.

La thèse se concentre sur le sous-bloc microcontrôleur du nœud. Néanmoins, dans ce chapitre, chaque sous bloc (radios, microcontrôleur, capteurs) va être présenté avec leurs différentes caractéristiques et leurs différentes contributions en termes de consommation.

1.6 Architecture du nœud de capteurs

1.6.1 Radios

Le domaine de la radiofréquence (RF) est une discipline à part entière dans la microélectronique. L'objectif n'est pas ici de présenter toute la théorie et l'architecture d'une radio mais de présenter les caractéristiques en consommation et les modes de consommation des radios existantes pour les nœuds de capteurs communicants. De plus ces analyses vont être utiles pour la simulation de la consommation d'un nœud de capteurs du chapitre 4.1. Le tableau 8.1 de l'annexe A.1 nous montre les consommations de radios du commerce pour tous les standards existants.

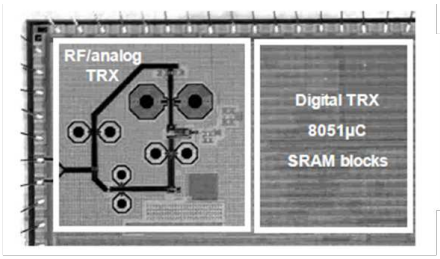
L'un des standards les plus utilisés dans le domaine des réseaux de capteurs communicants est la radio Zigbee conçue pour des communications radiofréquences à 2.4GHz sous le standard IEEE 802.15.4 [78]. L'une des dernières radios d'ATMEL, la AT86RF233 [36] respectant cette norme, consomme $13.8mA$ sous 3V en transmission pour un gain en sortie de $4dBm$, et $11,8mA$ en réception. Son courant de veille atteint $0.2\mu A$. Étant donnés les courants consommés par la radio en réception et transmission, la radio va être mise la plupart du temps dans son état de veille profonde.

Dans la littérature, de nombreuses radios ont été conçues pour le standard IEEE 802.15.4 [44] [66] avec maintenant des radios capables de respecter plusieurs standards [109] [39] comme le Bluetooth LE, le Zigbee et le 802.15.6. La figure 1.5 présente les travaux de Bernier [44] sur un circuit totalement compatible 802.15.4 consommant $5,4mW$ en réception et $8.1mW$ en transmission pour un gain en sortie de $0dBm$ à 1.2V et ceux de Liu sur le Front End RF (RFFE) [109] et Bachmann sur le Digital Base Band (DBB) [39] sur une radio multistandard. Les résultats communs en consommation des deux circuits associés du RFFE et DBB sont présentés dans le tableau de la figure 1.5. Celui-ci indique une consommation de $4mW$ pour le 802.15.4 en réception pour une tension d'alimentation de 1.2V pour le RFFE et 0.74V pour le DBB et une consommation de $5.48mW$ en transmission pour un gain en sortie de $0dBm$. Ces radios atteignent des consommations nettement plus faible que les radios commerciales. De plus, elles arrivent à gérer de plus en plus de standard dans un seul circuit.

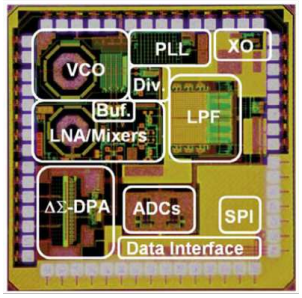
Néanmoins un nouveau type de radio a vu le jour ces dernières années pour réduire la part de la consommation de la radio dans la consommation moyenne du nœud de capteur. Ces radios sont les radios de réveil nommées WUR [6]. Des réalisations de WUR ont été faites dans la littérature en ASIC [7] [131] et avec des composants du commerce [113] [112]. De plus, des couches MAC [8] pour WUR ont été mises en place [100] [96]. Dans un nœud classique la radio est éteinte la plupart du temps et le microcontrôleur est mis dans son mode de consommation le plus bas. Le microcontrôleur réveille périodiquement la radio pour écouter si des données sont transmises sur son canal. Cette méthode de rapport cyclique a un coup en terme de synchronisation pour que le nœud émetteur et récepteur s'échangent des données. Il y a trois types de synchronisation : synchrone (les nœuds sont pré-synchronisés entre eux pour savoir à l'avance quand se réveiller), pseudo-asynchrone (le nœud émetteur envoie un signal de préambule pour indiquer l'intention de transmission qui doit être assez long pour coïncider avec le réveil du nœud récepteur) et asynchrone (les nœuds sont en mode de veille profonde et peuvent être réveillés par leurs voisins sur demande avec une radio de réveil très basse consommation).

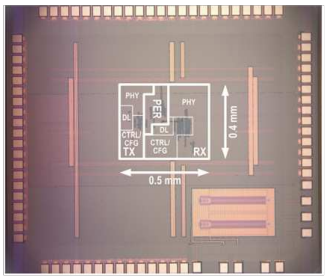
La figure 1.6 présente l'architecture d'un nœud avec une radio principale et une radio de

6. Wake Up Radio ou Wake Up Receiver
7. Application-Specific Integrated Circuit
8. Media Access Control



RX blocks	1.12mA	TX blocks	3.38mA
RF Front-end	440µA	Up-converter	2.73mA
Limiter	150µA	I/Q DAC	400µA
I/Q ADC	50µA	Digital	250µA
RSSI ADC	250nA	**PLL total**	**2mA**
Digital	480µA	Quad. VCO	400µA





Technology	40nm (DBB) and 90nm (RFFE)		
Standard	BTLE	802.15.4	802.15.6
Modulation	GFSK	HS-OQPSK	pi/4-DQPSK
Symbol rate [kS/s]	1000	1000	600
Data rate [kb/s]	1000	250	971.4
RX sensitivity* [dBm]	-98	-101	-96
RX DBB / digital power diss. [µW]	180	200	140
TX DBB / digital power diss. [µW]	80	80	60
RX RFFE power diss. [mW]	3.80	3.80	3.80
TX RFFE power diss. [mW]	5.40	5.40	4.60
RX total power diss. [mW]	3.98	4.00	3.94
TX total power diss. [mW]	5.48	5.48	4.66
RX energy eff. [nJ/b]	4.0	16.0	4.1

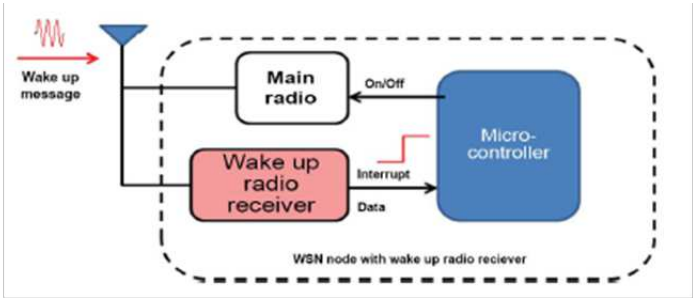
FIGURE 1.5 – Radio 2.4GHz totalement compatible 802.15.4 [44] et compatible multi-standard [109] [39]

réveil. Le diagramme de temps de la figure 1.6 présente le mécanisme d'échange de données entre une source et un destinataire avec un type de synchronisation purement asynchrone. La WUR du destinataire est en écoute et sa radio principale est éteinte. Un signal de réveil est envoyé par la source et reçu par la WUR qui envoie un signal d'interruption au microcontrôleur pour réveiller la radio principale si ce signal est pour ce nœud. La radio principale envoie un signal d'acquittement au nœud source, le nœud source envoie alors le paquet de données et attend l'acquittement de la bonne réception des données par la radio principale du récepteur.

1.6.2 Capteurs

Le but du capteur dans un nœud de capteur est de mesurer une grandeur physique et de la convertir en signal électrique. Ce signal électrique analogique est ensuite converti en signal numérique grâce à des convertisseurs analogique-numérique (ADC [9]) directement dans le capteur ou grâce aux ADC du microcontrôleur. Il existe une multitude de capteurs dont une liste non exhaustive est donnée dans le tableau 8.2 en annexe A.3. Différents types de capteurs sont présentés que ce soit pour des mesures acoustiques, chimiques, environnementales, de déplacements/vibrations/angles, physiologiques, de radiations, de positions, optiques ou mécaniques. Le tableau 8.2 en annexe A.3 montre des exemples de capteurs du commerce qui peuvent être utilisés pour des nœuds de capteurs sans fil. Leur référence, leur interface avec le microcontrôleur ainsi que leur consommation en activité ou au repos sont présentées. La consommation moyenne d'un capteur dans une application dépend fortement de la fréquence d'échantillonnage de celui-ci et est entièrement dépendante de l'application. Il est possible de voir que les consommations de certains capteurs sont relativement élevées que ce soit en mesure (16,5 mA pour le GPS, 3,7 mA pour la centrale inertielle) ou au repos (300 μA pour le GPS et 8 μA pour la centrale inertielle). Il

9. Analog to Digital Converter



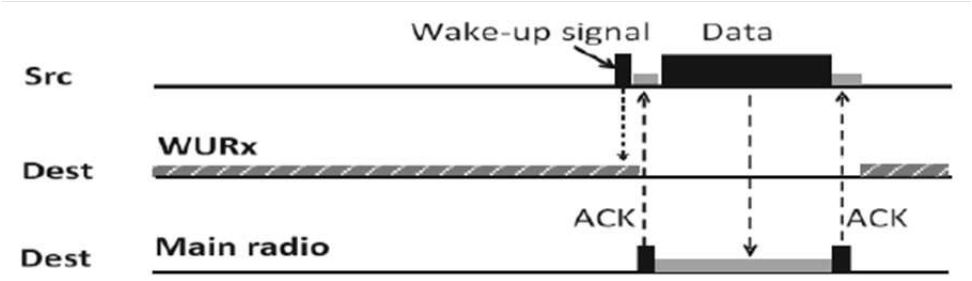


FIGURE 1.6 – Architecture d'un nœud de capteur avec une radio de réveil [113] [112]

faudra donc adapter la fréquence d'échantillonnage du capteur à l'application pour avoir le meilleur rapport consommation/performance. Il serait aussi possible de couper l'alimentation de ces capteurs pour réduire davantage leur consommation statique au repos.

1.6.3 Microcontrôleur

Le microcontrôleur est le centre de calcul et de contrôle du nœud de capteurs. Dans cette section nous allons passer en revue les différentes tâches de celui-ci ainsi que quelques exemples de microcontrôleurs du commerce utilisés pour les nœuds de capteurs communicants et l'ordre de grandeur des consommations. L'architecture du microcontrôleur ainsi que les réalisations de la littérature seront présentées en détail à la section 2.1. Dans un nœud de capteurs, le microcontrôleur est en charge des tâches de transfert des données provenant des capteurs vers la mémoire du microcontrôleur en passant par des périphériques de communications ou par des ADCs. Ensuite, celui-ci doit effectuer des calculs sur ces données brutes pour les transformer en données utiles et agir sur des actionneurs suivant ces valeurs et suivant l'application. Il peut aussi envoyer les données utiles à la radio en passant par des périphériques de communication en gérant la couche MAC de celle-ci pour centraliser ces données sur les serveurs applicatifs et traiter globalement l'ensemble des données provenant de tous les nœuds de capteurs. Le microcontrôleur est aussi en charge, la plupart du temps du réveil du nœud entier grâce à l'utilisation de timer et d'horloges très basse consommation. Les tâches les plus courantes pour le microcontrôleur sont donc les transferts de données de ses périphériques à sa mémoire dans le cadre des mesures sur les capteurs ou de la réception de données par la radio, des transferts de données de sa mémoire à sa mémoire pour les calculs sur les données brutes aux données utiles et des transferts de sa mémoire à ses périphériques dans le cadre de transmissions radio. Les tâches de réveil sont aussi des tâches courantes pour un nœud de capteurs communicant, que ce soit pour allumer ou éteindre ses modules internes ou pour mettre la radio et les capteurs dans des modes de consommation différents. De plus, les ressources en calcul peuvent être plus ou moins importantes suivant l'application. C'est pourquoi des instruc-

tions pour le traitement du signal (instruction DSP [10]) sont implémentées dans le cœur de processeur ainsi que des instructions pour des opérations arithmétiques pour nombres flottants (FPU [11]), comme dans le cœur de processeur ARM Cortex M4 [33]. Certains microcontrôleurs du commerce sont illustrés figure 1.7. Le microcontrôleur STM32L0 [145] de STMicroelectronics utilise l'architecture du Cortex M0+ [32]. Il est capable d'atteindre 0,27 μA dans son mode de consommation le plus bas avec seulement deux lignes d'interruptions disponibles et 6,3 mA de consommation pour une fréquence de 32MHz en mode d'exécution. Les microcontrôleurs SAMD20-21 [37] utilisent aussi l'architecture du Cortex M0+ et atteignent une consommation de 6,32 mA pour une fréquence de 48 MHz et 2,70 μA dans son mode de consommation le plus bas. Une autre gamme de processeurs commerciaux est beaucoup utilisée dans le domaine des nœuds de capteurs communicants, il s'agit de la famille des MSP430 [152] de Texas Instruments. Ils ont des consommations de 9,2 mA à 16 MHz en mode actif et 0,1 μA dans son mode de consommation le plus bas.

FIGURE 1.7 – Exemples de microcontrôleurs du commerce utilisés dans les nœuds de capteurs sans fil. (De gauche à droite) microcontrôleur STM32L0 [145], microcontrôleur TI MSP430 [152], microcontrôleur ATMEL SAMD20 [37]

Il est possible de voir que la consommation des microcontrôleurs peut être élevée en mode actif et que dans le mode de consommation les plus bas, les fonctionnalités restent très limitées pour certaines applications. La section 3 passera en revue les différentes techniques existantes pour réduire la consommation globale du microcontrôleur.

1.6.4 Récupérateurs d'énergie

Comme vu précédemment, pour certaines applications, l'un des objectifs des nœuds de capteurs communicants est d'avoir une très grande autonomie afin d'éviter de remplacer les batteries de ces nœuds trop souvent. Une des techniques pour avoir des nœuds capteurs avec une autonomie presque illimitée est de récupérer l'énergie directement depuis son environnement grâce à des récupérateurs d'énergie. Plusieurs études ont été menées ces dernières années sur énormément de types de récupérateurs d'énergie pour produire de l'énergie localement dans les nœuds de capteurs communicants [84] [88] [167].

Il existe cinq types d'énergie récupérables :

- l'énergie vibratoire, mécanique, mouvement,
- l'énergie thermique,
- l'énergie électromagnétique,

10. Digital Signal Processor
11. Floating Point Unit

- l'énergie solaire, lumière,
- l'énergie biologique.

Ces récupérateurs sont présentés dans le détail en annexe A.4. Chaque récupérateur tiré de la littérature est présenté avec son ordre de grandeur de récupération d'énergie et un tableau comparatif est exposé tableau 8.3 en annexe A.4. Les puissances récupérées sont relativement faibles comparées à la consommation des différentes parties du nœud de capteurs. Le choix du récupérateur (ou des récupérateurs) dépend entièrement de l'application visée. Il faudra donc prendre garde à bien dimensionner les récupérateurs et chaque partie du nœud de capteurs en fonction de l'énergie disponible, et gérer les scénarios en fonction de celle-ci. Le tableau 8.3 de l'annexe A.4 récapitule les principaux résultats pour les différentes techniques de récupération d'énergie.

1.6.5 Stockage de l'énergie

Le stockage de l'énergie dans un nœud de capteurs communicant est primordial. Il faut que cette source de stockage soit capable de supporter les pics de puissance maximale à fournir au nœud (de l'ordre du milliwatt) et à fournir de l'énergie suffisamment longtemps pour respecter le cahier des charges du nœud en rapport avec l'application. Même lorsque des récupérateurs d'énergie sont utilisés, il faut, dans beaucoup de cas, ajouter une source de stockage tampon afin de permettre au système de fonctionner même lorsqu'aucune récupération d'énergie n'est possible. De plus, les ordres de grandeur des puissances récupérées sont la plupart du temps largement en dessous de la puissance pic consommée. C'est pourquoi une source de stockage intermédiaire est souvent nécessaire.

Le diagramme de Ragone de la figure 1.8 permet d'évaluer les différents moyens de stockage avec leur densité massique d'énergie (Wh/Kg) en fonction de densité massique de puissance que ces moyens de stockage (W/Kg) sont capable de supporter.

Aujourd'hui les moyens de stockage les plus utilisés pour les nœuds de capteurs communicants sont les condensateurs/supercondensateurs et les batteries électrochimiques, telles que le Lithium-Ion (Li-Ion) ou Lithium-Polimère (LiPo). Le diagramme de Ragone figure 1.8 montre que les condensateurs/supercondensateurs ont des très grandes densités massiques de puissance avec une très faible densité massique d'énergie. Ils sont donc parfaits pour fournir rapidement beaucoup de puissance mais, en contrepartie, ils ne peuvent pas maintenir cette puissance pendant très longtemps. Les batteries électrochimiques comme les Li-Ion ont une forte densité massique d'énergie mais des faibles densités massiques de puissance. Les LiPo ont une forte densité massique d'énergie (de l'ordre de 250 Wh/kg) et peuvent atteindre aujourd'hui des puissances pic de sortie assez élevées, ce qui permet d'avoir de petites batteries pour des capacités d'énergie et des courants pic élevés. Le problème de ces batteries électrochimique restant le nombre de cycle de recharge.

1.6.6 Module de gestion de l'énergie

Le module de gestion de l'énergie permet de distribuer l'énergie à tous les modules du nœud de capteurs, de convertir la tension d'alimentation provenant du moyen de stockage ou des récupérateurs d'énergie pour les adapter aux tensions de chaque module du nœud mais aussi de recharger le moyen de stockage grâce aux récupérateurs d'énergie quand il y en a.

Une présentation des différents convertisseurs et gestionnaires d'énergie du commerce pour nœuds de capteurs est effectuée en annexe A.5.

Par exemple, Christmann et al. [63] [61] [62] ont développé un gestionnaire d'énergie multi sources de récupérations et de stockages et multi sorties avec un contrôleur asyn-

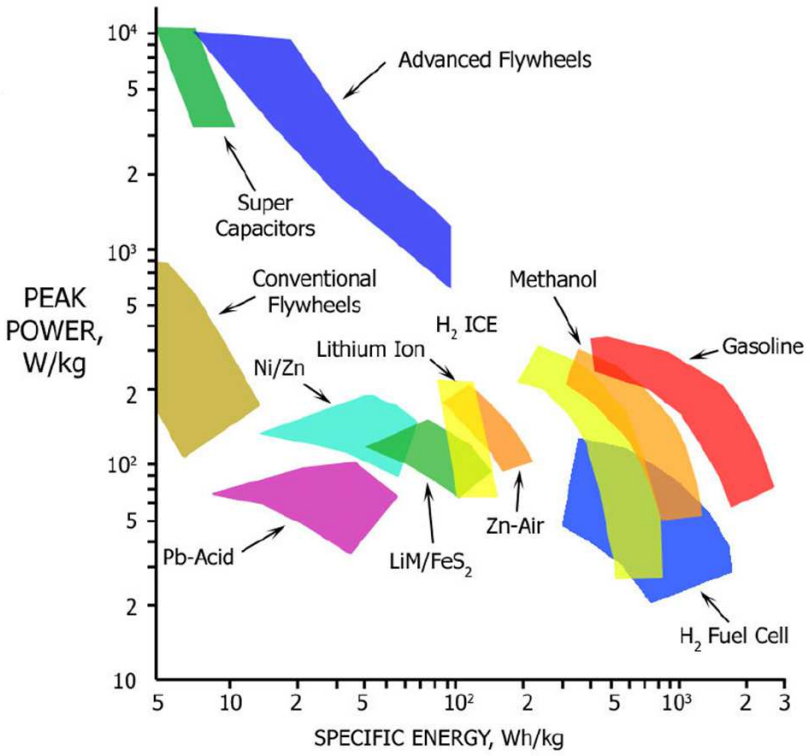


FIGURE 1.8 – Diagramme de Ragone pour différents moyens de stockage d'énergie [8]

chrone et sensible à des évènements énergétiques. Ils ont mis en place une architecture d'alimentation à chemin de puissance multiple avec un chemin direct d'alimentation à haut rendement entre les récupérateurs d'énergie et les charges consommantes (figure 1.9). Ce système permet d'optimiser le rendement énergétique entre les récupérateurs, les capacités et batteries et les charges consommantes. Les auteurs ont développé deux types de requêtes d'énergie, les requêtes immédiates et les requêtes opportunistes. Les requêtes immédiates consistent à fournir immédiatement l'énergie provenant des capacités chargées par les récupérateurs et si l'énergie pour exécuter la tâche n'est pas suffisante alors la batterie fournit le reste de l'énergie. La requête d'énergie opportuniste va permettre, par contre, d'attendre que la capacité soit chargée suffisamment pour l'exécution de la tâche et l'exécute uniquement quand la capacité a atteint le niveau nécessaire. Le gestionnaire d'énergie réussit à obtenir 40% de gain d'efficacité énergétique et une consommation moyenne du contrôleur de $1\mu A$ dans la technologie 180nm. D'autres travaux comme le nœud de capteurs PowWow de Sentieys et al. [43] utilisent un FPGA pour s'occuper des modes de consommation du nœud et de la DVFS [12] du microcontrôleur MSP430. Un gestionnaire d'énergie LTC3108 peut gérer trois types de récupérateurs d'énergie ainsi que la recharge de différents moyen de stockage de l'énergie.

12. Dynamic Voltage and Frequency Scaling

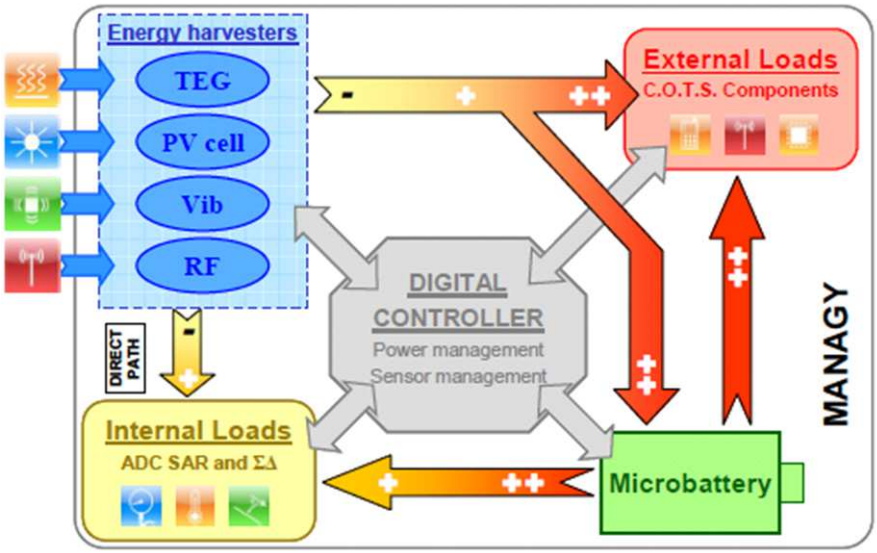


FIGURE 1.9 – Module de gestion des récupérateurs d'énergie, de la rechage de batterie et de la distribution évènementielle de l'énergie dans le nœud de capteurs [63]

1.7 Historique des réalisations de nœuds de capteurs complets

Comme vu précédemment, les nœuds de capteurs sont constitués d'une multitude de composants qu'il faut dimensionner et interconnecter convenablement en fonction des besoins applicatifs. Dans la littérature et dans le commerce, de nombreuses réalisations ont été faites et les principales vont être présentées dans cette section. Pour chacune d'entre elles, une présentation de leur consommation ainsi que leurs caractéristiques sera résumée dans le tableau 1.1.

1.7.1 Démonstrateurs de laboratoires

1.7.1.1 Nœuds de taille millimétrique

Dans la littérature, plusieurs nœuds de capteurs ont été conçus pour atteindre des tailles millimétriques en empilant les différentes puces, batterie ultrafine et récupérateurs d'énergie les uns sur les autres et en les reliant par des fils de connexions (par technique de *wirebonding*) [60] [101] [98]. Dans [60], l'équipe a conçu un nœud de capteurs pour mesurer la pression interne de l'œil dans le cadre de la maladie du glaucome (figure 1.10). Son volume est de $1,5mm^3$ et contient le transceiver pour communiquer à courte portée (10cm), l'unité de gestion de l'énergie avec des cellules photovoltaïques (PV), la batterie ultrafine ainsi que le microcontrôleur et le capteur de pression intraoculaire. Le *transceiver* radio est alimenté par RF au moment de transmettre les données. Ce nœud est capable de consommer en moyenne 5,3nW pour une mesure de la pression interne de l'œil toutes les 15 minutes. Les cellules solaires sont capables de fournir 80,6nW à la batterie en ensoleillement. Le processeur consomme 90nW en mode actif en étant alimenté sous le seuil (Sub-Vt [13]) et 71,8pW en mode de repos le tout dans la technologie 180nm.

Dans [101] l'équipe du professeur Sylvester a développé un nœud de capteurs d'un volume de $1mm^3$ ayant comme principal capteur un imageur de 96×96 pixels consommant

13. Subthreshold

680nJ par image. La partie numérique comporte deux processeurs ARM cortex M0 dont l'un sert aux calculs avec une mémoire volatile pour le traitement de l'image (cœur de traitement du signal conçu dans une puce en technologie 65nm, alors que l'autre sert au contrôle de l'application et est implémentée avec sa mémoire de rétention pour le programme dans une puce de technologie 180nm. Toutes les couches (imageur et cellules solaires de $0,54mm^2$, batterie de $0.6\mu Ah$, cœur DSP, cœur de contrôle et communication optique ainsi que les capacités de découplage) sont empilées et reliées par wirebonding (figure 1.10). Le circuit arrive à atteindre des consommations de $40\mu W$ en mode actif et 11nW en mode de repos. Dans [98], l'équipe du professeur Sylvester a ensuite développé un nœud de volume $2 \times 4 \times 4mm^3$ avec un imageur de résolution supérieure : 160×160 pixels (figure 1.10). Ce nœud est capable de détecter en continu des mouvements avec seulement 304nW de consommation et 15nW en mode de repos. Ce système est toujours basé sur un processeur ARM Cortex M0 et possède une cellule solaire de $2,5mm^2$ pouvant fournir une charge à la micro batterie jusqu'à 120nA.

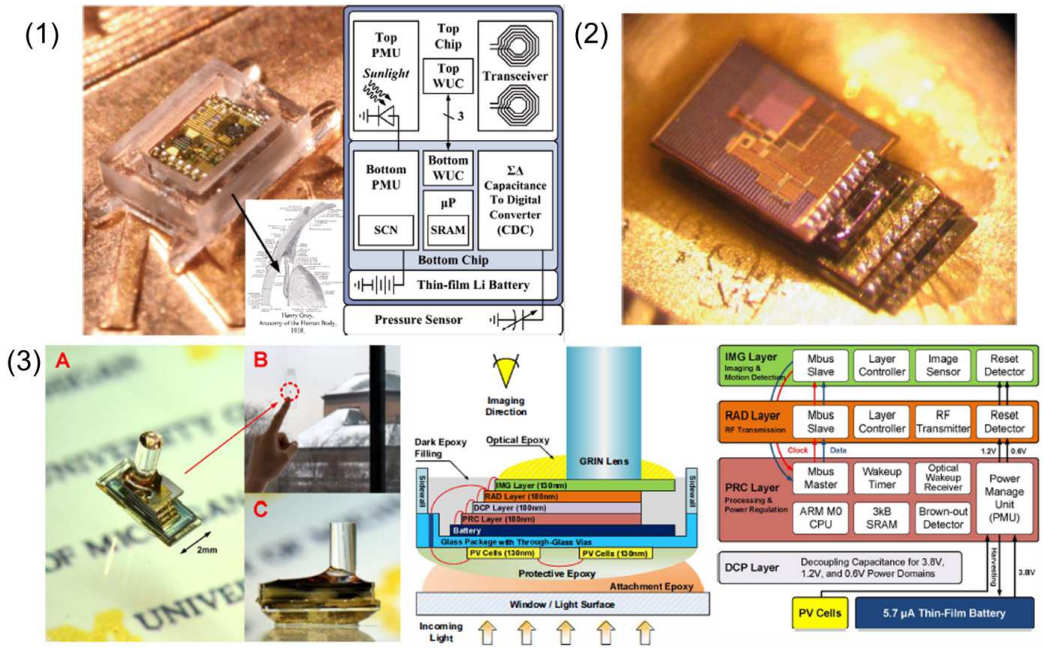


FIGURE 1.10 – Nœuds de taille millimétrique construits par empilement de puce. (1) Nœud de capteurs pour pression intraoculaire [60], (2) nœud de capteurs de température et imageur ainsi que récupération d'énergie par cellule solaire [101], (3) nœud de capteurs avec détection de mouvement en continu et récupération d'énergie par cellule solaire [98]

1.7.1.2 Nœuds de taille centimétrique

Plusieurs équipes ont également créé des nœuds de capteurs de tailles centimétriques avec des composants discrets soudés sur PCB [14]. PicoCube [59] est un nœud de $1cm^3$ composé de 5 PCBs empilés (figure 1.11). Ce nœud intègre un microcontrôleur MSP430, un capteur de pression de pneu (TPMS [15]), un accéléromètre, une radio conçu pour l'application et consommant 1,35mW, des interrupteurs de puissance (power switchs), une batterie NiMH et super-capacité. Ce système est aussi capable de récupérer l'énergie vibratoire

14. printed circuit board : circuit imprimé
15. Tire Pressure Monitoring System

pour alimenter et recharger la batterie. En moyenne PicoCube consomme $6\mu W$ pour l'application de mesure de la pression des pneus. Un microsystème implantable sans fil pour l'enregistrement neuronal multicanaux [138] a été développé par l'équipe de Sodagar. Ce nœud de $1,4cm \times 1,55cm$ permet d'enregistrer simultanément sur 64 canaux l'activité neuronale d'une personne (figure 1.11). Ce nœud étant directement implanté sous la boite crânienne, une alimentation par couplage inductif au niveau de la partie réception de la radio a été nécessaire. Ce nœud consomme en moyenne 14,4mW pour un scan des canaux à 62.5kS/s. Pour cela deux processeurs neuronaux (NPU [16]) sont utilisés pour échantillonner chacun 32 canaux. Dans [43], l'équipe de Sentieys a développé un nœud de capteur PowWow intégrant comme processeur le MSP430, un FPGA [17] pour l'accélération de tâches de calculs comme les codes correcteurs d'erreurs mais aussi pour les tâches de gestion de la puissance et de la fréquence en appliquant de la gestion dynamique de fréquence et de la tension (DVFS [18]) sur le processeur. De plus, le FPGA qui est placé entre le processeur et la radio CC2420 [150], est capable de filtrer les paquets reçus et de réveiller uniquement le processeur quand cela est utile. Il peut aussi modifier dynamiquement la puissance de la radio en fonction des conditions du canal. Toutes ces techniques permettent de réduire considérablement la consommation globale du nœud. Le FPGA consomme de $2.2\mu W$ à 30mW suivant le mode de consommation et ce qui est exécuté pendant que le processeur consomme $330\mu A$ à 1 MHz et 2,2 V et $1,1\mu A$ dans le mode de veille. Ce nœud est capable aussi de pouvoir gérer différents récupérateurs d'énergie ainsi que différents types de stockage d'énergie.

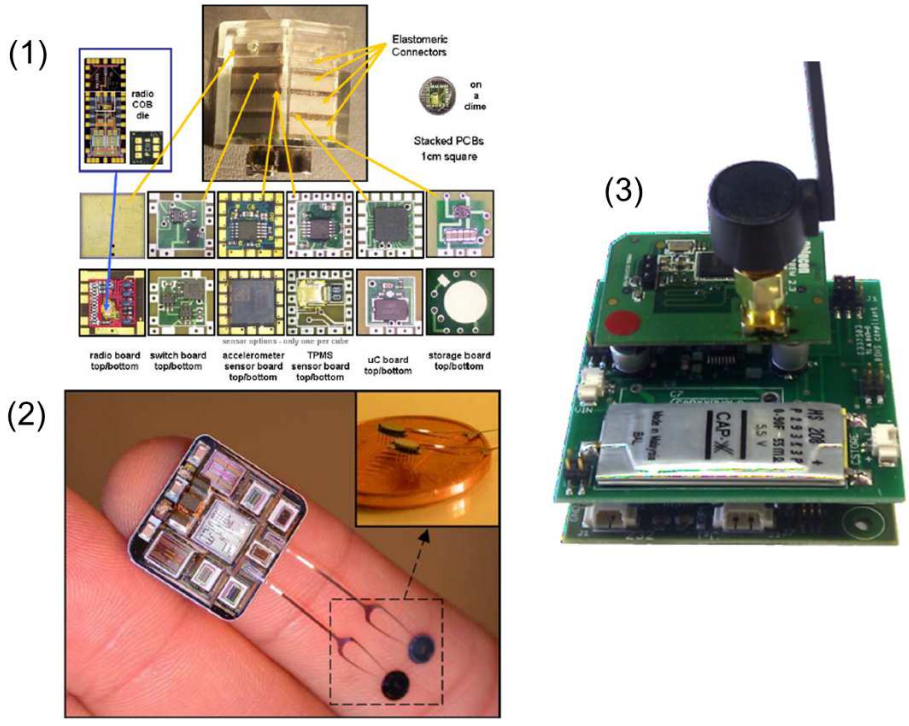


FIGURE 1.11 – Nœud de taille centimètrique. (1) PicoCube [59], (2) microsystème implantable sans fil pour l'enregistrement neuronal multicanaux [138], (3) PowWow [43]

16. Neural Processing Unit
17. Field-Programmable Gate Array
18. Dynamic Voltage and Frequency Scaling

1.7.2 Nœuds de capteurs commerciaux

Dans le commerce, beaucoup de nœuds de capteurs ont également été développés. Le nœud de Texas Instrument TIDA-00374 [159] intègre un capteur de température/humidité et un système sur puce (SoC [19]) CC2650 [156] contenant la radio, le processeur principal et un processeur pour les tâches courantes. La radio contient elle-même un processeur ARM cortex M0 pour gérer la couche MAC (Bluetooth, Zigbee, 6LowPAN, RF4CE). Le processeur principal est un ARM Cortex M3 et le processeur pour les tâches courantes, appelé Sensor Controller, est un processeur avec un jeu d'instructions 16-bit semblable à celui du MSP430. Ce SoC intègre bien sûr tous les périphériques nécessaires pour des nœuds de capteurs communicants. Ce nœud intègre aussi un power switch pour couper l'alimentation du SoC et du capteur piloté par un timer, le tout alimenté par une pile bouton 3V. Il atteint des consommations de 183nA en mode de veille et 4mA en mode actif pour une durée de vie estimé à 10 ans. Libelium a développé un nœud de capteurs [106] très modulaire pouvant accueillir énormément de types de cartes filles différentes suivant l'application visée. Cette carte principale intègre un microcontrôleur ATmega 1281 d'ATMEL. Elle consomme 15mA en mode actif et $55\mu A$ en mode de veille. N'importe quel type de radio (Zigbee, Bluetooth, LoRa, Sigfox, Wifi, 3G, NFC) et différents types de capteurs sont proposés par Libelium (GPS, gaz, détection de mouvement, capteur pour l'eau, acoustique, agriculture, caméra, radiation...) pour les brancher directement sur la carte principale.

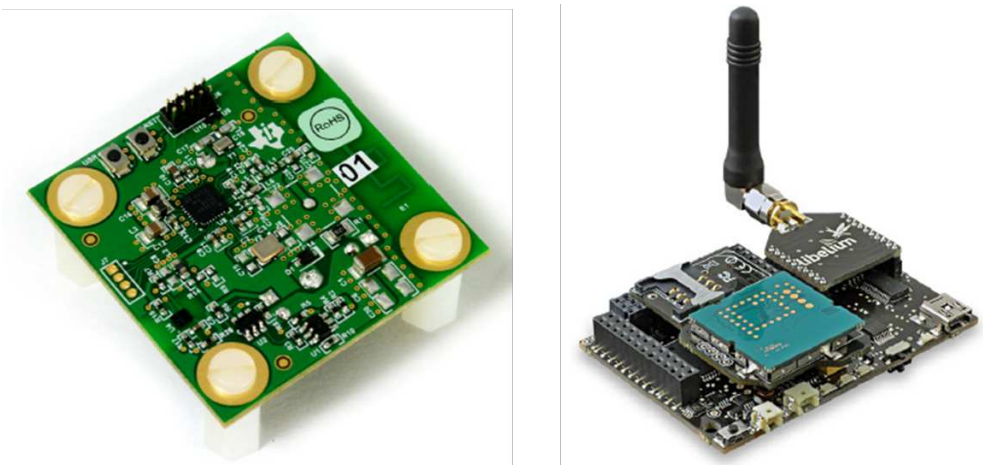


FIGURE 1.12 – Nœuds commerciaux. Plateforme de développement TI TIDA-00374 [159] et Waspmote de Libelium [106]

1.7.3 Synthèse

Nous avons pu étudier l'architecture d'un nœud de capteurs communicant ainsi que la consommation de différentes réalisations, que ce soit celles du commerce ou de la littérature. La consommation d'un nœud peut varier de quelque nW en mode de veille à plusieurs mW lorsque la radio communique ou que des mesures sont faites sur les capteurs. Lorsque l'application et les contraintes sont parfaitement définies, il est alors possible d'atteindre des nœuds de taille millimétrique et donc d'atteindre des consommations de l'ordre de la centaine de nW en mode actif ce qui permet de récupérer l'énergie dans son environnement pour une autonomie presque perpétuelle. Malheureusement ces nœuds de capteurs sont le

19. System on Chip

plus souvent dédiés à une application pour atteindre des consommations aussi faibles et manquent considérablement de flexibilité. C'est pourquoi des nœuds de taille centimétrique qui utilisent des composants d'étagère (COTS [20]) sont développés pour englober énormément d'applications. Évidement les consommations sont plus importantes mais la modularité du nœud est énorme ce qui est très important dans l'industrie pour développer rapidement des prototypes et pour avoir très peu de modifications du logiciel embarqué. Toutefois, la consommation de chacune des entités du nœud doit continuer à être réduite pour augmenter l'autonomie de ces dispositifs. Différentes techniques de réduction de la consommation ont pu être abordées en étudiant les nœuds de capteurs complets mais seront présentées dans le détail au chapitre 3. Le travail de cette thèse se portant tout particulièrement sur la réduction de la consommation du microcontrôleur, il va être présenté, dans la prochaine section, les architectures de microcontrôleur ainsi que les réalisations de la littérature et du commerce.

20. commercial off-the-shelf

Nœud	Microcontrôleur			Radio			Capteurs	Alimentation	Année	Ref
	Ref	Active	Sleep	Ref	Rx	Tx				
P° Monitor	NC	90nW	71.8pW	ASIC	-	47mW	P° intra oculaire	0.4V 3.6V Bat, PV	2011	[60]
1mm3 sensor	2 Cortex M0	$40\mu W$	11nW	Optique	NC	NC	Imageur, T°	3.2-4.1V, Bat, PV	2012	[102]
1mm3 motion	Cortex M0	304nW	15nW	Optique RX, RF TX	NC	NC	Imageur	3.8V Bat, PV	2014	[98]
PicoCube	MSP430	$2.7mW$	$<1\mu W$	[58]	NC	$1.35mW$	P°, T°, α, V_{supply}	$1.2V$ NiMH	2008	[59]
Neural Rec	ASIC NPU	$14.4mW$	NC	ASIC RF	NC	NC	sondes 64 cannaux	1.5V-3V RF	2009	[138]
PowWow	MSP430, FPGA	1.5mW	$7.8\mu W$	CC2420	56mW	52mW	T°	TEG, PV, Piezo, Bat/Capa 2.3V-3.3V	2010	[43]
Waspmote	ATmega 1281	15mA	$55\mu W$	Zigbee ou autre	92mW	100mW	T°, α, Gaz, Water, Cam, Rad ...	3.3 V Bat, PV	2015	[106]
TIDA-00374	CC2650	4.04mA	183 nA	CC2650	5.9mA	9.1mA	T°/ Humidity	$3V$ coin cell	2016	[159]

TABLE 1.1 – Tableau récapitulatif des spécifications et performances des nœuds de capteurs communicants complets

Chapitre 2

Microcontrôleurs pour nœuds de capteurs sans fil

2.1 Architecture des microcontrôleurs

Le microcontrôleur s'occupe des tâches suivantes dans un nœud de capteurs :

- gestion des modes de puissances et fréquences de ses modules internes, des capteurs, des radios et de l'unité de gestion de l'énergie,
- gestion du transfert de données entre les capteurs et sa mémoire ainsi qu'entre la mémoire et la radio,
- gestion des protocoles de communication série avec les capteurs et les radios,
- gestion de la couche MAC [1] de la radio,
- algorithmes de calcul pour transformer les données bruts en données compréhensibles et utiles à l'application,
- gestion de la maintenance du nœud (mise à jour logiciel) et des commandes utilisateurs,
- sauvegarde temporaire des données dans une mémoire non volatile pour un traitement global par des algorithmes de fusion de données,
- sécurisation des transferts radio de données avec des méthodes de chiffrement.

Ces tâches sont donc très hétérogènes et requièrent beaucoup de matériel et logiciel aux domaines de compétences très larges. En effet les microcontrôleurs sont un mélange entre une conception électronique analogique et numérique tout en ayant connaissance des problématiques de la technologie utilisée et des problématiques logicielles. Le microcontrôleur est donc l'intersection du logiciel du matériel et de la technologie. Nous allons dans cette section décrire l'architecture d'un microcontrôleur classique et décrire les différents modules qui le composent ainsi que les différentes alternatives qui existent pour chacun de ces modules. Le but est d'analyser les différentes composantes d'un microcontrôleur afin de connaitre les différentes sources de consommations dans celui-ci.

2.1.1 Architecture globale d'un microcontrôleur

La figure 2.1 présente l'architecture d'un des derniers microcontrôleurs de ST Microelectronics pour les applications ultra basse consommation [146] intégrant un cœur ARM Cortex M0+ et pouvant être utilisé pour les nœuds de capteurs communicants.

On peut y voir huit principaux blocs :

- le cœur de processeur,
- la mémoire non volatile (pour le programme),
- la mémoire volatile (pour les données),

1. Media Access Control

- les *timers*,
- les périphériques de communication,
- les accélérateurs,
- les entrées-sorties,
- le gestionnaire de puissance, des horloges et des fréquences.

Nous allons détailler une partie de ces modules dans les sections qui suivent.

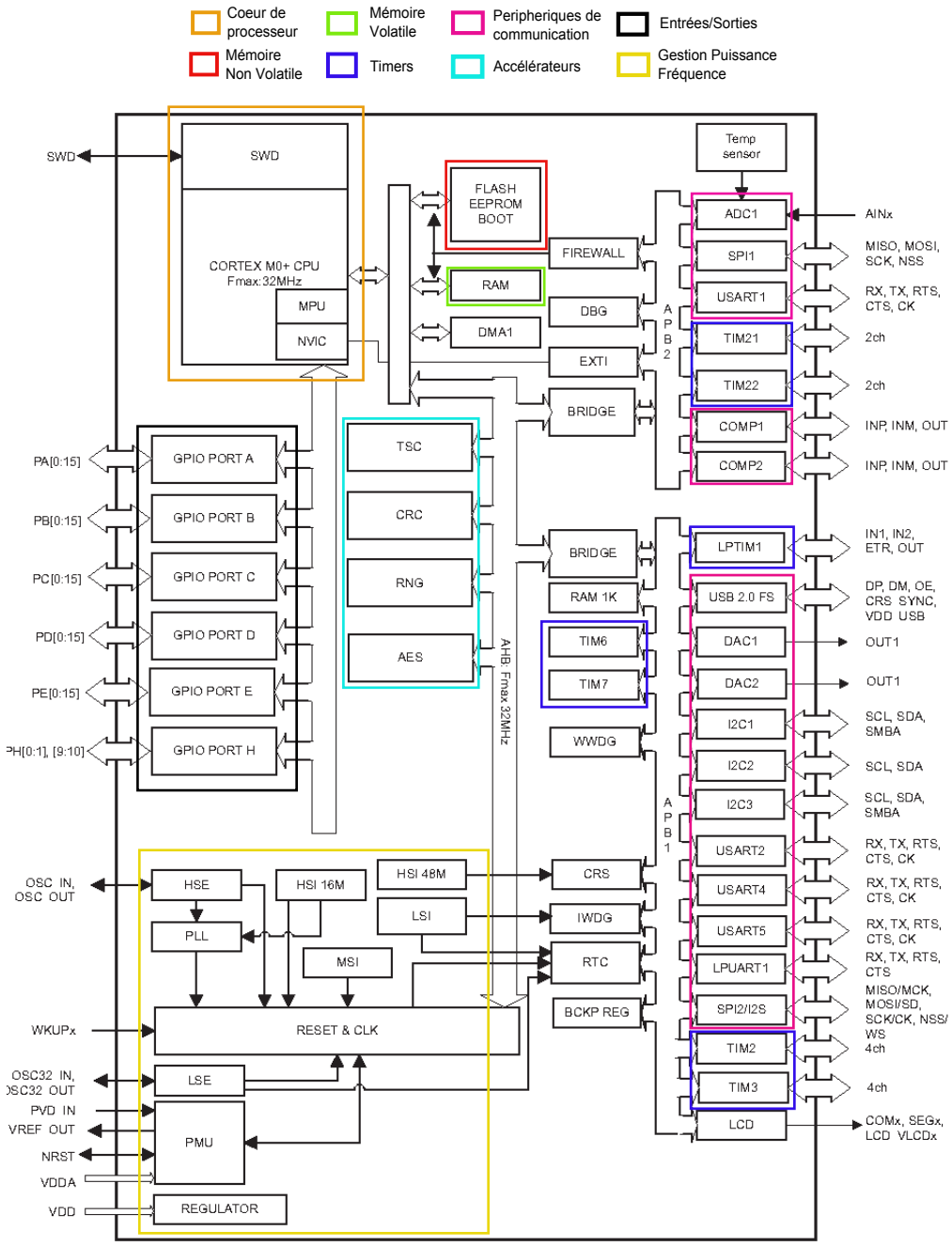


FIGURE 2.1 – Architecture du microcontrôleur STM32L0 avec les principaux modules le composant [146]

2.1.2 Cœur de processeur

Le cœur de processeur s'appuie sur un ensemble d'instructions machine qu'il est capable d'exécuter. Un programme, écrit en langage C par exemple, est compilé pour une cible en utilisant uniquement les instructions que ce cœur est capable d'exécuter. Ces instructions machines sont regroupées dans un jeu d'instructions qui peut être implémenté avec des microarchitectures différentes. Il existe plusieurs familles de jeux d'instructions dont les plus utilisées sont les jeux d'instructions étendus (CISC [2]), comme le jeu d'instructions Intel X86, [89] et les jeux d'instructions réduits (RISC [3]), comme le jeu d'instructions ARM [31]. Les processeurs de type RISC se traduisent par un ensemble d'instructions de taille fixe dont le décodage est plus simple et donc plus rapide contrairement au CISC dont la taille des instructions est variable. De plus, les processeurs RISC possèdent moins de modes d'adressage que le CISC et utilisent beaucoup leurs registres généraux. Les processeurs RISC sont basés sur les instructions du type chargement/stockage (*load/store*). Le code généré pour les RISC est souvent plus volumineux que pour le CISC car une instruction CISC est plus complexe (elle peut donc effectuer plus d'opérations en un seul mot) et se décompose potentiellement en plusieurs instructions RISC.

La microarchitecture d'un processeur définit tous les éléments contenus dans un processeur et l'interconnexion entre ces différents éléments. Elle définit aussi les chemins de données et de contrôle entre les différentes unités, ainsi que les différents étages de pipeline pour accélérer le débit d'instructions. La microarchitecture définit aussi le nombre d'unité d'exécution ainsi que les caches et contrôleurs mémoire.

Un exemple de pipeline est donné figure 2.2. Cette architecture de pipeline, extraite de la documentation du processeur LEON3 [65] et implémentant le jeu d'instruction Sparc V8, est définie par les sept étages de pipeline suivants :

- *Fetch* : chargement de l'instruction ;
- *Decode* : Décodage de l'instruction et génération des adresses de branchement et d'appel à fonction si décodée ;
- *Register Access* : Les opérandes sont lues depuis le banc de registres ou depuis l'immédiat de l'instruction ;
- *Execute* : opération logique et arithmétique ou calcul de l'adresse pour les opérations mémoires, les sauts ou retours d'interruptions ;
- *Memory* : le cache de données est lu ou écrit dans cet étage ;
- *Exception* : gestion des interruptions, fautes et exceptions ;
- *Write Back* : le résultat des opérations arithmétiques et logiques ou opérations sur le cache sont écrites dans le banc de registres.

Lorsqu'un pipeline est implémenté, il faut que le processeur soit alors capable de détecter les aléas de contrôle dus aux branchements, exceptions ou interruptions. Il doit être capable de gérer l'interdépendance entre les données lorsque deux instructions proches ou consécutives essayent d'accéder à la même donnée en écriture ou en lecture dans un registre ou une adresse mémoire (*Read after Read*, *Read after Write*, *Write after Read* et *Write after Write*) ou encore détecter les aléas structurels, lorsqu'une partie du processeur doit être utilisée par deux instructions en même temps. Il faut alors résoudre tous ces problèmes avec des solutions logicielles (*Branch Delay Slot*, ré-ordonnancement des instructions par le compilateur, etc.) ou matérielles (prédiction de branchement, renommage de registres, ajout de bulles dans le pipeline, etc.).

Ainsi l'implémentation de la microarchitecture d'un processeur se caractérise par les

2. Complex Instruction Set Computer
3. Reduced Instruction Set Computer

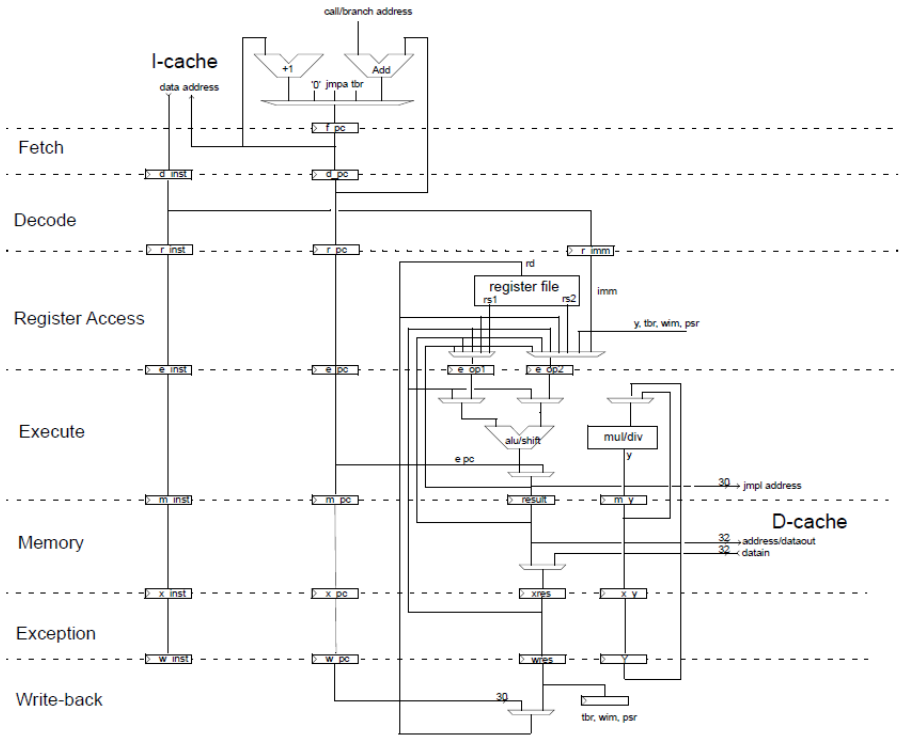


FIGURE 2.2 – Pipeline du processeur LEON3 basé sur le jeu d'instructions Sparc V8 [65]

différents choix suivants :

- le débit d'instructions,
- la latence des différents modules,
- la surface,
- le type d'architecture (Harvard/Von Neumann),
- le nombre d'étages de pipeline,
- le type de jeu d'instructions,
- les problèmes d'interdépendances des données et aléas,
- l'utilisation de cache d'instructions et de données,
- la prédiction de branchements,
- le choix d'exécution dans l'ordre ou dans le désordre,
- plusieurs unités d'exécution (superscalaire),
- l'implémentation d'instructions spécifiques pour le traitement du signal ou les nombres flottants.

Ces choix vont faire que des processeurs avec des jeux d'instructions identiques ou différents auront des performances et des consommations différentes pour une même technologie.

2.1.3 Mémoires

Il existe deux types de mémoire dans un microcontrôleur : les mémoires volatiles et les mémoires non volatiles. La mémoire non volatile permet la rétention des données lorsqu'elle n'est plus alimentée. Elle est typiquement utilisée pour le programme à exécuter. La mémoire volatile est l'endroit où les données et variables sont manipulées et traitées temporairement. Elle perd toute information lorsqu'elle n'est plus alimentée. La mémoire

est l'une des sources les plus consommantes du microcontrôleur, que ce soit lors de l'exécution d'instructions de lecture/écriture en mémoire ou encore lors des phases de veille pour la rétention des données en mémoire volatile. C'est pourquoi beaucoup de recherches sur les mémoires volatiles et non volatiles sont menées pour augmenter leur densité, diminuer leur consommation statique et dynamique tout en diminuant leur tension d'alimentation et en proposant de nouvelles architectures. De nouveaux procédés, matériaux et principes physiques sont utilisés afin de rendre une mémoire non volatile et ainsi couper les mémoires pendant les longues phases de veille.

2.1.3.1 Mémoire Volatile

Les mémoires de données les plus utilisées aujourd'hui dans les microcontrôleurs sont les mémoires volatiles de type SRAM [4]. Une SRAM classique 6T [5] est très rapide d'accès, possède une endurance illimitée et s'intègre parfaitement dans les procédés CMOS. Par contre, sa densité est mauvaise par rapport aux mémoires de type flash car la *bit-cell* contient à elle toute seule 6 transistors. Ces mémoires sont aussi utilisées pour les mémoires cache des processeurs, voire même pour les mémoires de programme quand aucun autre type de mémoire ne peut être intégré dans le circuit [98]. Ces caractéristiques sont présentées dans le tableau 2.1 pour comparaison avec les mémoires non volatiles. Ces SRAM peuvent être implémentées avec des *bit-cells* de 8T jusqu'à 14T pour obtenir des consommations dynamiques/statiques très faibles et des tensions d'alimentation plus ou moins basses.

2.1.3.2 Mémoire Non Volatile

Aujourd'hui la mémoire flash est la plus utilisée en tant que mémoire de programme dans les microcontrôleurs. Elle est de type NOR pour pouvoir exécuter le programme depuis celle-ci. Elle est très dense mais possède des temps d'accès assez long. De plus, son endurance est assez faible ($1.10^5 cycles$) et ses tensions d'alimentation la rendent difficilement compatible avec les procédés CMOS. De nouvelles mémoires non volatiles émergent telles que les mémoires résistives (RRAM [6] et les mémoires magnétiques MRAM [7]). La RRAM a une très bonne densité, de bonnes performances en temps d'accès et une parfaite compatibilité avec le CMOS. Sa durée de vie reste encore limitée mais la recherche est en cours pour l'augmenter et améliorer l'étape de *forming* [8]. La MRAM, quant à elle, possède de très bonnes performances en temps d'accès (voir tableau 2.1 mais une densité assez mauvaise ainsi qu'une compatibilité avec les procédés CMOS plus difficile.

2.1.4 Horloge

L'une des sources de consommation importantes d'un microcontrôleur pour nœuds de capteurs communicant est l'horloge utilisée. En effet, le nœud de capteur a la plupart du temps un rapport cyclique de fonctionnement. Celui-ci est mis dans son mode de consommation le plus faible possible et se réveille sur interruption provenant soit de l'extérieur soit d'un timer (cadencé par une horloge très basse consommation). Ces horloges étant tout le temps actives, elles représentent une très grosse part de la consommation pendant

4. Static Random Access Memory
5. Le point de mémoire (bit cell) contient six transistors
6. Resistive Random Access Memory
7. Magnetic Random Access Memory
8. formage : claquage partiellement réversible du matériau actif

Caractéristiques	SRAM 6T	Flash NOR	RRAM	MRAM
Rétention	Non	Oui	Oui	Oui
Compatibilité avec CMOS	++	- -	++	-
Densité	$80 - 100f^2$	$40 - 50f^2$	$45 - 57f^2$	$> 50f^2$
Maturité	++	++	- -	-
Cycle d'écriture	100ps-10ns	0.1ms	10-100ns	20ns
Endurance	illimité	1.10^5	1.10^6	1.10^{15}
Avantages	performances, compatibilité CMOS	densité	performances, densité	performances, endurance
Désavantages	densité	tensions, endurance, performances	endurance, forming	coût de la technologie, densité, compatibilité CMOS

TABLE 2.1 – Caractéristiques des mémoires existantes et émergentes pour les microcontrôleurs

les périodes de veille. La précision de ces horloges de réveil est aussi importante pour des communications entre deux nœuds par exemple. Si le réveil ne s'effectue pas à un moment précis entre les deux nœuds alors cela engendrera des consommations supplémentaires pour l'un des deux nœuds [103]. Il existe différents type d'oscillateurs :

- oscillateurs à quartz,
- oscillateurs interne CMOS,
- oscillateurs à base de MEMS.

Ces oscillateurs ont des consommations, des précisions et des surfaces bien différentes. Néanmoins, beaucoup de recherches sont en cours afin de réduire la consommation de ces oscillateurs et de la rendre la plus faible possible avec le moins de dérive en température et en tension. Par exemple, dans [173], l'oscillateur à quartz pour une fréquence de 32,768KHz consomme 5,58nW pour une très faible dérive en température de 0.04ppm/°C, alors que l'oscillateur interne CMOS [107] pour une fréquence de 11Hz consomme 150pW pour une dérive en température importante de 490ppm/°C. L'oscillateur MEMS créé dans [56] consomme quant à lui 660nW pour une fréquence de 32,768KHz.

2.1.5 Module de sécurité et autres accélérateurs matériels

Dans une application pour nœuds de capteurs, certaines fonctions logicielles répétitives et coûteuses en terme de temps d'exécution peuvent être remplacées directement par des accélérateurs matériels. Cela permet de réduire considérablement le temps d'exécution et donc de réduire la consommation due à ces tâches. Dans les microcontrôleurs pour nœuds de capteurs, beaucoup d'accélérateurs existent pour assister à la fois les algorithmes de sécurité ainsi que les transferts radios. On peut voir figure 2.1 que le microcontrôleur STM32L0 [146] intègrent deux accélérateurs pour la sécurité (un algorithme de chiffrement symétrique AES [9] de 128 bits et un générateur de nombre aléatoire RNG [10]) ainsi qu'un accélérateur pour les communications (un module de contrôle de redondance cyclique CRC [11]), très utilisé dans les transferts de données radio pour savoir si les paquets reçus sont corrects ou non. L'AES permet, quant à lui, de chiffrer et déchiffrer les données échangées par radio grâce à une clé privée.

9. Advanced Encryption Standard
10. Random Number Generator
11. Cyclic Redundancy Check

2.1.6 Unité de gestion de l'énergie et des horloges

Dans un microcontrôleur il est aujourd'hui possible de choisir par quelle type d'horloge (quartz, oscillateur interne ou FLL [12]) la logique et certains périphériques sont cadencés et ainsi éteindre les horloges des périphériques ou du cœur lorsqu'ils ne sont pas utilisés. C'est ce que l'on appelle le *clock gating*. Ce module, en plus de sélectionner, distribuer et activer les horloges, permet de sélectionner le mode de puissance (*power mode*) pour venir couper l'alimentation du cœur et des périphériques (*power gating*) ainsi que de sélectionner la tension d'alimentation de la logique. Les derniers microcontrôleurs possèdent un grand nombre de *power mode* [144] pour mettre le microcontrôleur dans son mode de consommation le plus faible en fonction de l'application. Mais la plupart du temps, le mode de consommation le plus bas du microcontrôleur ne permet que quelques sources de réveil (2-3 au maximum) et coupe la mémoire de données. Il faut donc sauvegarder les données les plus importantes avant d'utiliser ce mode de consommation.

2.1.7 Périphériques

La section sur le nœud de capteurs a montré que les radios et les capteurs avaient des interfaces de communications soit analogiques (ADC, DAC) soit numériques utilisant des protocoles de communication série ou sortant une information sous forme d'interruptions externes. Il faut donc que le microcontrôleur possèdent un nombre de périphériques suffisants et assez paramétrables afin de pouvoir s'interfacer avec n'importe quel composant du commerce. Ainsi, comme montré figure 2.1, le microcontrôleur intègre une multitude de périphériques de communication série (USART, SPI, I2C, USB), ainsi qu'une multitude de *Timers* pour s'adapter à différentes applications. Il intègre aussi un convertisseur analogique-numérique ADC pour s'interfacer avec certains capteurs ou encore un contrôleur d'interruptions externe pour configurer la sensibilité des fronts d'interruptions. Néanmoins, certaines liaisons série peuvent coûter cher en terme de consommation comme l'I2C dû au résistance de tirage qui peuvent à elle seul consommer une centaine de microwatt [132]. Il faut donc bien sélectionner le périphérique de communication à utiliser en fonction du budget de consommation de l'application.

2.2 Réalisations de microcontrôleurs

Quelques références de microcontrôleurs du commerce ont déjà pu être présentées dans les sections précédentes. Cette section va exposer dans le détail les réalisations de microcontrôleurs de la littérature et du commerce en présentant leurs différentes caractéristiques, leurs performances et leurs consommations. Le tout est résumé dans le tableau 2.2.

2.2.1 Système sur puce de laboratoires

Le microcontrôleur SNAP [72] de Cornell University présenté figure 2.3 a été conçu en logique asynchrone QDI [13]. Ce processeur possède un jeu d'instructions 16 bits avec un chemin de données de 16 bits. Son architecture est assez particulière puisqu'elle exécute du code uniquement quand les unités comme *Message Coprocessor* et *Timer Coprocessor* insèrent des évènements dans la queue d'évènement. Il exécute son code depuis une RAM d'instruction et possède aussi une RAM de données. Il a une très grande plage de tension d'alimentation de 0,6V à 1,8V avec des temps de réveil du mode veille au mode actif

12. Frequency Locked Loop
13. Quasi Delay Insensitive

de 21,4ns pour 0,6V à 2,5ns pour 1,8V. Il consomme en moyenne 24pJ par instruction à 0,6V pour une performance de 28MIPS et 218pJ par instruction à 1,8V pour un débit de 240MIPS. Ses caractéristiques sont résumées figure 2.2. Ce processeur possède donc de très bon chiffres en consommation, performances et réveil en utilisant une architecture originale, un ISA fait spécialement pour les applications de nœuds de capteurs communicants et de la logique asynchrone choisie pour sa large gamme de tension de fonctionnement et sa robustesse face à ces variations de tension. Malheureusement il peut être assez difficile à utiliser pour un programmeur étant donné l'architecture du processeur et son jeu d'instruction. En effet, les périphériques "Timer" et "Message Coprocesseur" sont presque intégrés dans l'architecture du cœur de processeur et celui-ci possède des instructions spéciales pour ces périphériques, ce qui engendre un très grand manque de modularité. De plus il ne contient aucune solution de debug.

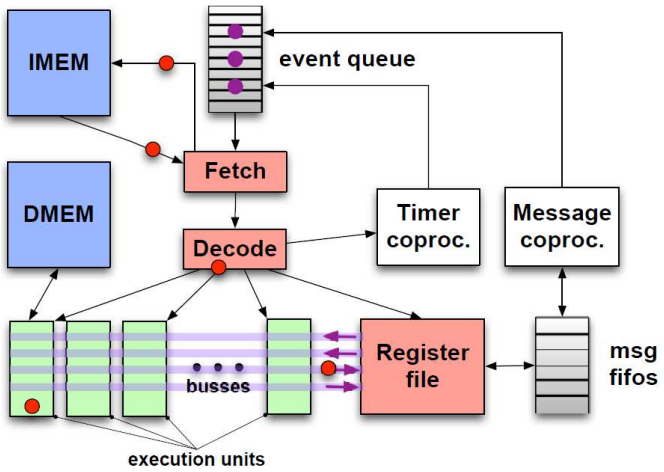


FIGURE 2.3 – Architecture du microcontrôleur SNAP/LE [72]

ULSNAP [126] est une évolution du microcontrôleur SNAP [72] implémenté en technologie 90nm. Il est implémenté en logique asynchrone QDI utilisant plusieurs famille de logique QDI (PCHB [14]/PCFB [15] et WCHB [16]). Son architecture reste similaire à SNAP car il exécute du code uniquement sur évènements, possède un ISA similaire à SNAP (figure 2.4) mais dispose d'un compilateur. Les évènements sont gérés par le module Event Handler (EH) et le vecteur d'interruption est contenu dans le module Event Register Table (ERT) pour un réveil ultra rapide. Le circuit possède de bons chiffres en consommation puisqu'il atteint 29pJ par opération à 0,95V pour 47MIPS et 47pJ par opération à 1,2V et 93MIPS. En mode de veille il consomme $4\mu W$ pour 0,95V d'alimentation et $9\mu W$ pour 1,2V d'alimentation et ceci sans *ckock gating* et *power gating*. Son temps de réveil est aussi très rapide (6,5ns) et exécute la première instruction 40ns après l'arrivée de l'évènement. Néanmoins sa plage de fonctionnement est plus faible que pour celle de SNAP et il n'intègre toujours pas de solution de debug. De plus sa modularité reste encore faible car les périphériques sont encore très liés au cœur.

Le processeur Phoenix [136] créé en 2008 à l'université de Michigan utilise quant à lui la technique de *power gating*, la compression de données dans les mémoires et l'utilisation à la fois de transistors HVT et MVT utilisés en fonction des besoins en termes de fuite de courant et de vitesse. Ce processeur dont l'architecture est présenté figure 2.5 atteint

14. Precharge Half Buffer
15. Precharge Full Buffer
16. Weak Condition Half Buffer

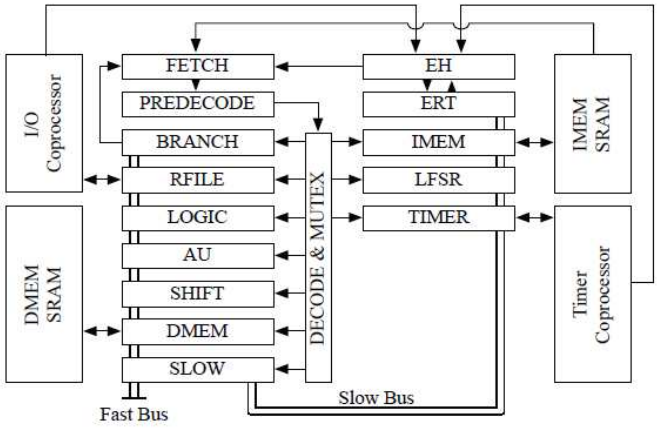


FIGURE 2.4 – Architecture du microcontrôleur ULSNAP [126]

une consommation de 29,6pW en mode de veille sous 0,5V et 2,8pJ/cycle en mode actif pour une fréquence de seulement 106KHz. Ce circuit atteint une très faible consommation d'énergie grâce à un fonctionnement des transistors près du seuil (*Near-Threshold*) et l'utilisation de power gating à grain très fin pour les phases de veille. Par contre, les performances dans le mode actif en fonctionnement près du seuil sont considérablement diminuées (40KHz) et les mémoires utilisées sont très petites (64×10 bits de mémoire de programme et 52×40 bits de mémoire de données). Le jeu d'instructions est seulement sur 8 bits donc très peu adapté à beaucoup d'applications de l'IoT. En fait ce microcontrôleur a été optimisé pour l'application de la mesure de température.

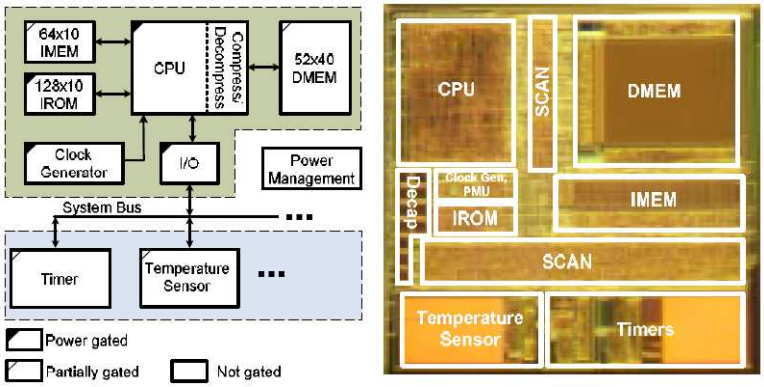


FIGURE 2.5 – Architecture du microcontrôleur Phoenix [136]

Le microcontrôleur [85] basé sur des accélérateurs utilise un partitionnement de l'architecture entre un *event processor* pour les tâches courantes du nœud de capteurs et un processeur général pour les tâches non régulières (figure 2.6). La technique de *power gating* à grain très fin est utilisée pour couper l'alimentation des différents accélérateurs et du processeur général dès que cela est possible et est géré par l'*event processor*. Cet *event processor* possède un ISA de seulement 8 instructions [86] pour pouvoir allumer et éteindre des blocs, faire des lectures et des écritures dans des registres ou des transferts de données par bloc à la manière d'une DMA [17] et enfin une instruction pour réveiller le processeur général. Cet *event processeur* est une sorte de DMA amélioré ayant la capacité de faire de la gestion de puissance. Les calculs nécessaires pour l'application sont exécutés sur des

17. Direct Memory Access

accélérateurs qui sont donc entièrement dédiés à une certaine application, ce qui laisse peu de flexibilité. Néanmoins, la consommation est de 0,44pJ par instruction à 0,55V d'alimentation pour 12,5MHz de fréquence de fonctionnement. Ici encore, le problème vient de la programmation d'un tel système. Si un compilateur n'est pas développé pour l'*event processeur* alors celui-ci ne va pas être utilisé par les programmeurs. De plus, l'ISA de l'*event processor* est trop limité pour s'adapter à différentes applications dans le cas où les accélérateurs ne peuvent pas gérer les calculs et algorithmes.

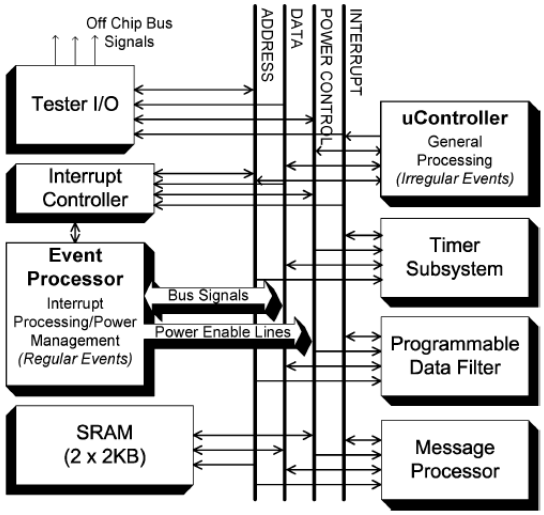


FIGURE 2.6 – Architecture du microcontrôleur avec un processeur à évènement pour les tâches courantes d'un nœud de capteurs et un processeur général pour les tâches non régulières [85]

Dans *SleepWalker* [50], dont l'architecture est montrée figure 2.7, beaucoup de techniques ont été mises en œuvre pour réduire la consommation du microcontrôleur. Ce circuit possède trois domaines de puissance (périphériques toujours allumés (AOP [18]), cœur de processeur, cache d'instructions (domaine ULV [19]) et les mémoires) et utilise trois types de technologies différentes pour être au point d'énergie minimum (MEP [20]) dans ces trois domaines de puissance en fonction de leur facteur d'activité. La fréquence cible pour le point d'énergie minimum f_{MEP} est de 25MHz, et le but est de mettre chaque domaine de puissance à la tension correspondante V_{MEP} pour le type de technologie utilisée. Ainsi, le type de technologie sélectionnée en 65nm pour le domaine ULV est du GP [21] SVT [22] pour une tension V_{MEP} de 0,4V. Les mémoires ainsi que le domaine AOP sont implémentés en LP [23] HVT [24], pour avoir une très faible puissance statique et une surface plus petite pour la RAM, et fonctionnent à une tension de 1-1,2V. Un cache d'instructions a été implémenté dans le domaine ULV afin de diminuer l'énergie de commutation dans le domaine de puissance des mémoires. De plus, un module d'adaptation de la tension (AVS [25]) du domaine ULV a été développé pour réguler la tension entre 0,32V et 0,48V afin de compenser les variations de délais dues aux variations de procédés et températures et ainsi s'adapter

18. Always On Peripherals
19. Ultra Low Voltage
20. Minimum Energy Point
21. General Purpose
22. Standard Voltage Threshold
23. Low Power
24. High Voltage Threshold
25. Adaptative Voltage Scaling

au point d'énergie minimum. Ce module régule aussi la fréquence de fonctionnement à 25MHz pour le domaine ULV. Ce microcontrôleur consomme 2,6pJ/cycle et $7\mu W/MHz$ en mode actif et $1,7\mu W$ en mode de veille avec une mémoire de programme de 16KB et une mémoire de donnée de 2KB. Néanmoins SleepWalker a été optimisé pour un point de tension et une fréquence pour être toujours à son point d'énergie minimum pour une certaine gamme d'application. Il sera donc moins bon en énergie sur les autres gammes d'applications. De plus le temps de réveil à partir du mode sleep ou le cœur est *power gaté* est très long puisqu'il lui faut 33 μs pour commencer à exécuter le code de l'application. Ceci est en grande partie dû au réveil du module AVS ($20\mu s$).

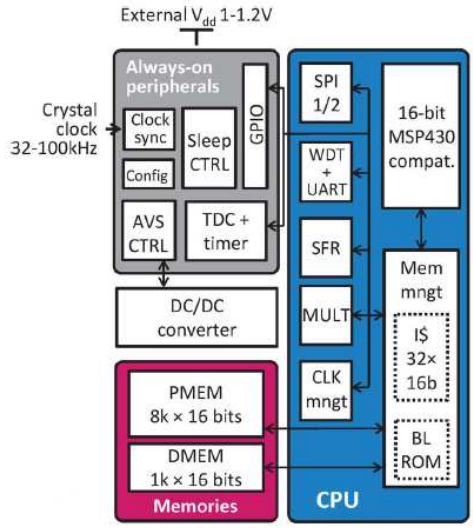


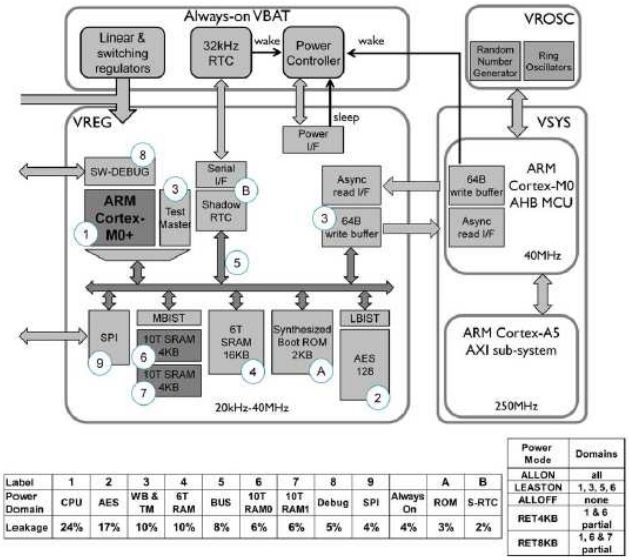
FIGURE 2.7 – Architecture du microcontrôleur *SleepWalker* [50]

Dans le papier de ARM à l'ISSCC 2015 [120], les auteurs ont développé un microcontrôleur en technologie 65nm avec quatre domaines de puissance principaux (figure 2.8) :

- VBAT : domaine *Always On* avec la RTC [26], le contrôleur de puissance et les régulateurs DC/DC et linéaire avec plusieurs types de régulateurs à capacités commutées et un LDO possèdant chacun leur plage de fonctionnement en terme de tension de sortie et charge maximale ;
- VREG : domaine avec processeur Cortex M0+, des périphériques de communication, des mémoires et un accélérateur avec 14 domaines de puissances dont leur alimentation peut être coupée à grain très fin avec beaucoup de modes de puissance (figure 2.8) ;
- VSYS : domaine avec un processeur Cortex M0 et un ARM Cortex A5 ;
- VROSC : domaine avec un accélérateur et un oscillateur en anneaux.

Le domaine VREG avec le cortex M0+ fonctionne sous le seuil avec un point d'énergie minimum à 0,35V. Dans son mode de consommation le plus bas (RET4KB dans figure 2.8), le microcontrôleur consomme 80nW en mode de veille à 250mV. En mode actif il peut atteindre 11,7pJ par cycle à 350mV pour une fréquence de 750KHz et 50pJ par cycle à 900mV pour une fréquence de 66MHz. Ce microcontrôleur utilise aussi différents types de mémoire, des SRAM 10T pour diminuer les fuites de courant pendant la phase de rétention de données et de la SRAM 6T dense pour les données temporaires qui est coupée pendant les phases de rétention. Néanmoins la fréquence de fonctionnement au point d'énergie minimale reste faible et n'est pas adapté à beaucoup d'application de l'IoT.

26. Real Time Counter



Label	1	2	3	4	5	6	7	8	9		A	B
Power Domain	CPU	AES	WB & TM	6T RAM	BUS	10T RAM0	10T RAM1	Debug	SPI	Always On	ROM	S-RTC
Leakage	24%	17%	10%	10%	8%	6%	6%	5%	4%	4%	3%	2%

Power Mode	Domains
ALLON	all
LEASTON	1, 3, 5, 6
ALLOFF	none
RET4KB	1 & 6 partial
RET8KB	1, 6 & 7 partial

FIGURE 2.8 – Architecture du microcontrôleur basé sur Cortex M0+ [120]

2.2.2 Système sur puce commerciaux

Énormément de microcontrôleurs existent dans le commerce pour les nœuds de capteurs communicants. Ici trois références vont être présentées de différent fournisseurs STMicroelectronics, Atmel, et Texas Instruments.

Le microcontrôleur STM32L0 [146], dont l'architecture est présentée figure 2.1, est basé sur le processeur ARM Cortex M0+ [30]. Ce microcontrôleur réussit à atteindre des consommations de $0,29\mu A$ dans son mode de consommation le plus bas avec seulement trois entrées de réveil et $65\mu s$ à 2,2ms de temps de réveil. La tension de la logique peut être programmée de 1,2 à 1,8V pour réduire la consommation dynamique et statique de la logique. Par contre, en modifiant la tension de fonctionnement, la fréquence maximum de fonctionnement est bornée. Par exemple pour une tension de la logique à 1,2V la fréquence maximum de fonctionnement n'est plus que de 4,2MHz. De plus, celui-ci est fonctionnel pour des températures comprise entre -40°C et 125°C, ce qui peut être intéressant pour certaines applications de l'IoT.

Le microcontrôleur d'Atmel SAMD21 [37] est aussi basé sur le processeur ARM Cortex M0+. Il atteint une consommation de $2,70\mu A$ dans son mode de consommation le plus bas pour une tension de la logique de 1,2V avec des temps de réveil de $19,6\mu s$. Ce microcontrôleur implémente un *Event System* permettant de faire communiquer les périphériques les uns avec les autres à travers des évènements, le tout sans utiliser de temps processeur ou de réveiller le processeur. Il peut être cadencé jusqu'à 48MHz pour une consommation dynamique de $70\mu A/MHz$. Ces consommations sont beaucoup plus élevé comparé aux réalisations de la littérature. Ceci est dû au technologies utilisées et aux différentes techniques de réduction de la consommations moins abouties dans les microcontrôleurs commerciaux.

Le système sur puce de Texas Instruments CC2650 [156] est très complet puisqu'il intègre trois processeurs, comme montré sur la figure 2.9 :

- ARM Cortex M3 : processeur principal qui gère l'application ;
- ARM Cortex M0 : processeur qui gère la couche MAC des protocoles Bluetooth LE et 802.15.4 de la radio intégrée ;
- *Sensor Controller* : processeur qui gère indépendamment les capteurs et autres

tâches courantes de l'application n'ayant pas d'intérêt à réveiller le processeur principal. Ce processeur est basé sur le jeu d'instruction du MSP430.

Le SoC intègre trois domaines de tension avec onze domaines de puissance, dont la plupart peuvent être coupés via des transistors de *power gating*. Le SoC consomme $1\mu A$ en mode de veille profonde. Le processeur M3 a une consommation dynamique de $61\mu A/MHz$ alors que le *Sensor Controller* ne consomme que $8,2\mu A/MHz$. De plus on peut voir que la mémoire disponible pour le *Sensor Controller* est très faible, seulement 2KB de SRAM.

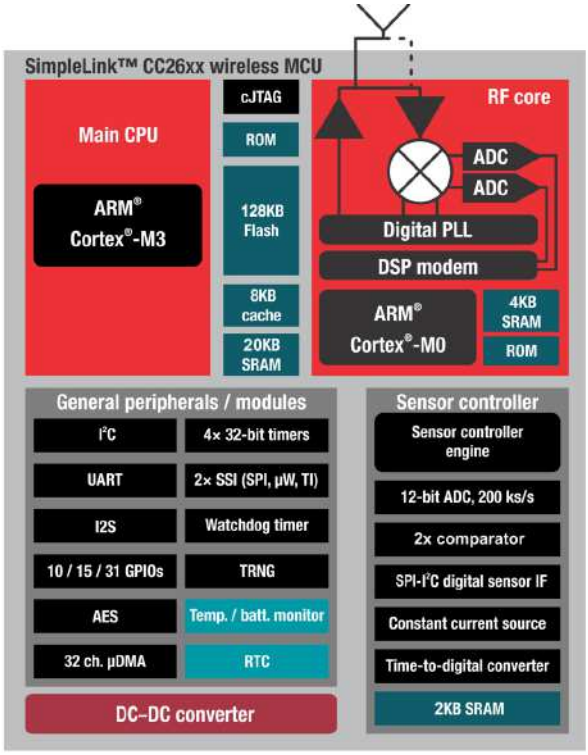


FIGURE 2.9 – Architecture du SoC CC2650 de TI [156]

Finalement, on voit que la consommation entre les microcontrôleurs de la recherche et de l'industrie est assez différente. Les consommations reportées dans la littérature sont nettement plus faibles que celles du commerce. Cela s'explique par la mise en place de techniques plus avancées comme le fonctionnement sous le seuil ou près du seuil, la mise en place de *power gating* à grain très fin, l'utilisation de technologies plus avancées, des mémoires très customisées, la variation de la tension de la logique ou encore l'utilisation de multiprocesseurs. Malgré tout, certaines techniques commencent à se démocratiser sur les derniers microcontrôleurs comme celui de TI [156] (multiprocesseur et *power gating* à grain très fin) ou ST [146] (possibilité de diminution de la tension de la logique). Les principales caractéristiques de tous ces microcontrôleurs sont résumées dans le tableau 2.2.

2.3 Pourquoi réduire la consommation du microcontrôleur et pour quelles applications ?

Aujourd'hui beaucoup d'applications de l'IoT requièrent des nœuds de capteurs autonomes (voir section 1.3 sur les applications biomédicales, dispositifs portatifs, environnement, agriculture, etc.). Cette autonomie est une contrainte forte pour augmenter la durée

de vie du système. Il faut absolument réduire la consommation des différentes entités de ces systèmes et, donc, celle du microcontrôleur grâce à toutes les techniques abordées dans ce chapitre et qui seront vues en détail dans le chapitre 3. Plus la consommation sera faible, plus les batteries à utiliser seront petites, et plus les nœuds seront intégrables dans l'environnement ou dans le vivant avec une très grande autonomie. Il existe aussi de nombreuses applications de l'IoT pour lesquelles les microcontrôleurs sont alimentés directement au réseau électrique. Pour ces applications aussi, la réduction de la consommation est primordiale car certes, à l'échelle d'un nœud, les milliampères de consommation ne représentent rien, mais en les multipliant par plusieurs milliards, cela peut faire atteindre des mégawatts de consommation qui pourraient être évités en appliquant les différentes techniques de réduction de la consommation. Ainsi, la réduction de la consommation du microcontrôleur d'un nœud de capteur a tout son intérêt, que ce soit pour des applications autonomes ou alimentées par réseau électrique.

2.4 Synthèse

Dans ce chapitre nous avons pu voir à la fois l'architecture d'un nœud de capteurs communicant avec la présentation pour chacune des entités du nœud des références de l'état de l'art et leurs consommations. Ce chapitre s'est ensuite concentré sur l'architecture des microcontrôleurs ainsi que des processeurs pour nœuds de capteurs communicants. Les ordres de grandeurs des consommations de chaque partie ont pu être présentés ainsi que les différentes problématiques et solutions pour réduire cette consommation afin d'avoir une vision système de la réduction de la consommation. Nous avons pu voir une multitude de microcontrôleurs créés pour les nœuds de capteurs que ce soit dans le domaine de la recherche [72][136][85][50][126][120] ou du commerce [146][37][156].

Différentes techniques de réduction de la consommation ont pu être abordées durant la présentation de ces réalisations mais seront détaillées dans le chapitre 3. Pour résumer, la réduction de la consommation peut se faire à différents niveaux.

- Architectural : ISA compact, instructions dédiées, profondeur de pipeline, accélérateurs dédiés, compression des données dans la mémoire, utilisation de cache d'instruction, multiprocesseur, taille du chemin de données, architecture basée sur évènement (event-driven), taille des mémoires, processeur type Harvard ou Von Neumann.
- Conception : utilisation de logique synchrone ou asynchrone.
- Gestion de l'énergie : mise en place de différents domaines de puissance avec utilisation de *power gating* et *clock gating* à grain très fin, fonctionnement sous le seuil ou près du seuil (*Sub-threshold* ou *Near-threshold*), fonctionnement au point d'énergie minimum (MEP), multiples modes de puissance, utilisation de différents types de régulateurs d'énergie pour avoir la meilleur efficacité dans différentes plages de fonctionnement (tension d'alimentation en sortie et charge demandée).
- Technologique : utilisation de technologies avancées pour réduire la surface, diminuer les tensions d'alimentation et augmenter les fréquences de fonctionnement, mélange pour une même technologie de transistors LP/GP avec différentes tensions de seuil (HVT/SVT/LVT), mémoires customisées pour les modes rétention et actif (SRAM 10T, 6T), choix des transistors de power gating, mémoires non volatiles émergentes
- Logiciel : compilation du programme avec optimisation pour réduire la taille du code, utilisation à bon escient de tous les modes de puissance.

Maintenant que les différentes réalisations et les différentes techniques de réduction de

la consommation ont été abordées, le prochain chapitre va détailler ces techniques. Ensuite la direction de la thèse et les choix en terme d'architecture, de *design*, de technologie et de gestion de l'énergie du système développé va être exposé. De plus, face à la multitude d'applications visées par un microcontrôleur, il est nécessaire que celui-ci soit suffisamment flexible pour s'adapter à n'importe quelle application, qu'il consomme le moins possible et qu'il soit facile à programmer. C'est pourquoi cette flexibilité va aussi diriger les choix architecturaux et technologique du système.

Microcontrôleur	Année	Chemin de donnée	Jeu d'instructions	Type	Mémoire - Qté	Tension	Fréq	Techno	Conso	Énergie	Ref
SNAP/LE	2004	16 bit	Snap ISA	asynchrone	2*4KB SRAM	0,6-1,8V	28-240MIPS	180nm	16nW-58nW à 0,6V (10 ev/s)	218pJ/ins à 1,8V 23pJ/ins à 0,6V	[72]
Phoenix	2008	8 bit	Compact ISA	synchrone	64*10bits IMEM 128*10 bits IROM 52*40 bits DMEM	0,5V	106KHz	180nm	29,6pW deep sleep	2,8 pJ/cycle actif	[136]
Accelerator Based Processor	2011	8 bit	z80 (CISC)	synchrone	2*2KB SRAM	0,55V	12,5MHz	130nm	$6\mu W$ deep sleep $48 - 78\mu W$ actif	678 pJ/tâche <=> 0,44 pJ/inst	[85]
SleepWlaker	2013	16 bit	MSP430 compatible	synchrone	16KB PMEM 2KB DMEM SRAM + I cache 32*16bit	1-1,2V (ext) 0,32-0,48V (int)	25MHz (Asservi)	65nm	$174\mu W$ actif $1,69\mu W$ deep sleep	2,6 pJ/cycle	[50]
ULSNAP	2014	16 bit	UlSnap ISA	asynchrone	4KB PMEM 4KB DMEM SRAM	0,95-1,2V	47-93MIPS	90nm	$4 - 9\mu W$ deep sleep	29-47 pJ/ operation	[126]
ArmISSCC	2015	32 bit	ArmV6-M	synchrone	8KB ULV SRAM 16KB SRAM	0,25-1,2V	29KHz - >66MHz	65nm	80nW deep sleep 850nW-6mW actif	11,7pJ/cycle à 0,35V	[120]
STM32L0	2014	32 bit	ArmV6-M	synchrone	64-192KB flash 8-20KB SRAM	1,65-3,6V ext 1,2-1,8V int	32KHz - 32MHz	-	6,65mA actif $0,29\mu A$ deep sleep	< 93 $\mu A/MHz$	[146]
SAMD21	2014	32 bit	ArmV6-M	synchrone	32-256KB flash 4-32KB SRAM	1,62-3,63V ext 1,2V int	32KHz - 48MHz	-	6,32mA actif $2,70\mu A$ deep sleep	< 70 $\mu A/MHz$	[37]
CC2650	2015	32 bit	ArmV7-M ArmV6-M MSP430	synchrone	128KB flash 20KB SRAM	1,8-3,8V ext	32KHz - 48MHz	-	2,94mA actif $1\mu A$ deep sleep	(M3) 61 $\mu A/MHz$ (SC) 8,2 $\mu A/MHz$	[156]

TABLE 2.2 – Tableau récapitulatif des spécifications et performances des microcontrôleurs pour nœuds de capteurs communicants

Chapitre 3

Perspectives de réduction de la consommation pour le sous système de traitement et de contrôle du nœud

Dans ce chapitre, les techniques de réduction de la consommation par solutions architecturales, par la gestion de l'énergie et par des solutions technologiques vont être explorées. Pour une meilleure compréhension de ces solutions, une présentation des circuits CMOS et des sources de consommation dans ce genre de circuit est faite en annexe A.6.

3.1 Augmenter la flexibilité d'un système et réduire sa consommation par des solutions architecturales

Au niveau du processeur l'une des premières spécifications à déterminer est le jeu d'instructions qui fixe la taille des instructions, les modes d'adressage et donc le type d'opérandes pour chaque instruction ainsi que la classe du jeu d'instructions. Cette classe détermine si les instructions vont directement agir sur une pile, la mémoire, des registres généraux ou les deux à la fois. L'avantage d'un jeu d'instructions RISC [1], comme celui de ARM [30], est qu'il a une taille d'instruction fixe (16 et/ou 32 bits), ce qui implique un décodage plus simple au niveau du matériel. De plus il y a beaucoup moins de modes d'adressage. Ainsi une opération arithmétique par exemple ne travaillera que sur des registres alors qu'un *load* ou *store* ne fera qu'un accès mémoire pour stocker la donnée dans un registre ou inversement. Ainsi, chaque type d'instruction peut être exécuté dans un module spécialisé différent, ce qui simplifie le contrôle et les unités d'exécution dans le processeur. Un exemple d'encodage d'instruction RISC est donné figure 3.1. La figure nous montre un exemple d'encodage d'une instruction qui possède comme opérandes sources Rs1 et Rs2 et comme opérande de destination Rd. Ces registres sont encodés avec leur numéro de registre et la taille de l'encodage dépend donc du nombre de registres. Ainsi plus il y a de registres généraux plus l'encodage du registre sera grand.

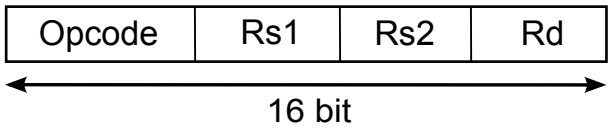


FIGURE 3.1 – Exemple d'un encodage d'instruction d'un jeu d'instructions de type RISC

1. Reduced Instruction Set Computer

Il existe deux types d'architecture entre des mémoires et un processeur figure (3.2). L'architecture de type Von Neumann possède une seule mémoire pour les instructions et les données, tandis que l'architecture de type Harvard possède deux mémoires distinctes pour les instructions et les données. L'architecture Harvard est plus performante que celle de Von Neumann puisqu'elle est capable de charger une instruction pendant qu'une autre instruction s'exécute en lisant ou écrivant dans la mémoire de données. Néanmoins l'architecture Harvard requière une structure plus complexe et donc nécessite plus de matériel.

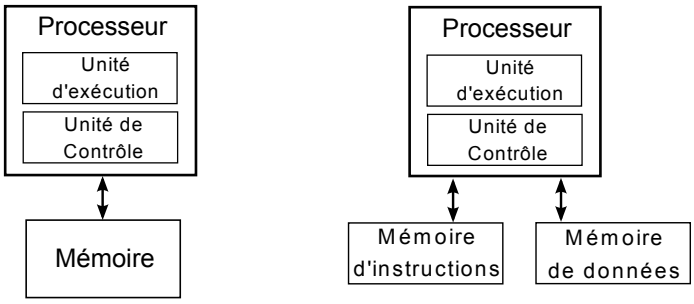


FIGURE 3.2 – Architectures de processeur de type Von Neumann (gauche) et Harvard (droite)

Pipeliner un processeur permet d'augmenter le débit d'instructions et la fréquence maximale du processeur puisque le chemin critique entre deux étages consécutifs est réduit. En effet, le pipeline permet d'avoir différentes tâches s'opérant simultanément dans les différents étages de pipeline. Par contre, il ne permet pas de diminuer le temps d'exécution d'une instruction. Un processeur pipeliné, contrairement à un processeur non pipeliné, devra avoir du matériel supplémentaire de contrôle pour gérer la dépendance des instructions et gérer le flush du pipeline en cas de mauvaises prédictions de branchements ou d'interruptions. Les aléas de structures comme l'accès à la mémoire dans l'étage de *Fetch* pour lire une instruction et d'écriture et de lecture à l'étage de mémorisation peuvent être résolus en séparant les mémoires d'instructions et de données. L'accès en lecture des registres généraux dans l'étage de lecture des opérandes et l'écriture dans les registres généraux à l'étage de *Write Back* peut être résolu en faisant les écritures sur les fronts montants et les lectures sur les fronts descendants de l'horloge. Les aléas de données de lecture après écriture, écriture après lecture et écriture après écriture peuvent être résolus en faisant du *forwarding* qui consiste à prendre le résultat de chacun des étages du pipeline et de le ramener à l'étage fonctionnel qui en a besoin à ce cycle. Une autre possibilité est de décrocher temporairement le pipeline, ou *stall* du pipeline, en bloquant temporairement le pipeline le temps que l'aléa soit résolu. L'ordonnancement des instructions peut prévenir les aléas de données qui peuvent arriver entre un chargement mémoire dans un registre et l'utilisation de cette donnée à l'instruction suivante. Les aléas de contrôle comme les branchements peuvent être résolus en ajoutant un module de prédiction de branchement pour éviter de faire un *stall* du pipeline. Ainsi le *pipelining* d'un processeur n'est pas trivial et demande beaucoup de matériel supplémentaire pour le mettre en œuvre, mais permet d'avoir des performances accrues.

La taille de la mémoire utilisée et son type doivent être bien dimensionnés en fonction de la taille des programmes finaux qui seront chargés pour l'application et en fonction de leur utilisation par le processeur. En effet, dans le cadre d'un microcontrôleur avec uniquement de la SRAM comme mémoire, la mémoire représente une partie importante de la consommation statique lorsque le microcontrôleur est en mode veille. C'est pourquoi on préfèrera des mémoires avec la consommation la plus faible pour la rétention du programme mais avec des temps d'accès plus importants et une densité moins élevée qu'une mémoire

de données plus rapide plus dense mais avec des courants de fuites plus importants qu'il faudra donc couper pendant le mode de veille.

Le cache d'instructions peut s'avérer très utile pour réduire la consommation d'énergie lors de la lecture d'une instruction dans la mémoire. Dans *SleepWalker* [50] la mémoire de programme est dans un domaine de puissance de 1V alors que la logique est dans un domaine de puissance de 0,4V. Un cache à 32 entrées de 16 bits est alors implémenté et permet d'avoir une réduction de la consommation du système de 21% à 52%. Grâce à ce cache, moins d'accès sont fait à la mémoire de programme mais le surplus en surface dans le domaine de puissance de 0,4V est de 33%.

Á un niveau hiérarchique plus élevé, la réduction de la consommation peut être obtenue en utilisant plusieurs processeurs dans un nœud de capteurs comme montré dans la figure 3.3 et utilisé récemment dans le système CC2650 de TI [156]. Un processeur très basse consommation (processeur LP) pour les tâches courantes du nœud et un processeur généraliste (processeur HP) avec une moins bonne efficacité énergétique mais une fréquence de fonctionnement plus élevée ainsi que des capacités de calcul très supérieures pour les tâches irrégulières et les calculs complexes. De cette manière le processeur principal est coupé pendant les longues phases de veille. Le processeur basse consommation peut lui aussi être coupé mais va se réveiller régulièrement pour faire des mesures sur les capteurs, gérer le protocole radio pour la réception ou la transmission de données. Il réveillera le processeur principal si des calculs complexes sont à faire sur les données mesurées ou si un besoin applicatif irrégulier doit être exécuté. Cette solution architecturale augmente fortement la flexibilité du système et permet de s'adapter à différents types d'applications en exploitant la même architecture.

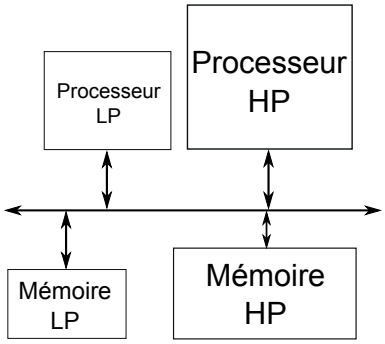


FIGURE 3.3 – Système Multi processeur avec un processeur basse consommation (Processeur LP) et un processeur haute performance (Processeur HP)

Les accélérateurs matériels et instructions dédiées sont très utilisés afin de déporter certaines parties du code exécutées régulièrement par du matériel dédié via un module externe au processeur, comme l'AES, le CRC ou le RNG dans le cas du STM32L0 [146] ou via des instructions DSP dédiées comme la MAC [2] ou le SIMD [3], très utilisés pour faire du filtrage ou une FFT [4]. Ces solutions améliorent considérablement l'efficacité énergétique du système et sont même développées aujourd'hui dans les processeurs généralistes comme celui du ARM Cortex-M4 avec le jeu d'instruction ARMV7-M [31].

Afin de rendre le système plus flexible et facile à utiliser pour les programmeurs, les microcontrôleurs possèdent des unités de débogage sur puce [5]. Il existe deux modes de

2. Multiply-Accumulate
3. Single Instruction Multiple Data
4. Fast Fourier Transform
5. Debug On Chip

débogage, le *Foreground Debug Mode* (FGDM) pour le contrôle complet du processeur (stopper, relâcher le cœur de processeur et accéder aux informations internes) et le *Background Debug Mode* (BGDM) pour visualiser l'activité du processeur en arrière-plan alors que le code s'exécute normalement. Aujourd'hui, au niveau industriel, le débogage s'effectue via le port JTAG [6] ou le port SWD [7]. Dans [105], les auteurs ont développé un module de débogage ICE [8] pour un accélérateur de code java implémenté en asynchrone. Il est capable de faire du FGDM et BGDM et possède une unité de détection des *breakpoints* [9]. Chez ARM le débogage s'effectue à l'aide de plusieurs IPs regroupées sous le nom de CoreSight. La figure 3.4 montre les deux IPs principales pour faire du FGDM et du BGDM dans les microcontrôleurs avec une architecture ARM. Le DAP [10] permet de faire du FGDM en ayant accès aux cœurs et à tout l'espace mémoire et l'ETM [11] permet de fournir la trace en temps réel des instructions exécutées et des données pour faire du BGDM.

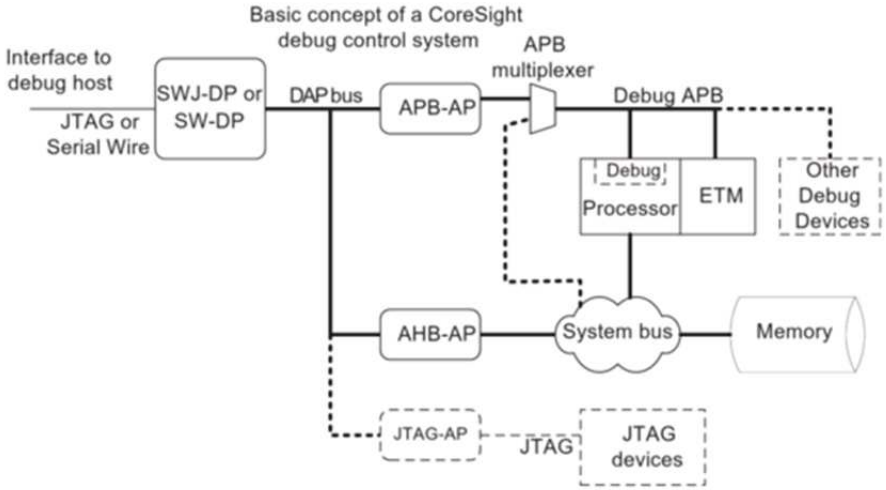


FIGURE 3.4 – IPs de ARM pour implémenter du Debug On Chip et obtenir une trace d'exécution [172]

Le debogage et le test sont très importants dans le domaine de l'industrie et il est impossible aujourd'hui d'avoir un microcontrôleur sans une unité de *debug*. C'est pourquoi il faut penser l'architecture du système et du processeur directement en y insérant une solution de *debug* et faire toujours attention au compromis consommation/service.

3.2 Réduire la consommation d'un système par une gestion intelligente de l'énergie

Les nœuds de capteurs communicants fonctionnent la plupart du temps en rapport cyclique [12]. C'est à dire qu'ils vont fonctionner sur une courte période d'activité et vont être mis dans le mode de consommation le plus bas le reste du temps. Le diagramme en

6. Join Test Action Group
7. Serial Wire Debug
8. In Circuit Emulator
9. Instruction d'arrêt d'exécution
10. Debug Access Port
11. Embeded Trace Macrocell
12. Duty Cycling

fonction du temps de la puissance consommée par une application pour nœud de capteurs est montré figure 3.5.

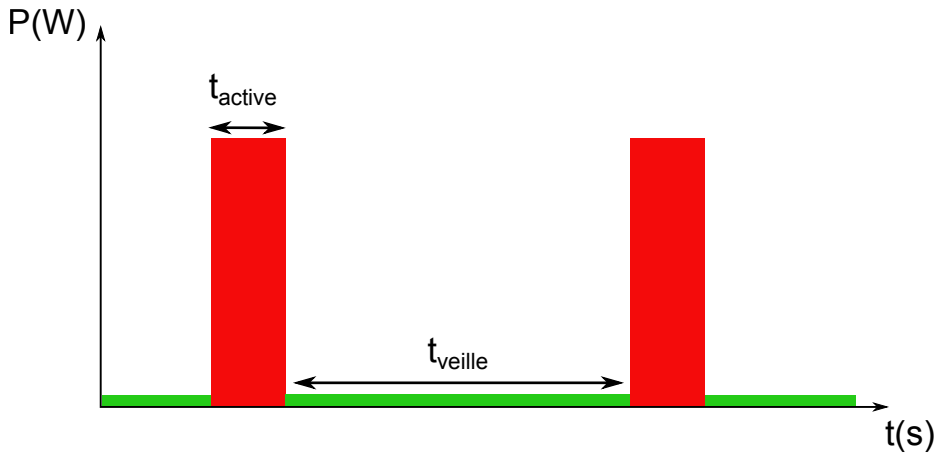
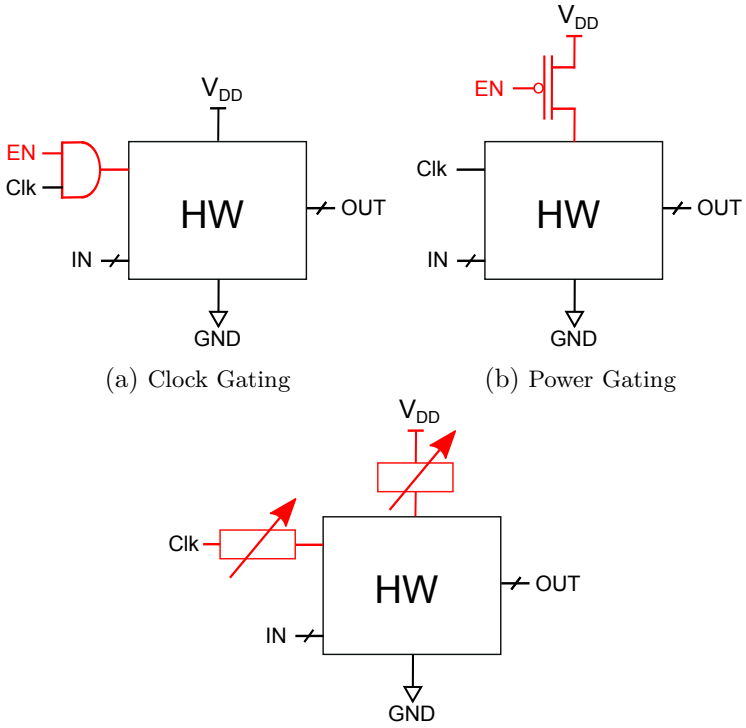


FIGURE 3.5 – Technique de duty cycling pour un nœud de capteurs communicant

Il existe différentes techniques de mise en mode de veille ou en mode dégradé. Il s'agit donc de cibler la bonne technique en fonction des contraintes et du besoin applicatif. Tout d'abord, il est possible de désactiver l'horloge d'un bloc ou *clock gating* comme schématisée dans la figure 3.6a. Ceci permet à la fois d'inhiber les cellules séquentielles mémorisant les données et de supprimer l'activité interne α pour annuler la composante dynamique de la consommation de puissance moyenne. En revanche, la composante statique induite par les fuites de courant est toujours présente. Pour supprimer cette composante statique il est alors possible de couper localement l'alimentation d'un bloc (*power gating*) comme présenté figure 3.6b. L'énergie statique du bloc est alors ramenée à zéro, sauf au niveau du transistor de coupure qui possède toujours une fuite de courant. Une autre technique consiste à dégrader les performances et la consommation en jouant dynamiquement avec la fréquence et la tension du bloc (DVFS [13]) comme présenté figure 3.6c. Un asservissement de ces deux paramètres en fonction de l'activité demandée à ce bloc est alors effectué. Le *clock gating* est une opération logique qui ne nécessite pas de temps d'établissement contrairement au *power gating* qui a besoin d'un certain temps pour re-stabiliser la tension après une coupure. De plus, avant une coupure de l'alimentation il est quelquefois nécessaire de sauvegarder le contexte (valeurs des registres ou données en mémoire) ce qui implique une latence et de l'énergie supplémentaire pour faire cette opération. Le choix d'une technique par rapport à une autre est donc déterminé par la granularité des unités à éteindre et du temps de réaction souhaité pour le système. Aujourd'hui, de plus en plus de microcontrôleurs sont dévellopés avec une granularité de *power domain* très fine [120], ce qui permet de couper de petits blocs pour agir sur la consommation statique très finement.

Une autre technique de réduction de la consommation d'un circuit consiste à l'alimenter à son point d'énergie minimum qui se situe dans la gamme ULV [14] du transistor comme montré figure 3.7. Ce point d'énergie minimum (MEP [15]) est atteint lorsque la somme de l'énergie dynamique et statique est minimale. En dessous de ce point, l'énergie statique devient plus importante que l'énergie dynamique. Par contre, comme montré dans la figure 3.7, à cette tension V_{MEP} la fréquence du transistor peut être divisée par 25. Alors qu'en

13. Dynamic Voltage and Frequency Scaling
14. Ultra Low Voltage
15. Minimum Energy Point



(a) Clock Gating (b) Power Gating

(c) DVFS

FIGURE 3.6 – Techniques de réduction de la consommation d'un bloc matériel

alimentant le circuit dans la gamme NTC [16], la performance est divisée uniquement par 5 et l'énergie par 4. Mais plusieurs problèmes sont engendrés par une alimentation du circuit en *Near Threshold Voltage* (NTV) ou ULV. Plus la tension d'alimentation et la tension de seuil diminuent, plus le courant *Ion* va être sensible aux variations de la tension de seuil ΔV_{TH}. Or, la variation de V_{TH} est inversement proportionnel au carré de la surface de la grille. Ainsi il faut augmenter la taille de la longueur de grille pour diminuer les variations du V_{TH} et être moins sensible aux fluctuations aléatoires de dopant (RDP [17]). Les autres problèmes qui peuvent être rencontrés en NTV et ULV sont les problèmes de maintien (*hold*) des éléments de mémorisation tel que les *latches* et *flip-flops* ainsi que des problèmes de décalages dans l'arbre d'horloge (*clock skew*). Avec de plus en plus de *power domain* dans les circuits très basse consommation, l'utilisation de *level shifters* entre les différents domaines est nécessaire pour adapter la tension des signaux du passage d'un *power domain* à l'autre. Malheureusement ces *level shifters* sont très sensibles aux variations PVT [18].

La plupart de ces problèmes sont rencontrés uniquement avec les circuits synchrones. La logique asynchrone étant très robuste aux variations de tension, elle a toute sa place pour les circuits fonctionnant en NTV et ULV. Les avantages de la logique asynchrone seront discutés section 3.4 mais seront présentés dans le détail au chapitre 6.1.

16. Near Threshold Computing
17. Random Dopant Fluctuations
18. Process Voltage Temperature

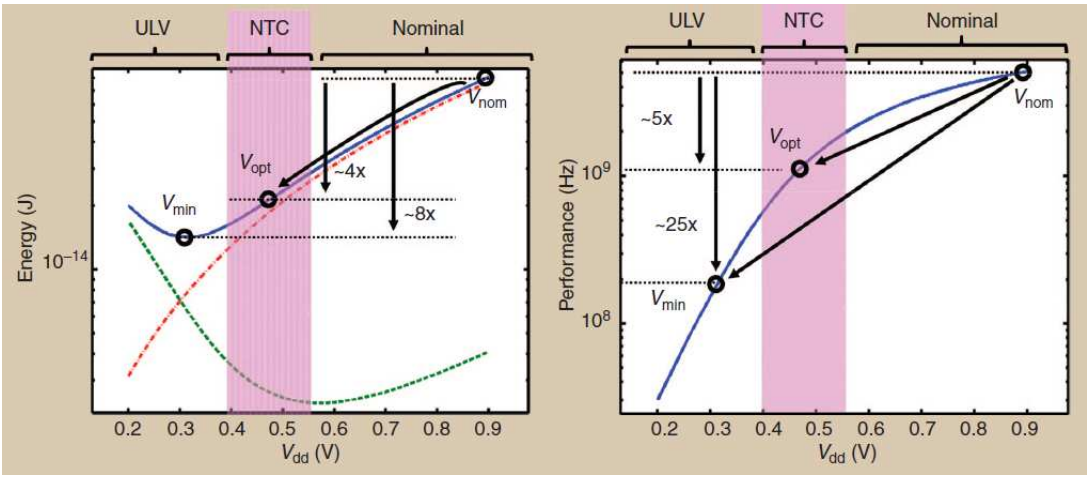


FIGURE 3.7 – Énergie et performance en fonction de la tension d'alimentation pour les trois gammes d'opération (nominale, ULV, NTV) [129]

3.3 Augmenter la flexibilité d'un système et réduire sa consommation par la technologie

Les différentes réalisations qui ont pu être présentées utilisent d'anciennes technologies pour une longueure de grille de moins de 65nm. Dans cette thèse, l'un des objectifs est d'utiliser une technologie avancée telle que la technologie de STMicroelectronics UTBB FDSOI [19]. Cette technologie fournie une large gamme de customisation entre vitesse et faible fuite pour une flexibilité maximale. En effet cette technologie a un excellent contrôle électrostatique du canal, de faibles fuites de courant et une immunité à la fluctuation aléatoire des dopants. De plus, l'utilisation de la polarisation arrière (BB [20]) permet dynamiquement d'augmenter les performances en vitesse en effectuant du FBB [21] ou de diminuer les fuites de courants en effectuant du RBB [22]. Les caractéristiques de la technologie UTBB FDSOI sont expliquées dans le papier [41] ainsi que dans l'article de journal [48]. Un résumé en est fait dans cette section.

3.3.1 Polarisation arrière de la technologie UTBB FDSOI

La technologie que nous allons utiliser dans cette thèse est l'UTBB FDSOI de STMicroelectronics en 28nm dont la vue en coupe d'un transistor NMOS et PMOS est présentée figure 3.8a. Contrairement à la technologie *Bulk*, le canal est créé dans un film très fin de silicium non dopé et isolé de la face arrière par un oxyde enterré.

La face arrière qui est soit un N-Well ou P-Well est implantée sous l'oxyde enterré afin d'améliorer l'effet de canal court et d'ajuster la tension de seuil. Une large gamme de tension de polarisation arrière est possible de -1,8V à +1,8V pour ajuster la tension de seuil du transistor et modifier les courants de fuites. La figure 3.8b montre pour un transistor NMOS la possibilité de faire du *Reverse Back Biasing* pour la gamme RVT [23] et de faire du *Forward Back Biasing* pour la gamme LVT [24].

19. Ultra Thin Body and Box Fully Depleted Silicon On Insulator
20. Back Biasing
21. Forward Back Biasing
22. Reverse Back Biasing
23. Regular V_{TH}
24. Low V_{TH}

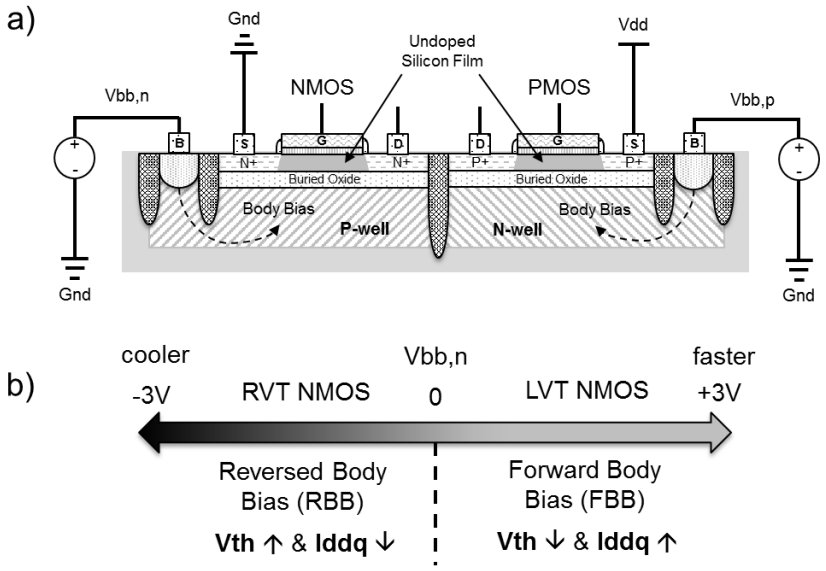


FIGURE 3.8 – Technologie UTBB FDSOI : (a) vue en coupe d'un transistor NMOS et PMOS avec des Well en configuration conventionnelle (b) Caractéristique de la polarisation arrière d'un NMOS

La figure 3.9a illustre les résultats de boost en fréquence de la technologie UTBB FDSOI 28nm comparés à la technologie Bulk pour différentes tension d'alimentation alors que la figure 3.9b illustre le fait de pouvoir réduire les fuites de courants en appliquant du RBB.

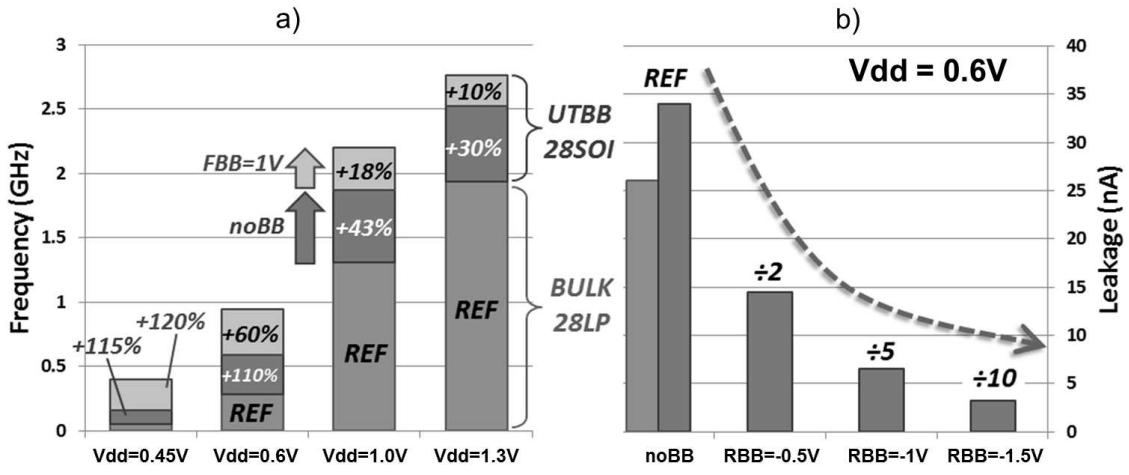


FIGURE 3.9 – Performances de la technologie UTBB FDSOI extraite des simulations électrique du chemin critique d'un ARM64 (a) boost en performance de la technologie LVT avec du FBB (b) réduction des fuites de la technologie RVT avec du RBB [42]

Ce *back biasing* va permettre, pour les nœuds de capteurs communicants, de booster les performances pendant les périodes d'activité et de réduire les courants de fuites pendant les phases de veille.

3.3.2 La technologie UTBB FDSOI près du seuil

Le point d'énergie minimum de la technologie UTBB FDSOI 28nm se situe entre 0,2V et 0,4V, comme montré figure 3.10. Celui-ci ne diffère pas trop entre un transistor RVT et LVT. De plus il est possible de voir que les dispositifs en LVT ont une plus grande énergie statique que leur équivalent RVT et que cette différence augmente en appliquant de la polarisation arrière.

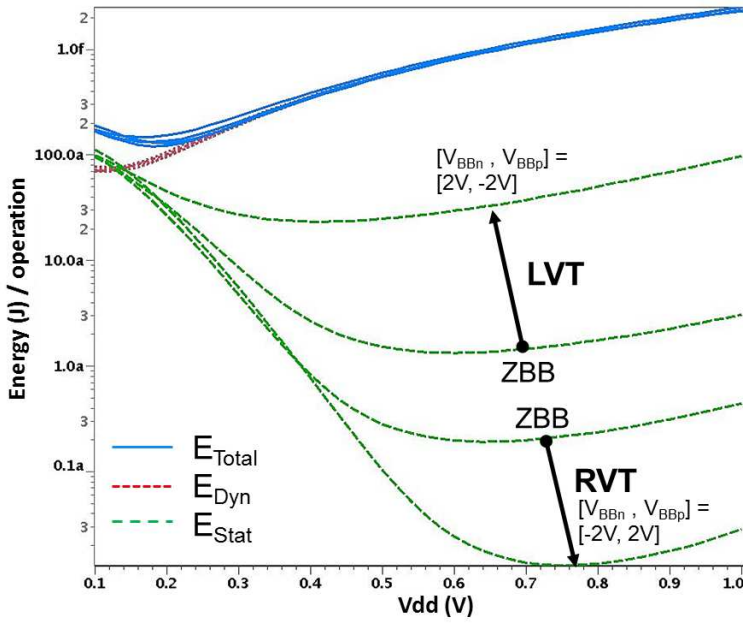


FIGURE 3.10 – Analyse du point d'énergie minimum (MEP) pour la technologie UTBB FDSOI RVT et LVT pour un oscillateur en anneau en fonction de la tension. ZBB = pas de polarisation

La figure 3.11 montre une comparaison des délais et de l'énergie au MEP pour un additionneur 8 bits asynchrone dans trois technologies différentes : l'UTBB FDSOI LVT/RVT, le Bulk 28nm et le FinFET 14nm. L'UTBB FDSOI RVT expose les meilleurs chiffres en fuites de courants mais par contre de faibles chiffres en fréquence. Quoi qu'il en soit, le Bulk 28nm possède les moins bonnes performances en regard de la technologie FinFET et FDSOI.

3.3.3 Élargissement de la largeur de grille de la technologie FDSOI

Une option de la technologie FDSOI 28nm est le *Poly Biasing* (PB). Cette technique consiste à agrandir la longueur de grille pour avoir un meilleur compromis délais/énergie. Les simulations de la figure 3.12 montre la fréquence au MEP en fonction de l'énergie d'un oscillateur en anneau implémenté de PB0 à PB16 (16nm d'agrandissement de longueur de grille). Entre du PB0 et du PB16 on peut observer une réduction de l'énergie par opération de 15% tout en restant à la même fréquence de 500MHz. Il est possible de voir aussi que le LVT PB16 peut consommer une énergie plus faible que le RVT. Cela permet de mixer différent PB tout en restant sur des transistors LVT et avoir des compromis délai/énergie.

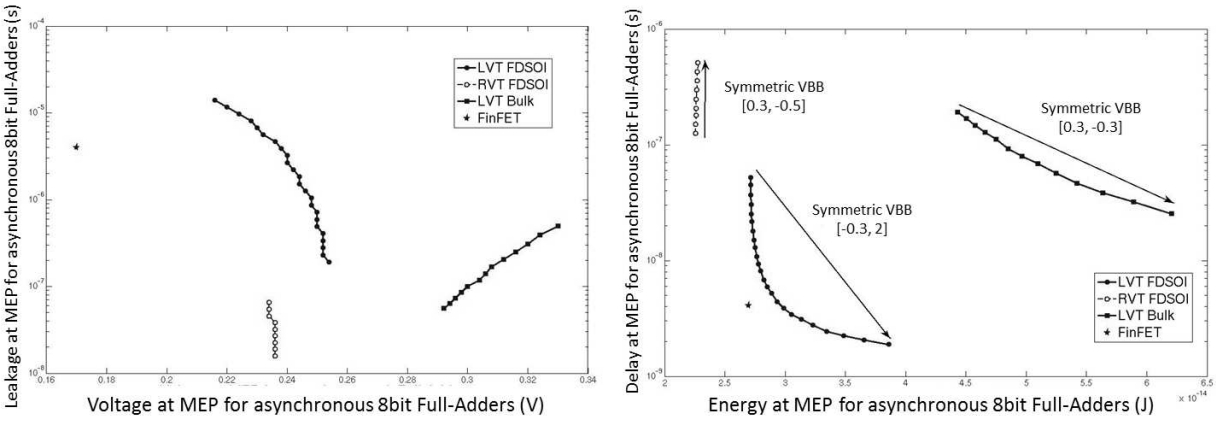


FIGURE 3.11 – Énergie et délais au MEP : comparaison avec les technologies Bulk et FinFET

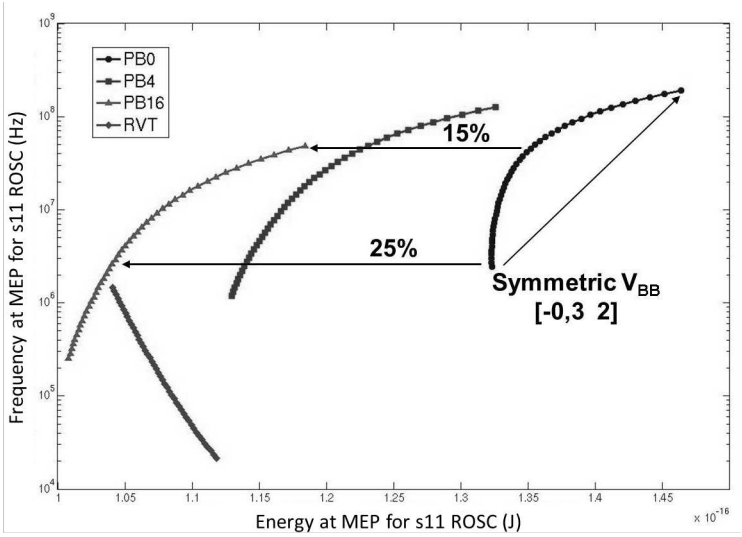


FIGURE 3.12 – Énergie au MEP en utilisant les options d'élargissement de largeur de grille de la technologie FDSOI

3.3.4 Co-intégration des différents types de technologie FDSOI

Il est très difficile de co-intégrer des cellules RVT et LVT dues aux contraintes d'isolation des *Well*. C'est pourquoi co-intégrer du LVT en différents *Poly Biasing* et LVT devient très intéressant pour le domaine de l'IoT. Comme montré dans la figure 3.13 qui représente l'énergie par opération en fonction de la tension d'alimentation, les cellules PB16 paraissent mieux s'adapter aux circuits dont l'alimentation est dans le domaine ULV/NTV alors que le RVT permet d'être meilleur en énergie après 0,5V par rapport au PB16 LVT.

3.4 Avantages de la logique asynchrone

Dans cette section, une courte présentation des avantages de la logique asynchrone va être faite et sera développée plus particulièrement dans le chapitre 6.1. La logique asynchrone est basée sur des mécanismes de synchronisation locale entre les différents opérateurs asynchrones. De ce fait, aucune horloge globale n'est nécessaire et la consommation

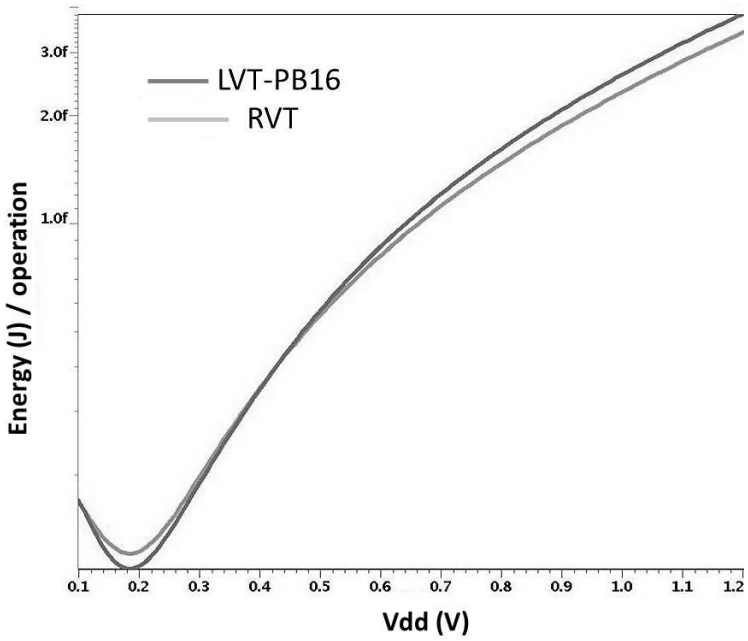


FIGURE 3.13 – Résultats en énergie d'un oscillateur en anneau sur une gamme de tension complète pour la technologie LVT-PB16 et RVT

est lissée et évènementielle. Cela a pour conséquence que le système est automatiquement en mode veille si aucun évènement n'est à traiter. La consommation des opérateurs de traitement est alors lissée car l'évènement va se propager au sein du système au fur et à mesure. Les pics de courant qui apparaissent dans les circuits synchrones auront ici un profil très lissé ce qui permet une réduction des émissions électromagnétiques. Les auteurs du papier [57] ont implémenté et fabriqué deux microcontrôleurs pour l'IoT dont un avec un cœur de processeur 8051 synchrone et un autre avec un cœur de processeur 8051 asynchrone en logique QDI[25]. Les deux ont été développés pour fonctionner sur une large gamme de tension V_{DD} à une tension sous le seuil. Il en ressort qu'à une tension nominale et sur la large gamme de tension, les performances des deux cœurs de microcontrôleurs sont comparables. Mais dans des conditions de larges variations de PVT et de charges de travail, la version asynchrone permet d'avoir une plus faible dissipation d'énergie que la version synchrone. Ainsi, pour les applications de l'IoT qui impliquent une large variation du PVT et de l'activité, une implémentation asynchrone des cœurs de processeur est mieux adaptée alors qu'une version synchrone sera préférable si ces variations sont moins sévères. Ainsi, l'utilisation de la logique asynchrone est largement justifiée pour le domaine de l'IoT même si la surface d'un système dans cette logique est multipliée par deux.

3.5 Synthèse

Cette section a permis au lecteur de comprendre le fonctionnement de la technologie à la base des circuits intégrés, de connaître les sources de consommation dans les circuits numériques et de se rendre compte de toutes les possibilités existantes pour réduire la consommation d'un SoC. Cette réduction de la consommation peut se faire à un niveau architectural/logique, technologique ou via une gestion intelligente de l'énergie dans le SoC.

25. Quasi Delay Insensitive

Le but étant de trouver la parfaite combinaison de toutes ces possibilités pour atteindre la consommation la plus basse et les performances en adéquation avec les besoins applicatifs. Dans certains cas, ces besoins applicatifs ne sont pas très bien définis étant donnée la multitude d'applications qui existe dans le domaine de l'IoT (section 1.3). C'est pourquoi un besoin en flexibilité de ces systèmes devient indispensable pour pouvoir s'adapter à n'importe quel type d'application. Cette flexibilité peut être faite au niveau architectural avec du multiproceseur, au niveau technologique en utilisant les différentes possibilités de l'UTBB FDSOI comme le *Back Biasing* et le *Poly Biasing*, au niveau gestion de l'énergie en mettant en place de multiples domaines de puissances et en alimentant le circuit en *Near Threshold Voltage*, et au niveau du type de logique utilisé en implémentant le système avec de la logique asynchrone.

Les objectifs de la thèse ont donc pu être fixés en fonction de cet état de l'art et il a été décidé de développer un système avec les caractéristiques suivantes.

- Un système multiprocesseur avec un mini processeur s'occupant des tâches courantes dans un nœud de capteurs et un processeur principal ou DSP s'occupant des tâches irrégulières.
- Ce mini processeur doit avoir un temps de réveil très rapide et être le plus simple possible pour réaliser les tâches courantes du nœud de capteurs et atteindre une très faible consommation.
- Le mini processeur sera implémenté en logique asynchrone qui fonctionne naturellement sur évènements et est donc très adapté à ce qu'il doit faire puisque les tâches courantes dans un nœud de capteurs sont des tâches évènementielles.
- Le tout sera conçu dans la technologie UTBB FDSOI 28nm et exploitera toutes les possibilités de cette technologie.

A titre de comparaison avec l'état de l'art, le tableau 3.1 récapitule les différences entre le processeur de réveil développé avec quelques références.

Caractéristiques	**Ce travail**	**SNAP** [72]	**SleepWalker** [50]	**ARM ISSCC** [120]	**TI CC2650** [156]
Technologie	28nm	180nm	65nm	65nm	
Logique	asynchrone	asynchrone	synchrone	synchrone	synchrone
Multiprocesseur	X	-	-	X	X
Debugable	X	-	X	X	X
Temps de réveil	ultra rapide	ultra rapide	lent	lent	lent
Facile à programmer	X	-	X	X	X
Facile à intégrer dans le flot de conception	X	-	X	X	X

Table 3.1 – Comparaison des caractéristiques du processeur de réveil avec certains microcontrôleurs ultra basse consommation de l'état de l'art

Mais, avant de se lancer dans l'implémentation d'un système tel que celui-ci, un modèle de simulation de la consommation a été mis en place afin d'évaluer les gains potentiels en énergie d'un tel système.

Chapitre 4

Modélisation de la consommation d'un nœud de capteurs et de son microcontrôleur

L'objectif du chapitre de ce chapitre est de présenter la simulation d'un nœud de capteurs entier dans différents scénarios, l'architecture du microcontrôleur avec et sans processeur de réveil avec tous les modes de consommation du microcontrôleur et, enfin, la simulation de la consommation d'un microcontrôleur avec et sans processeur de réveil dans différents scénarios applicatifs.

4.1 Modèle de consommation d'un nœud de capteurs

Dans un premier temps des simulations de la consommation d'un nœud de capteurs communicants ont été mises en place pour évaluer la part de consommation de chacune des entités du nœud dans des scénarios réels de nœuds de capteurs communicants. Ensuite des simulations du microcontrôleur seul ont été faites pour évaluer les gains possibles avec le processeur de réveil. Pour l'estimation de la consommation, le logiciel de Docea Power, Aceplorer a été utilisé. Cet outil permet de définir l'architecture du circuit, les différents blocs internes en termes de consommation dynamique, statique, surface, les différents domaines de puissance et d'horloge, les régulateurs d'énergie ainsi que leur efficacité. Le tout pouvant être paramétré, il est ensuite possible de jouer avec ces paramètres dans différents scénarios pour évaluer les consommations en termes de puissances moyennes et instantanées consommées. Plusieurs modes de consommation pour chacun des blocs peuvent être définis. Les scénarios permettent alors de définir des phases dans lesquelles les modes de consommation de chacun des blocs seront fixés. Il est ensuite possible de récupérer les données de puissances consommées instantanées et moyennes.

4.1.1 Phases applicatives des nœuds de capteurs

Pour évaluer les consommations d'un nœud de capteurs nous allons utiliser un flot d'exécution conventionnel comme présenté dans [103] et illustré figure 4.1.

La figure 4.1 présente la décomposition en trois phases applicatives d'un nœud de capteur sans fil : phase de mesure, phase de calcul et phase de veille. Cette décomposition est valide pour de très bas rapports cycliques jusqu'à des rapports cycliques élevés. Le rapport cyclique est défini par le rapport de la durée d'un phénomène sur la période de répétition du phénomène $t_{période}$. Pendant la phase de veille de durée t_{veille} la plupart des modules ont l'horloge coupée ou encore l'alimentation coupée. Seul les modules réveillant le système ou les modules ayant besoin d'une rétention des données sont alimentés et/ou

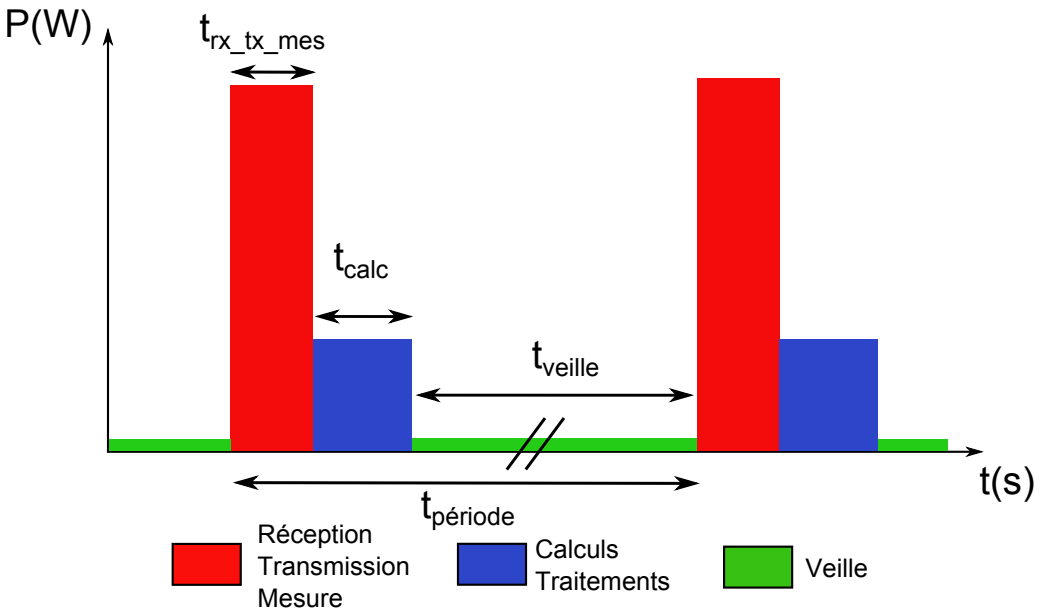


FIGURE 4.1 – Décomposition des différentes phases applicatives d'un nœud de capteurs sans fil

ont une horloge active. Pendant la phase de réception/transmission/mesure, des mesures peuvent être faites sur les capteurs, une réception ou une transmission peut avoir lieu à travers le lien radio. La durée $t_{rx\_tx\_mes}$ de cette phase dépend de la précision de conversion ou du protocole radio requis pour l'application. Pendant la phase de calcul ou de traitement des données, tous les sous-module du microcontrôleur sont alimentés et fonctionnent à la fréquence idéale pour l'application.

4.1.2 Architecture du nœud modélisé

L'architecture du nœud est présenté figure 4.2. Il est composé d'une radio du commerce qui était très utilisée dans le domaine des nœuds de capteurs, la radio CC2420 de TI [150], un capteur de pression de Bosch BMP180 [51], un microcontrôleur ainsi que certains composants comme la FLL [1], l'oscillateur ou le LDO [2]. Une partie des composantes minimales d'un microcontrôleur pour l'application est modélisée avec les chiffres en consommation de la technologie UTBB FDSOI 28nm dont la mise en équation sera présentée dans la prochaine section 4.1.3. Les chiffres en consommation de la FLL et de l'oscillateur sont extraits de la datasheet du microcontroleur d'Atmel [35]. Le LDO est modélisé à l'aide des équations de la section 4.1.3.

4.1.3 Mises en équation du modèle de consommation

Nous souhaitons implémenter le microcontrôleur en technologie FDSOI 28 nm, c'est pourquoi un modèle de consommation dynamique et statique a été extrait de la technologie FDSOI 28 nm de STMicroelectronics pour l'implémenter dans le modèle. Un modèle a été extrait pour la logique et pour les mémoires. Comme il peut être vu dans l'annexe A.6, la consommation d'un circuit peut se modéliser avec l'équation 4.1 possédant une composante statique et une composante dynamique.

1. Frequency Locked Loop
2. Low Dropout regulator

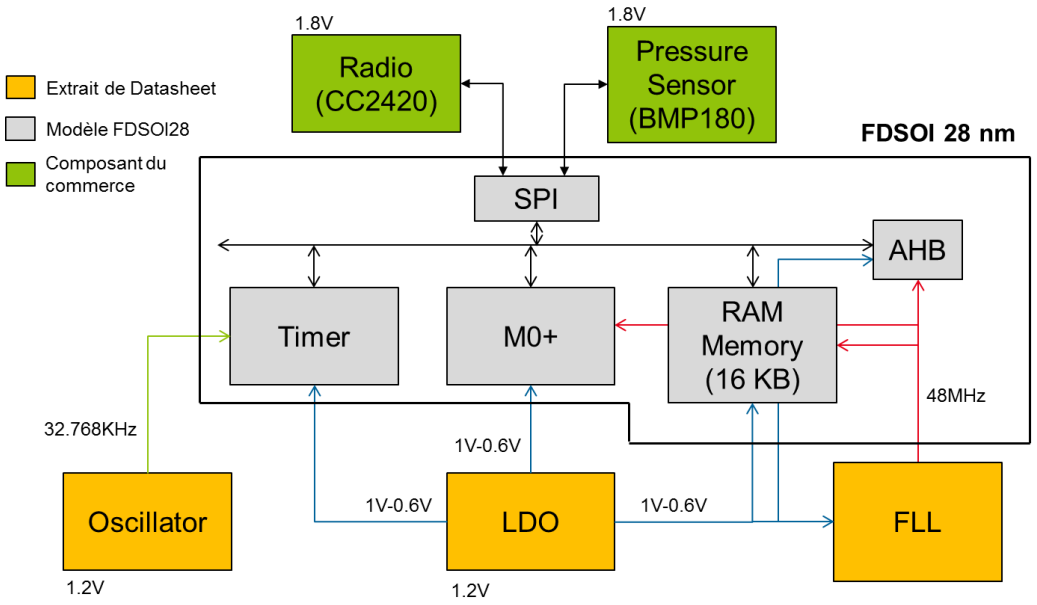


FIGURE 4.2 – Architecture du nœud utilisé pour les simulations dans les différentes phases applicatives

$$P_{moy} = \alpha \frac{1}{2} \Delta_t I_{SC_{max}} V_{DD} f + \alpha C_L V_{DD}^2 f + V_{DD} I_L \tag{4.1}$$

Avec α le facteur d'activité qui modélise la probabilité de changement d'état du nœud de sortie, V_{DD} la tension d'alimentation, $I_{SC_{max}}$ le courant de court-circuit maximal, f la fréquence de fonctionnement du circuit, Δ_t le temps de court-circuit, C_L la capacité de grille des transistors ramené à une valeur équivalente et I_L le courant de fuite duc circuit.

Pour la modélisation de modules en technologie FDSOI 28 nm, des valeurs de courant dynamique et de courant statique ont été extraites et se modélisent par les coefficients $G_p(V_{DD_i})$ et $G_L(V_{DD_i})$ dont les unités s'expriment en A/Kgate/MHz et A/Kgate respectivement. Ce coefficient dépend de la tension d'alimentation, du type de technologie (RVT/LVT) et du *Back Biasing*. Ainsi l'équation de la puissance moyenne consommée par un module s'exprime de la manière suivante :

$$P_{moy_{mod}} = \frac{\sum_{i=1}^{S} (V_{DD_i} \cdot G_p(V_{DD_i}) \cdot n \cdot a_i \cdot f_{clk_i} + V_{DD_i} \cdot n \cdot G_L(V_{DD_i})) \cdot \Delta_{t_i}}{\Delta_{t_p}} \tag{4.2}$$

Avec V_{DD_i} la tension d'alimentation dans la phase i, n le nombre de Kgate, a_i l'activité du module dans la phase i, f_{clk_i} la fréquence du module dans la phase i, Δ_{t_i} la durée de la phase i et Δ_{t_p} la période du scénario.

L'équation 4.2 effectue la somme des puissances dynamiques et statiques d'un module multipliée par le temps de la phase, ce qui donne l'énergie consommée pour chaque phase. Ensuite les énergies des S phases sont sommées et divisées par le temps de la période du phénomène pour avoir la puissance moyenne sur une période de phénomène pour ce module.

Il suffit alors de sommer chaque puissance moyenne de chaque module pour avoir la puissance moyenne du système comme montré dans l'équation 4.3.

$$P_{moy_{sys}} = \sum_{mod=1}^{N} P_{moy_{mod}} \tag{4.3}$$

Les chiffres de puissance en fuite et dynamique des portes ont été extraits de la technologie FDSOI 28nm pour la technologie RVT [3] avec RBB [4] à 0V. La figure 4.3 montre l'évolution du coefficient G_L en fonction de la tension d'alimentation alors que la figure 4.4 représente l'évolution du coefficient G_P en fonction de la tension d'alimentation.

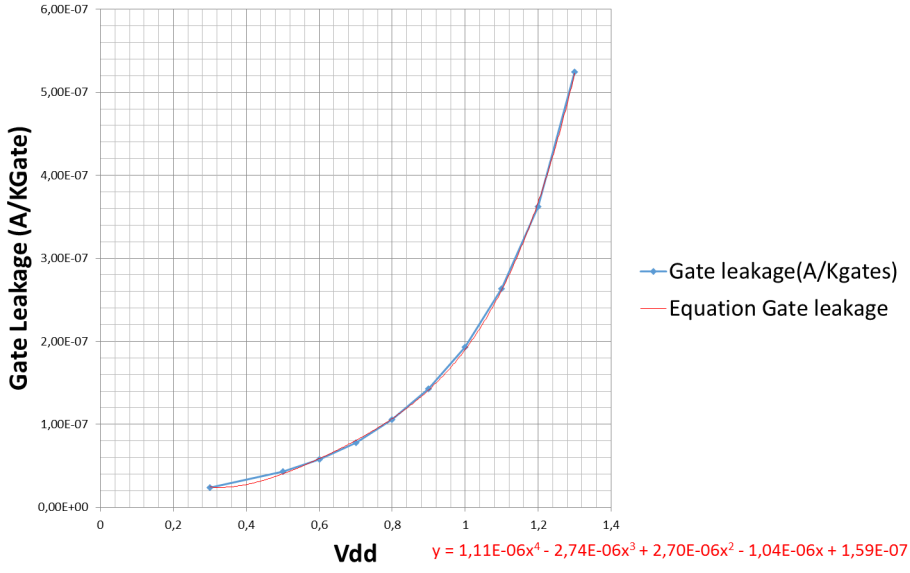


FIGURE 4.3 – Consommation statique des portes FDSOI 28 nm (A/Kgates) en RVT avec RBB = 0V

Pour les mémoires, un modèle a été extrait des SRAM de STMicroelectronics. Les courbes en courant dynamique et statique d'une SRAM FDSOI 28 nm en fonction du nombre d'octet dans la mémoire ont été obtenues. La figure 4.5 représente le courant de fuite en fonction de la capacité mémoire et la figure 4.6 représente la consommation dynamique de la mémoire en fonction de sa capacité en considérant qu'il y a 50% de lecture et 50% d'écriture sur celle-ci. Ces mémoires ont deux modes de consommation, un mode à 1V lorsqu'il y a de l'activité et un mode basse consommation à 0,6V pour la rétention de données.

La consommation du régulateur linéaire (LDO) qui alimente le microcontrôleur a été modélisée en utilisant les équations 4.4 à 4.9. Les puissances d'entrée, de sortie et perdue sont déterminées dans les équations 4.5, 4.6 et 4.7. $I_{quiescent}$ représente le courant perdu dans le contrôle de suivi de la tension, I_{drawn} est le courant tiré par la totalité du LDO, I_{lost} est le courant perdu par $I_{quiescent}$ et le rendement et I_{load} est le courant tiré par la charge. Ainsi en connaissant la tension d'entrée V_{in}, de sortie V_{out}, $I_{quiescent}$ et I_{load} déterminé par le courant de charge du microcontrôleur, I_{lost} peut être déterminé en équation 4.8, et ainsi I_{drawn} est connu (équation 4.9).

3. Regular V_{TH}
4. Reverse Back Biasing

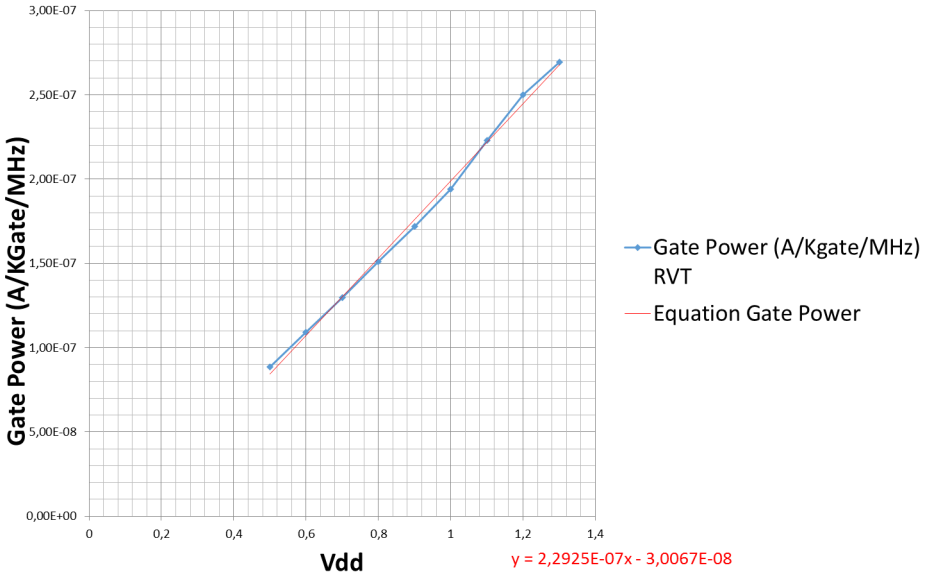


FIGURE 4.4 – Consommation dynamique des portes FDSOI 28 nm (A/Kgates/MHz) en RVT avec RBB = 0V

$$P_{in} = P_{out} + P_{lost} \tag{4.4}$$

$$P_{in} = V_{in} \cdot I_{load} + V_{in} \cdot I_{quiescent} = V_{in} \cdot I_{drawn} \tag{4.5}$$

$$P_{out} = V_{out} \cdot I_{load} \tag{4.6}$$

$$P_{lost} = (V_{in} - V_{out}) \cdot I_{load} + I_{quiescent} \cdot V_{in} \tag{4.7}$$

$$I_{lost} = (1 - \frac{V_{out}}{V_{in}}) \cdot I_{load} + I_{quiescent} \tag{4.8}$$

$$I_{drawn} = I_{lost} + I_{load} \cdot \frac{V_{out}}{V_{in}} \tag{4.9}$$

Les modes de consommation de la radio et du capteur ont été modélisés à l'aide des chiffres fournis dans leur datasheet.

Toutes ces équations et paramètres ont été rentrés dans le modèle. Les différents modes de consommation pour chacun des modules ont ainsi pu être définis. Il suffit alors de fixer les différents modes de consommations de chacun des modules dans différentes phases applicatives pour obtenir un scénario entier de consommation. La section suivante présente une étude de cas pour des scénarios de réception de paquets et des scénarios de mesure sur un capteur.

4.1.4 Étude de cas d'un nœud de capteur

Le but de cette section est de montrer la part de consommation des différentes entités du nœud dans des scénarios réels avec une configuration minimale d'un microcontrôleur. Certains paramètres du modèle ont été fixés comme montré tableau 4.1. Ces paramètres étant définis il reste à déterminer les différentes phases applicatives et de fixer pour chaque module leur mode de consommation.

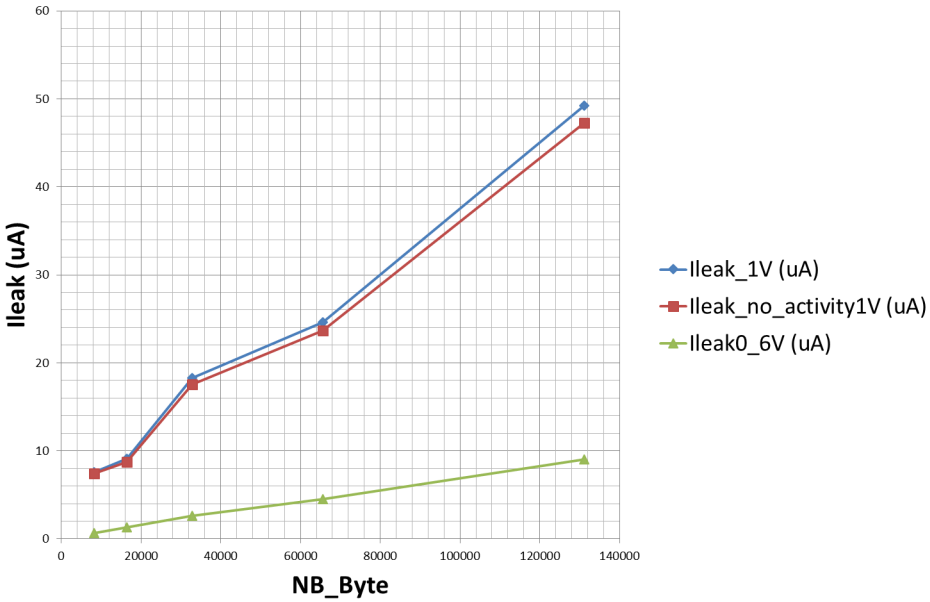


FIGURE 4.5 – $Ileak = f(nb_Byte)$, consommation statique des SRAM FDSOI 28 nm à 1V en mode actif et 0,65V en mode rétention

4.1.4.1 Scénario de Réception

Dans ce scénario de réception, on suppose recevoir toutes les 60 secondes 128 octets. Ainsi, le temps de réception du paquet dépend du débit qui est 250Kb/s. C'est le processeur Cortex M0+ qui s'occupe des deux phases de réception et traitements des données. Le temps de traitement dépend du nombre d'instructions à exécuter, du CPI [5] et de la fréquence de fonctionnement. Ensuite la configuration des modules dans les phases s'effectue comme dans le tableau 4.2

Les résultats de la puissance moyenne consommée pour le scénario de réception sont présentés figure 4.7. Cette répartition de la puissance nous montre que la communication radio a un impact très fort sur la puissance consommée. Ensuite, le convertisseur linéaire intégré dans le microcontrôleur, utilisé ici pour la régulation de ses modules internes, a une consommation non négligeable et son rendement est d'autant plus faible que la différence de tension d'entrée et de sortie est grande. Ainsi, le microcontrôleur dans ce scénario consomme 21% de la puissance moyenne et la radio 79% de puissance moyenne sur les $48,5\mu W$. Pour ce scénario, le microcontrôleur a une configuration minimale pour faire exécuter un programme.

4.1.4.2 Scénario de Mesure

Le même travail a été réalisé pour un scénario de mesure et les résultats de la puissance consommée sont montrés figure 4.8. Dans ce scénario de mesure, le microcontrôleur consomme à lui tout seul 98% de la puissance moyenne contrairement au capteur qui en consomme 2% sur les $10,3\mu W$ au total. Le capteur de pression possède beaucoup de modes de consommation et permet d'en réduire considérablement son impact. Un GPS par exemple aura une consommation nettement plus importante dans ce genre de scénario.

5. Clock Per Instruction

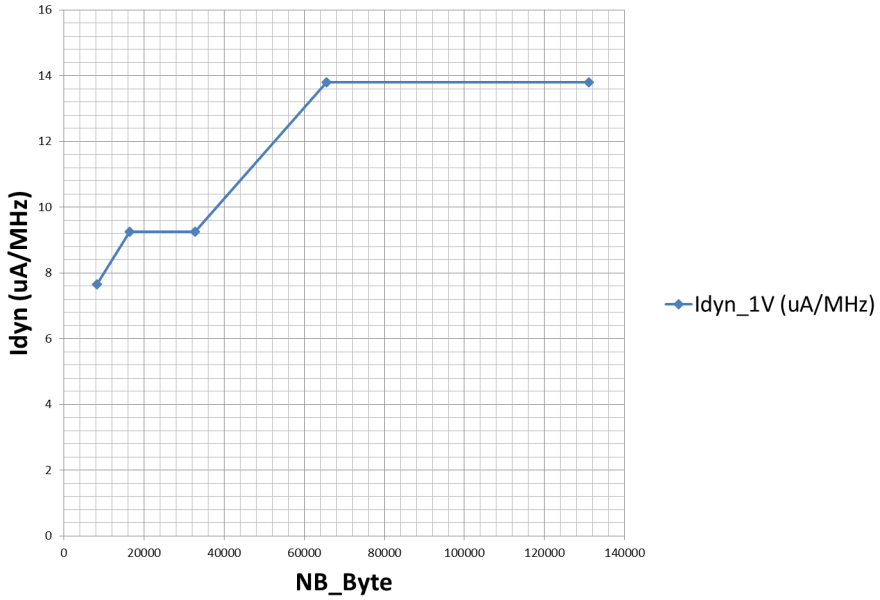


FIGURE 4.6 – $Idyn = f(NB_Byte)$, consommation dynamique des SRAM FDSOI 28 nm à 1V

4.1.5 Synthèse

D'après les premières simulations d'un nœud de capteurs communicant, la répartition de la consommation est très différente suivant le scénario envisagé. En effet, pour un scénario majoritairement de réception ou transmission, le microcontrôleur représente uniquement 20% de la consommation alors que pour un scénario majoritairement de mesure, celui-ci représente 98% de la consommation (mais peut varier aussi en fonction du capteur utilisé). Ce niveau de simulation au niveau d'un nœud complet est trop haut pour essayer de voir l'impact de l'ajout d'un processeur de réveil sur la consommation d'un nœud de capteurs. C'est pourquoi les prochaines simulations se concentrent sur le microcontrôleur seul. De plus la radio et les capteurs dépendent beaucoup trop de l'application et rendrait l'analyse incorrecte.

4.2 Impact de l'utilisation d'un processeur de réveil

Nous nous concentrons sur la simulation de la consommation du microcontrôleur pour les mêmes phases applicatives que précédemment afin d'avoir une évaluation des gains en consommation qui pourraient être obtenus par l'ajout d'un processeur de réveil (WUC [6]). Ainsi, l'objectif est de pouvoir comparer les gains en consommation de deux architectures de microcontrôleur avec et sans le processeur de réveil. Dans un premier temps une présentation de l'architecture du microcontrôleur avec et sans processeur de réveil va être faite pour ensuite définir les différents modes de consommation du microcontrôleur et enfin expliquer la différence de fonctionnement entre une architecture partitionnée entre un processeur de réveil et un processeur principal et une architecture avec un seul processeur dans les scénarios applicatifs considérés.

6. Wake-Up Controller

Modules	paramètres	valeurs
LDO	Vin	1,2V
	Vout RUN	1V
	Vout LP	0,6V
Oscillateur	Freq	32768Hz
	Iosc	$1\mu A$
FLL	Freq	48MHz
	Ifll	$140\mu A$
M0+	Gates	15KGates
	Activité RUN	0,5
	NB Inst	2000
	CPI	1
AHB bus	Gates	2000
	Activité RUN	0,5
Timer	Gates	500
SRAM	Taille	16KB
SPI	Gates	500
CC2420	RX NB Data	128Byte
BMP180	Mesure par réveil	1
	Mode mesure	ULP
	Temps conversion	f(mode)
-	Période	60s

TABLE 4.1 – Paramètres fixés pour les scénarios

	Rétention	Réception	Traitement
LDO	LP	RUN	RUN
Oscillateur	RUN	RUN	RUN
FLL	OFF	ON	ON
M0+	Power ON	Clk Power ON Idle	Clk Power ON RUN
AHB Bus	Power ON	Clk Power ON Idle	Clk Power ON RUN
Timer	LP	RUN	RUN
SRAM	LP	RUN	RUN
CC2420	Power Down	RX	Power Down

TABLE 4.2 – Configuration du mode de consommation des différents modules dans chaque phase applicative

4.2.1 Architecture du microcontrôleur simulé

Les tâches du microcontrôleur dans un nœud de capteurs sont multiples et peuvent se décomposer en trois catégories qui sont le calcul sur les données, le transfert de données et la gestion de la puissance. En effet, celui-ci doit s'occuper de l'allumage et de l'arrêt de différents blocs en réponse à des évènements internes et externes. Il doit également récupérer les données mesurées par les capteurs à travers les périphériques et aussi envoyer et recevoir les données converties à travers les périphériques de communication vers la radio. C'est pourquoi nous proposons d'ajouter un *Wake Up Controller* (WUC) dédié seulement aux tâches de transfert de données et de mesures ainsi qu'à la gestion de la puissance à grain très fin du nœud et des modules internes du microcontrôleur.

Le modèle de simulation est toujours basé sur la technologie FDSOI 28 nm. Plusieurs autres périphériques ont été ajoutés comme un bus I2C pour la mesure sur les capteurs, un deuxième Timer, ainsi que des mémoires SRAM séparées pour les données (8KB) et le programme (64KB) et une mémoire de sauvegarde (2KB). La partie conversion de l'énergie a été enlevée pour se concentrer uniquement sur l'architecture du microcontrôleur. La figure 4.9 présente le modèle de consommation mis en place pour un microcontrôleur avec et sans

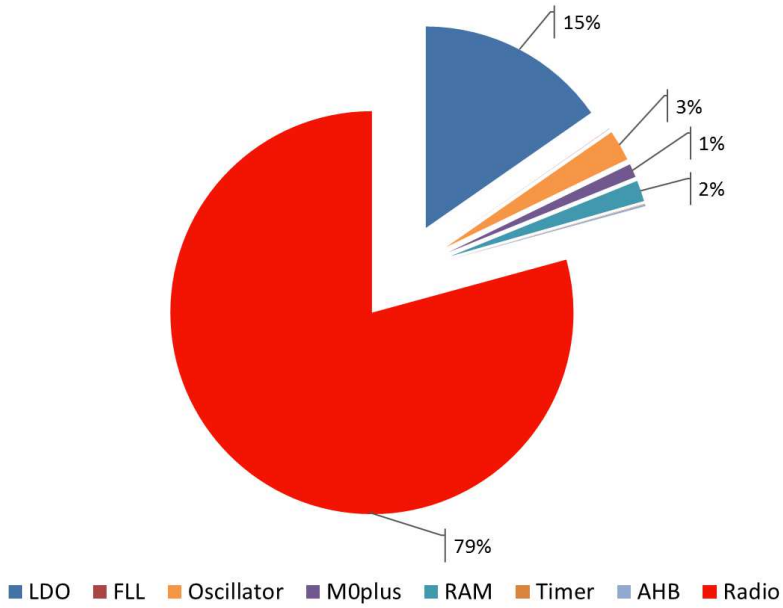


FIGURE 4.7 – Répartition de la puissance moyenne consommée dans le nœud de capteurs pour un scénario de réception radio

WUC. Trois tensions d'alimentation pour la logique ont été incluses : haute tension (HV) à 1V, basse tension (LV) à 0,6V et tension près du seuil (NTV) à 0,3V. Pour les mémoires, uniquement les tensions HV et LV sont utilisées.

Ainsi, pour comparer les deux architectures, le WUC sera utilisé dans certaines phases applicatives au lieu de réveiller le processeur principal dans toutes les phases comme avec une architecture à un seul processeur (cf. section 4.2.3).

Le modèle du microcontrôleur possède trois domaines de tension :

- Vlogic : le processeur principal, les périphériques et les bus de communication peuvent être coupé individuellement ;
- Mémoire : mémoire de programme, de données et sauvegarde. Elles peuvent être coupées individuellement ;
- *Always On*[7] : la RTC[8], l'oscillateur 32K et le WUC dans le cas d'une architecture partitionnée.

Ainsi, pour chaque module, différents modes de consommation ont été définis en fonction des tensions d'alimentation et de l'horloge active.

4.2.2 Modes de consommation du microcontrôleur

Dans ce modèle, quatre modes de consommation ont été définis. Ils sont résumés dans le tableau 4.3 pour un microcontrôleur classique avec un seul processeur. Le mode RUN dans lequel l'horloge et l'alimentation des différents blocs sont activés. Le mode IDLE où l'horloge du CPU et de la mémoire sont coupées. Le mode STANDBY où les domaines mémoires et le domaine Vlogic ont l'horloge coupée. Dans ce mode-ci les sources de réveil sont déjà limitées. Dans le mode OFF, tout le domaine Vlogic est coupé. Dans ce mode le domaine des mémoires est géré en fonction de l'implémentation qui est faite. Si le programme est dans une mémoire non volatile alors celle-ci peut être coupée, sinon elle

7. Toujours allumé
8. Real Time Clock

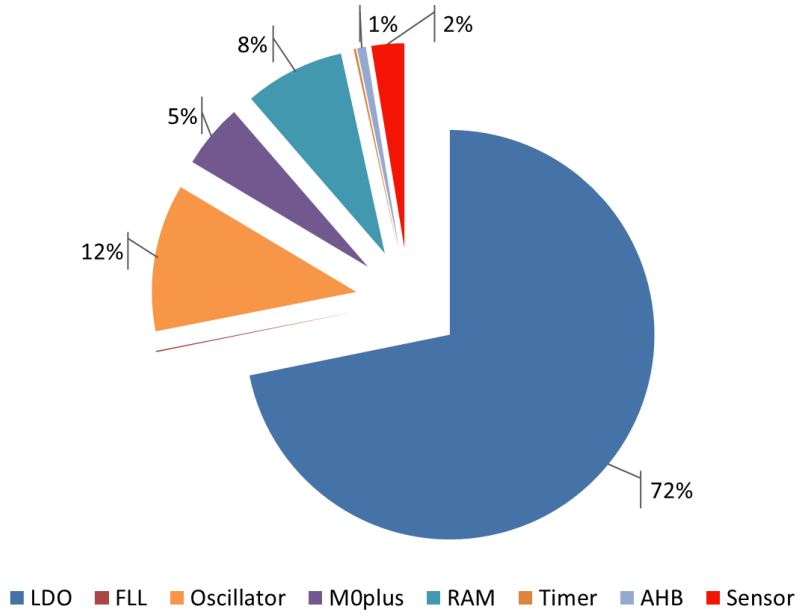


FIGURE 4.8 – Répartition de la puissance moyenne consommée dans le nœud de capteur pour un scénario de mesure sur un capteur de pression

reste sous tension de rétention. La mémoire de données sera coupée et perdra toutes les valeurs, c'est pourquoi une mémoire de backup, si elle est implémentée, peut être utilisée pour la rétention des données les plus importantes. Dans le mode OFF, très peu de sources de réveil sont disponibles. Dans le cas du STM32L0, seules 3 sources de réveil sont présentes ce qui peut devenir problématique pour certaines applications.

L'utilisation du *Wake Up Controller* avec une architecture partitionnée permet de couper à grain très fin les périphériques basse consommation de son domaine et de couper tout le domaine du processeur principal. Cela permet aussi d'avoir les sources de réveil voulues dans les modes de consommation les plus bas. Le but est de pouvoir s'adapter à n'importe quelle application tout en restant dans les modes de veille avec la plus basse consommation.

4.2.3 Spécification du processeur de réveil dans chaque phase applicative

Dans cette section l'utilisation dans chaque phase du *Wake Up Controller* va être détaillée. Une illustration de la différence de comportement entre une architecture partitionnée entre un *Wake Up Controller* et un processeur principal et une architecture classique de microcontrôleur en est faite figure 4.10. Pour un microcontrôleur classique (figure 4.10b), l'unique processeur s'occupe de toutes les phases applicatives (figure 4.10a). En l'occurrence, l'utilisation du *Wake Up Controller* dans une architecture partitionnée (figure 4.10c) permet de ne pas réveiller le processeur principal pour la tâche de réveil, de mesure ou de transfert de donnée à travers le lien radio. Cette architecture permet de réveiller uniquement le processeur principal quand des calculs seront nécessaires ou qu'une tâche irrégulière est nécessaire. La figure 4.10c montre le partitionnement d'un microcontrôleur entre une partie *Always Responsive* contenant le *Wake Up Controller* et les périphériques basses consommation et une partie *On Demand* contenant le processeur principal, ses mémoires et ses périphériques hautes performances. Ainsi, des gains en consommation sont possibles surtout dans la phase de mesure/réception/transmission et

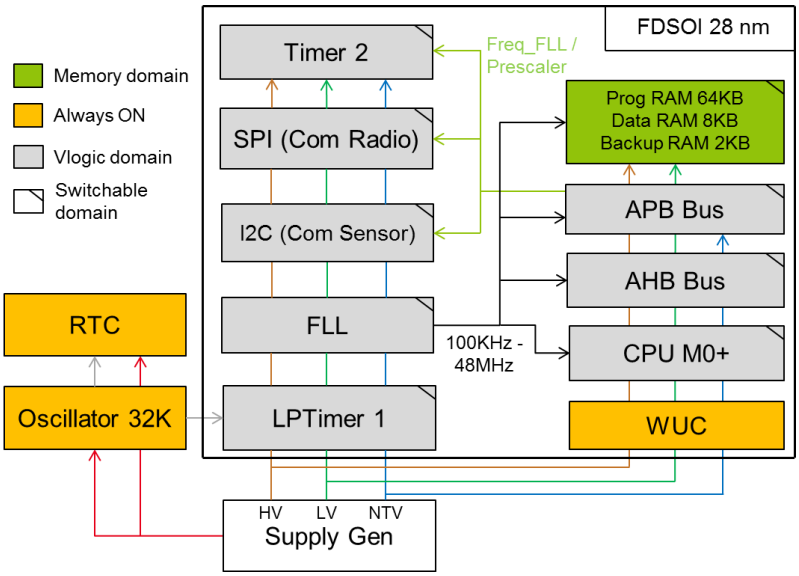


FIGURE 4.9 – Architecture du microcontrôleur simulé pour les simulations de consommation avec et sans processeur de réveil (WUC)

dans la phase rétention, comme montré figure 4.10a. Si les calculs minimaux peuvent être faits directement par le WUC alors le processeur principal n'a pas besoins d'être réveillé.

Maintenant que les modèles de consommation du microcontrôleur sont définis, la comparaison des deux architectures dans les différentes phases applicatives et pour des scénarios complets peut être réalisé dans la section suivante.

4.3 Simulations de la consommation d'un microcontrôleur dans des scénarios de nœuds de capteurs avec et sans processeur de réveil

Ce chapitre va présenter les différentes estimations de gains en consommation d'un microcontrôleur avec une architecture partitionnée pour les différentes phases applicatives et pour des scénarios complets. Ces résultats ont été publiés dans l'article [47] et ont été présentés en juin 2015 à la conférence NEWCAS. Les principaux résultats ainsi que le début des spécifications du WUC ont aussi été présentés dans une *Fresh Ideas* à la conférence ASYNC en 2015 dans le papier [46].

4.3.1 Simulations dans les différentes phases applicatives

4.3.2 Simulations dans la phase de rétention

La figure 4.11 présente la consommation du microcontrôleur avec et sans WUC pour la phase de rétention. Pendant cette phase les modes STANDBY et OFF peuvent être utilisés. Ces simulations prennent en compte les cas où les mémoires sont non volatiles (_NV) ou volatile (_V) pour le programme et données.

Dans le mode STANDBY la principale différence vient du fait que le processeur principal peut être entièrement coupé quand le WUC est disponible. Si la mémoire de programme du processeur principal est non volatile et qu'une mémoire de backup est implémentée, alors l'architecture partitionnée peut facilement rentrer dans le mode OFF. De plus, le WUC

Mode	Effet sur les domaines d'horloge	Effet sur le domaine de tension Vlogic	Effet sur le domaine de tension de la mémoire	Sources de réveil
RUN_HP	Clk Periph On Demand, CPU Clk ON, RAM Clock ON	High Voltage	High Voltage	All
RUN_LP		Low Voltage	Low Voltage	
RUN_ULP		Near Threshold Voltage	Low Voltage	
IDLE_HP	Clk Periph On Demand, CPU Clk OFF, RAM Clock OFF	High Voltage	High Voltage	All
IDLE_LP		Low Voltage	Low Voltage	
IDLE_ULP		Near Threshold Voltage	Low Voltage	
STANDBY_LP	VLogic + Memory Domain	Low Voltage	Low Voltage	LPTimer, I2C, RTC, GPIO(all)
STANDBY_ULP		Near Threshold Voltage	Low Voltage	
OFF	Clocks OFF	Power Gated	Implementation Dependant	RTC, GPIO(2 pins)

TABLE 4.3 – Définition des modes de puissance du microcontrôleur simulé

étant sensible aux évènements et pouvant s'activer sur chacune des sources de réveil, il est capable d'exécuter du code sans latence sur ces évènements. Ici la consommation supplémentaire du WUC n'est pas représentée, elle sera présentée uniquement pour les scénarios complets. Les consommations dans le mode OFF affichent à peu près la même consommation de puissance, ce qui était attendu car presque tous les modules sont coupés. La seule différence vient du fait que le WUC peut être réveillé par beaucoup plus de sources préalablement sélectionnées et commence à exécuter le code presque instantanément.

4.3.3 Simulations dans la phase de réception, transmission et mesure

C'est dans la phase de réception/transmission/mesure que les gains les plus significatifs peuvent être obtenus. En effet, dans cette phase le processeur principal est coupé ainsi que sa mémoire de programme si elle est non volatile. Si une mémoire de backup est utilisée pour sauvegarder les données les plus importantes alors la mémoire de données peut aussi être coupée. Ainsi, dans l'idéal, toute la partie *On Demand* de la figure 4.10c est coupée.

La figure 4.12 présente une comparaison de la consommation des deux architectures avec et sans WUC. Ces simulations montrent que l'utilisation du WUC permet de réduire de 84% pour un mode de puissance équivalent la consommation du microcontrôleur dans cette phase. Toutes les consommations dynamiques et statiques du CPU et de ses mémoires sont supprimées.

4.3.4 Simulations dans la phase de calcul

Dans la phase de calcul, deux possibilités s'offrent au WUC. Soit les calculs à effectuer sont simples et, dans ce cas là, le WUC n'a pas besoin de réveiller le processeur principal. Soit ils sont complexes et alors le WUC a besoin de réveiller le processeur principal. Le WUC n'est peut-être pas obligé de réveiller le processeur principal à chaque période de fonctionnement, mais peut par exemple cumuler les mesures et réveiller le processeur principal pour travailler sur beaucoup de mesures à la fois. Si le processeur principal n'est

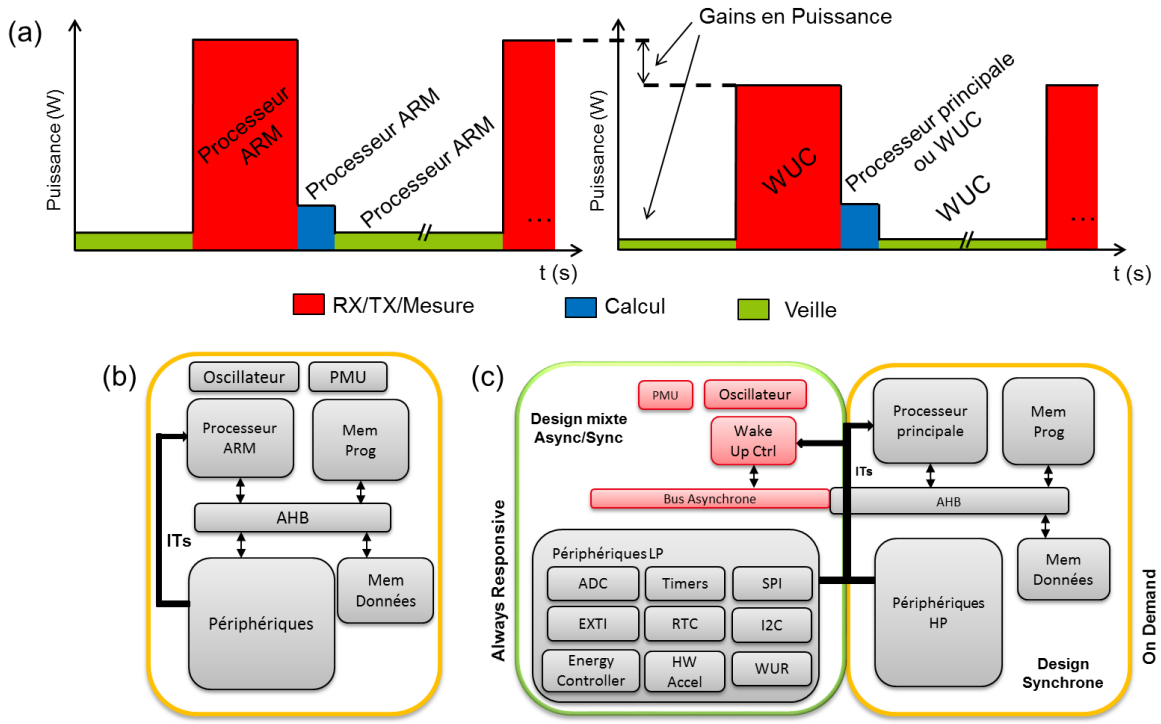


FIGURE 4.10 – Décomposition des phases applicatives (a) d'un scénario de nœud de capteurs pour différentes architectures de microcontrôleur (b) architecture de microcontrôleur classique (c) architecture partitionnée : partie *Always Responsive* avec le *Wake Up Controller* (WUC) et les périphériques basses consommations et un partie *On Demand* avec le processeur principal, ses mémoires (programme et données) et ses périphériques hautes performances

pas réveillé alors, comme dans la phase de réception/transmission/mesure, un gain en consommation de 84% peut être obtenu dans la phase de calcul. Sinon, aucun gain n'est à prévoir dans cette phase si le processeur principal est réveillé. Dans les scénarios complets, le processeur principal est réveillé à chaque période.

4.3.5 Simulations dans des scénarios applicatifs

Maintenant que toutes les phases applicatives ont été étudiées, des scénarios complets peuvent être analysés. Les simulations de la consommation vont être faites pour des applications de très faibles activités à des activités moyennes.

4.3.5.1 Scénario avec une très faible activité

Dans ce scénario de très faible activité, la période de fonctionnement est fixée à 60s, soit 1 évènement/min. Ici, un scénario de réception est imaginé avec une phase de rétention suivi d'une phase de réception et d'une phase de calcul. Dans le cadre d'un microcontrôleur classique, pour la phase de rétention, le mode STANDBY_LP est utilisé et le mode RUN_HP est appliqué pour la phase de réception et calcul, correspondant à une fréquence de 32MHz à 1V. En phase de réception, 128 octets sont reçus et 3000 instructions sont exécutées pendant la phase de calcul. Les transitions ont aussi été considérées pour le passage d'un mode de consommation à l'autre avec des temps de transition et de la consommation supplémentaire. Cela comprend les coûts de redémarrage du processeur, les temps de sta-

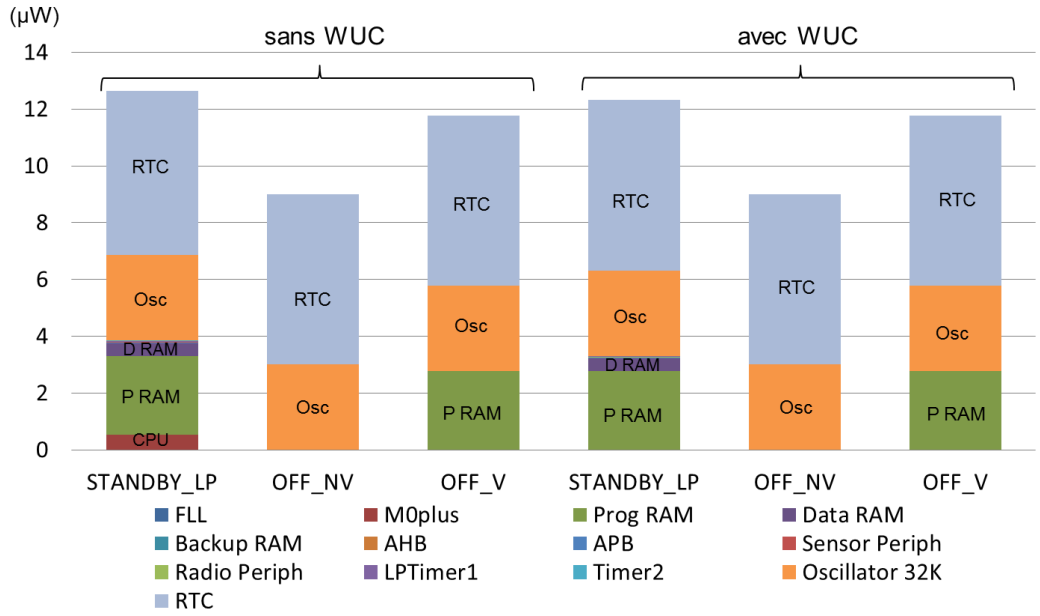


FIGURE 4.11 – Phase de rétention pour les modes basse consommation comparant la consommation du microcontrôleur avec et sans Wake Up Controller

bilisation des tensions, les temps de verrouillage de la FLL et les temps de reconfiguration des périphériques. La figure 4.13 montre les résultats de la consommation moyenne d'un microcontrôleur avec et sans WUC pour ce scénario de réception à très faible activité. Pour un microcontrôleur classique, le mode *standby* est utilisé pour la phase de rétention afin d'avoir une fonctionnalité équivalente au mode off du microcontrôleur avec le WUC en termes de sources de réveil.

On observe une réduction de la consommation de 14,5% avec l'utilisation du WUC en considérant que celui-ci consomme en moyenne $1\mu W$ dans ce scénario.

4.3.5.2 Scénario avec une activité moyenne

Un scénario d'activité moyenne a été défini avec une période de réveil de 100 ms soit 10 évènements/s. La consommation moyenne du microcontrôleur avec et sans WUC est montré figure 4.14 avec les mêmes autres paramètres. Les résultats nous montrent une estimation de la réduction de la consommation moyenne de 76% avec l'utilisation du WUC.

4.3.6 Synthèse

Dans cette partie, nous avons pu étudier les modèles de consommation d'un microcontrôleur basé sur la technologie FDSOI 28nm et sur quelques caractéristiques provenant de *datasheets* de composants du commerce. Une architecture partitionnée entre une partie *Always Responsive* contenant le *Wake Up Controller* (WUC) ainsi que les périphériques basses consommations et une partie *On Demand* contenant le processeur principal, ses mémoires et les périphériques haute performance a été définie. Des simulations ont pu être faites dans les différentes phases applicatives pour nœuds de capteurs. Ensuite, des simulations de scénarios complets ont été réalisées pour différentes activités. Ainsi, pour des scénarios à très faible activité (1 évènement/min) le WUC permettrait de réduire la

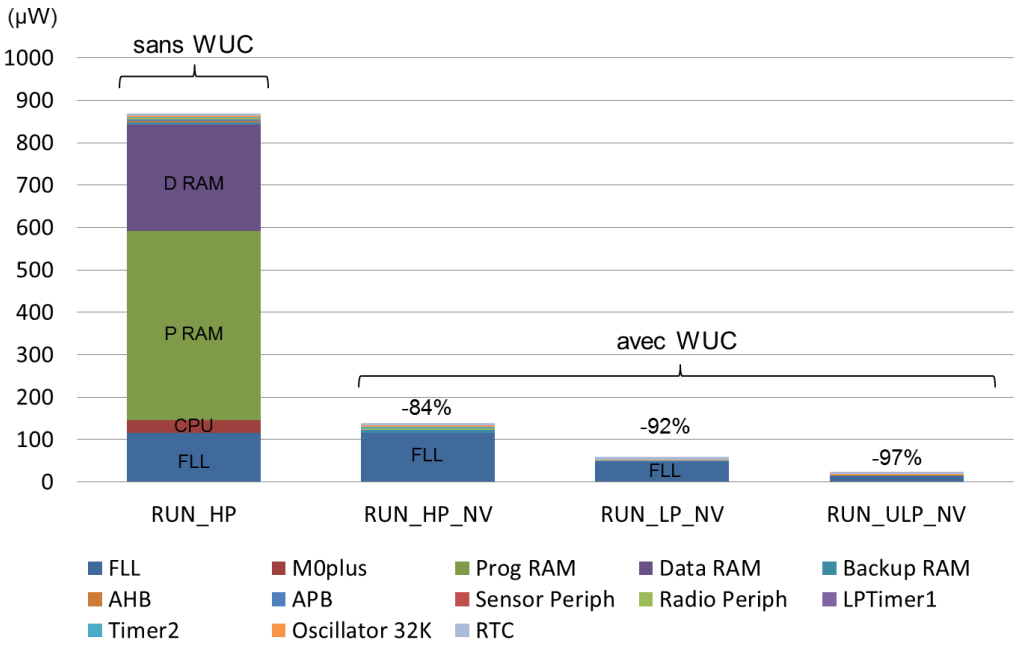


FIGURE 4.12 – Phase de réception/transmission ou mesure pour des modes de consommation du microcontrôleur avec et sans Wake Up Controller

consommation moyenne du microcontrôleur de 14,5% alors que pour une activité moyenne (10 évènements/s) le WUC permettrait de réduire la consommation moyenne du microcontrôleur de 76%. La contribution de chaque sous module devient très différent en fonction de l'activité de l'application. En effet, dans le cas d'une application dont l'activité est très faible, les principaux contributeurs sont l'oscillateur et la RTC, alors que pour des applications à activité moyenne, les principaux contributeurs deviennent la FLL, le processeur et les mémoires.

Maintenant qu'une estimation de gain en consommation d'une telle architecture est déterminée, une implémentation physique de cette partie *Always Responsive* va être mise en place. C'est l'objet de la seconde partie du manuscrit qui traitera de la spécification du processeur de réveil (WUC) et de la plateforme de réveil, de sa microarchitecture et des tests en performances et consommations de l'implémentation physique. Une explication de la méthodologie de travail et de la logique asynchrone sera aussi présentée dans la seconde partie.

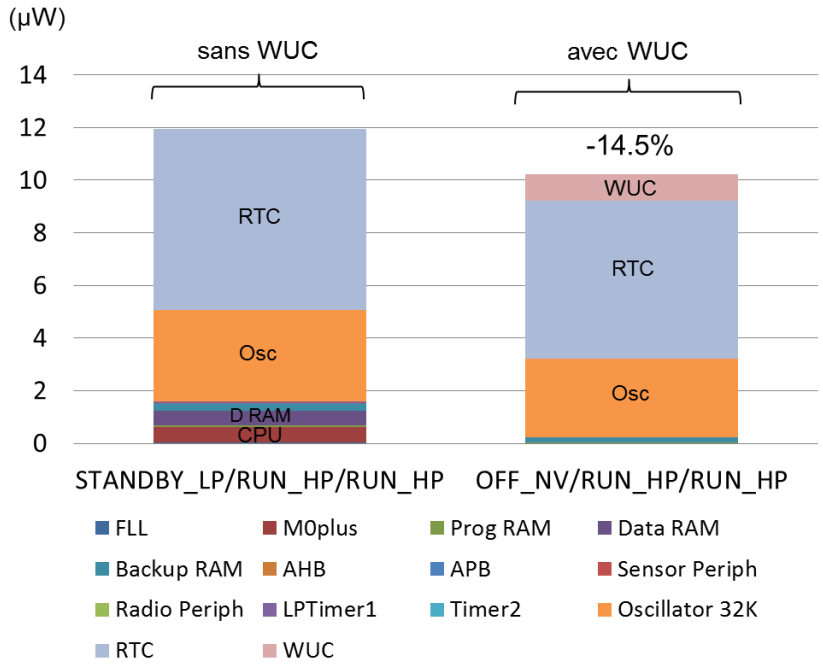


FIGURE 4.13 – Comparaison de la consommation d'un microcontrôleur avec et sans Wake Up Controller dans un scénario de nœud de capteurs de très faible activité

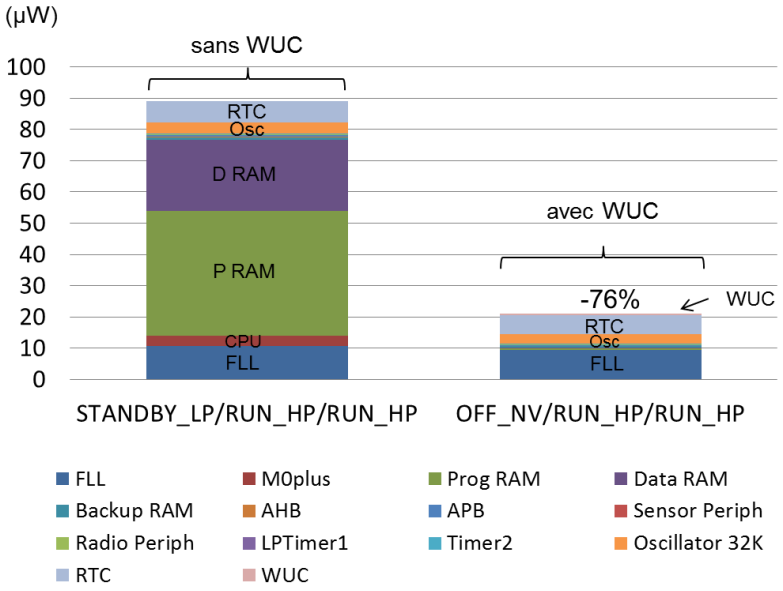


FIGURE 4.14 – Comparaison de la consommation d'un microcontrôleur avec et sans Wake Up Controller dans un scénario de nœud de capteur à activité moyenne

Deuxième partie

Conception d'un processeur de réveil ultra basse consommation

Chapitre 5

Spécifications et architecture

Nous avons pu voir qu'une architecture partitionnée entre un processeur de réveil et un processeur principal permettait de réduire la consommation moyenne d'un microcontrôleur pour nœuds de capteurs communicants. Maintenant deux choix s'offrent à nous pour la réalisation de ce module matériel. Soit développer une solution pré-câblée et configurable à la manière d'une DMA [1] soit concevoir un processeur programmable et donc plus flexible pour l'application mais par contre plus consommant. Étant donné le nombre de capteurs et radios avec lesquels ce module doit être capable de communiquer et s'interfacer, alors une solution programmable est préférable. C'est pourquoi notre choix s'est dirigé sur un processeur.

La conception d'un microcontrôleur est un projet d'envergure et multidisciplinaire. En effet il faut comprendre les besoins logiciels, applicatifs et énergétiques, être au courant des contraintes matérielles et technologiques et savoir développer des architectures tenant compte de tous ces paramètres. Le microcontrôleur est l'intersection de toutes ces disciplines.

Nous avons délibérément choisi de ne pas cloner un jeu d'instructions existant, cela nous laisse beaucoup plus de flexibilité et nous libère de toutes contraintes venant du synchrone et d'architecture existantes. Néanmoins certaines ressemblances existent avec les jeux d'instructions tels que celui de ARM V6 [30] ou du MSP430 de TI [154] car cela reste un jeu d'instructions de type RISC et en plus de taille 16 bits donc assez répandu.

Les spécifications, l'architecture et une partie de la micro-architecture ont été présentées dans l'article de journal Solid State Electronics [48]. Ce chapitre a pour but de présenter dans un premier temps l'architecture partitionnée du microcontrôleur développé dans le cadre de cette thèse, le modèle de programmation et les tâches exécutées par le *Wake Up Controller* [2] dans la partie *Always Responsive* du microcontrôleur, pour enfin en arriver à son jeu d'instructions et les spécifications de la plateforme de *Wake Up* ainsi que l'architecture du processeur de réveil.

5.1 Méthodologie de travail

La première spécification à déterminer est le jeu d'instructions du processeur. Celui-ci a pu être travaillé avec des personnes côté logiciel et compilation du laboratoire LIALP au CEA LETI. Un premier jeu d'instructions leur a été proposé afin qu'ils développent le compilateur. Jusqu'à avoir un premier système fonctionnel, nous nous faisions des retours réguliers sur les problèmes côté matériel et sur les problèmes côté logiciel afin d'identifier les instructions indispensables à ajouter, ou en supprimer certaines inutiles. Le jeu d'instructions est présenté dans le détail section 5.4.3, l'architecture du WUC est présentée en

1. Direct Memory Access
2. WUC

section 5.5 et la micro-architecture du WUC est détaillée chapitre 6. Une première version de la plateforme fonctionnant uniquement en simulation fonctionnelle leur a été fournie pour qu'ils puissent tester leurs premiers programmes compilés. Ensuite la partie pour faire du deboggage intrusif a été développé en fonction des besoins logiciels. Enfin une version synthétisable sur la technologie FDSOI 28nm a été implémentée pour avoir une macro asynchrone et pour pouvoir l'intégrer dans un circuit final. Le circuit final nommé WALPP, pour *Wakeup Asynchronous Low Power Processor*, a été développé conjointement entre une personne s'occupant de la partie mémoire, une personne s'occupant du placement routage, une autre personne s'occupant de l'intégration *top* et moi m'occupant des simulations post-backend, du développement des *drivers* C du circuit, et des applications pour tester le bon fonctionnement de toutes les possibilités du circuit. Des simulations et des tests de la consommation et des performances ont été extraites des simulations post-backend et des tests du circuit seront présentés section 7.4.

5.2 Architecture partitionnée du microcontrôleur

La figure 5.1 présente l'architecture globale du microcontrôleur. Seule la partie *Always Responsive* constitue la contribution de cette thèse. Cette partie est développée en conception mixte asynchrone-synchrone alors que la partie *On Demand* sera conçue en conception synchrone. Le jeu d'instructions du WUC est présenté section 5.4.3, et l'architecture du processeur de réveil section 5.5. La micro-architecture de la plateforme *Always Responsive* sera présentée quant à elle chapitre 6.

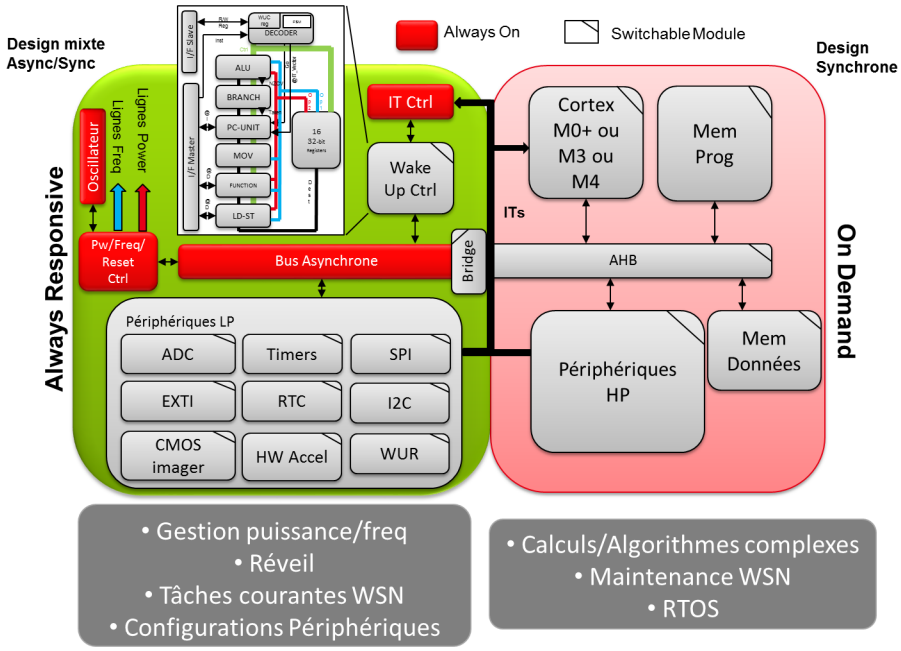


FIGURE 5.1 – Architecture partitionnée du microcontrôleur entre une partie *Always Responsive* contenant le *Wake Up Controller* (WUC) et une partie *On Demand* contenant le processeur principal

5.3 Modèle de programmation et tâches exécutées par la plateforme de réveil

5.3.1 Tâches exécutées par le processeur de réveil

Comme il a pu être présenté section 4.2.1, les tâches dont le WUC doit s'occuper sont les suivantes :

- gérer chaque mode de puissance/fréquence des sous modules du microcontrôleur et des périphériques extérieurs,
- gérer les interruptions/évènements quand le processeur principal est coupé,
- exécuter les routines d'interruption (ISR [3]) associées à chaque vecteur d'interruption,
- gérer les transferts de données entre les capteurs/radios et la mémoire,
- reconfigurer les périphériques, les radios ou les capteurs dont l'alimentation a été coupée,
- redonner le contrôle au processeur principal quand des tâches irrégulières ou de calculs sont à exécuter.

Par rapport à ces tâches, plusieurs types d'instructions peuvent être déduites. Le réveil et la configuration de périphériques impliquent au niveau des instructions des opérations de logique et de masquage ainsi que des instructions de chargement/sauvegarde (*load/store*). Des exemples d'opérations de masquage sont présentés ci-dessous pour mettre à 1 certains bits d'un registre ou mettre à 0 certains autres bits d'un registre. VALUE est une valeur où la position des 1 indique l'endroit dans le registre où faire un *set* ou bien un *clear*.

```
1 REG_CFG |= VALUE; //Set bit
2 REG_CFG &= ~VALUE;  //Clear bit
```

Les tâches de transfert de données se font à l'aide d'instruction *load/store* et doivent supporter plusieurs tailles de données. Il va être aussi nécessaire de faire quelques calculs sur des données comme des additions ou soustractions ou encore des opérations logiques comme des décalages ou des opérations bit à bit comme les opérations logiques ET, OU, XOR, etc. Comme expliqué en section 5.3.2 et 5.3.3, le WUC exécute du code sur évènements, il exécute en fait des machines d'états dont le passage d'un état à l'autre peut être conditionné par certaines valeurs de variables constituant les prédicats. Ces conditions doivent donc être évaluées par des instructions de branchements. Au niveau programmation, il est la plupart du temps utile de coder des fonctions et de faire un appel à elles durant l'exécution du code, et il est préférable de définir des instructions pour le support d'appel à des fonctions. Les entrées en interruptions sont automatisées par le matériel pour rendre le processeur le plus réactif possible. De cette manière, en quelques nanosecondes, le processeur commence déjà à exécuter du code. Pour annoncer la fin d'une IT, une instruction spéciale sera nécessaire.

5.3.2 Programmation évènementielle

Le processeur ne va exécuter que des routines d'interruptions, par conséquent aucune fonction *main* n'est exécutée. Un contrôleur d'interruption vérifie en continu si une interruption s'est déclenchée et indique au processeur l'interruption la plus prioritaire à prendre en compte. Ces interruptions proviennent des différents périphériques internes au microcontrôleur comme par exemple un SPI, une UART, le contrôleur d'interruptions externe, un ADC, etc.

3. Interrupt Service Routine

Ainsi, dans un cas général, lorsque le système est en veille, plusieurs interruptions peuvent être permises et lorsque l'une d'entre elle se déclenche, le processeur est immédiatement réveillé et le numéro de l'IT [4] lui est envoyé. Le processeur charge alors l'adresse de la routine d'interruption et charge ensuite la première instruction correspondante à l'ISR [5]. Cette ISR s'exécute jusqu'à complétion. Si pendant l'exécution de l'ISR plusieurs autres interruptions se sont déclenchées, alors le contrôleur d'interruption enverra immédiatement après la fin de l'ISR courante, le numéro de l'IT la plus prioritaire.

5.3.3 Exemple de programmation par machine d'état

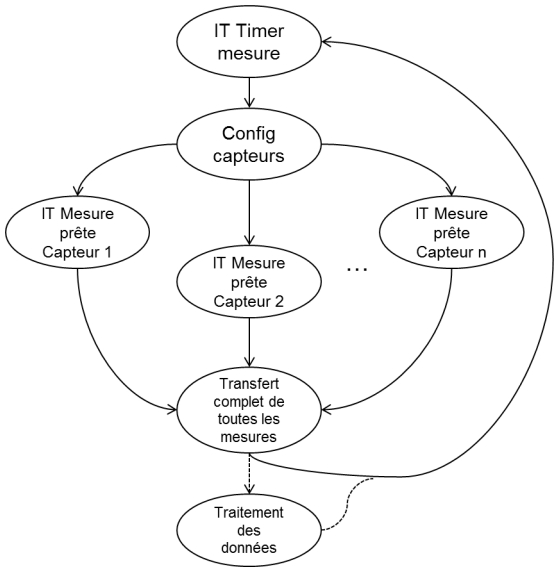


FIGURE 5.2 – Exemple de machine d'états pilotée par des interruptions. L'exemple ici montre une FSM pour lancer des mesures sur n capteurs

La figure 5.2 montre un exemple de machine d'états se réveillant sur une interruption timer, venant allumer et configurer les différents capteurs s'ils étaient éteints, lançant une mesure sur chacun d'entre eux. Ensuite, le WUC est en attente des ITs provenant des périphériques de communication branchés aux capteurs. Celles-ci signalent que les mesures sont prêtes et les valeurs peuvent être lues dans les registres des périphériques (ADC, registre de réception de l'I2C, le SPI ou l'UART). Et, dès que toutes les mesures sont complétés, alors le WUC peut soit réveiller le processeur principal pour traiter les données, soit les traiter lui même, soit se mettre en mode d'attente d'IT et donc de veille.

5.4 Jeu d'instructions et spécifications

5.4.1 Architecture RISC 16 bits

Le jeu d'instructions retenu pour le *Wake Up Controller* est un jeu d'instructions RISC 16 bits (section 3.1) proposant certaines instructions arithmétiques et logiques, de branchement, et d'accès mémoire ainsi que quelques instructions pour le support de fonctions. Ce processeur possède 16 registres généraux, des registres d'états et de configurations et un support pour le debug-on-chip.

4. Interruption
5. Interrupt Service Routine

L'architecture mémoire est une architecture de type Von Neumann dans l'optique de limiter la surface de la partie *Always Responsive*. Ce processeur possède une seule mémoire pour le programme et les données avec un seul port d'accès. Il a été décidé d'intégrer uniquement 4KB de mémoire. Cela parait amplement suffisant vu les tâches assignées au WUC, et permet de réduire la part de consommation de puissance de la mémoire.

La mémoire ne peut être accédée qu'avec des instructions de chargement/sauvegarde. Aucune autre instruction ne peut accéder à la mémoire de données, ce qui correspond bien à une architecture RISC. Les modes d'adressage pour les accès mémoire sont restreints à trois types : l'adressage indirect à registre (`[Rs] -> Rd` ou `Rd -> [Rs]`), l'adressage base + index (`[Rs1 + Rs2] -> Rd` ou `Rd -> [Rs1 + Rs2]`), l'adressage base + offset (`[Rs + #imm] -> Rd` ou `Rd -> [Rs + #imm]`).

Les instructions de branchements ont uniquement un type d'adressage relatif (`PC + offset -> PC`), mais le compteur de programme (`PC`[6]) peut aussi être modifié par une instruction de déplacement `MOV` ou par les instructions d'appel à fonction `BL` et `RET` via un adressage indirect (`Rs -> PC`).

Les instructions arithmétiques et logiques ne travaillent que sur les registres généraux grâce à un adressage registre (`Rs op Rd -> Rd`) ou registre et immédiat (`Rd op #imm -> Rd`). Ces instructions vont mettre à jour des bits de statut qui vont ensuite être utilisés pour les instructions de branchement conditionnel.

Certains des registres généraux ont une fonction particulière, il s'agit du registre `R6`, `R7` et `R15`. En effet ceux-ci correspondent respectivement au registre de lien (`LR`[7]), au pointeur de pile (`SP`[8]) et au compteur de programme (`PC`[9]). Le *Link Register* sert à sauvegarder l'adresse de retour lors d'un appel à une fonction, le pointeur de pile sauvegarde l'adresse de la pile. La pile permet de sauvegarder ou restaurer le contexte lors d'appel de fonction. Cette pile est elle aussi incluse dans la mémoire de programme/données.

L'architecture proposée pour le WUC est une architecture qui exécutera uniquement une instruction à la fois, et ne pourra pas charger une instruction tout en exécutant une autre, ceci pour une question de gestion de conflit d'utilisation de ressources. Il faudra alors moins de matériel pour le contrôle pour l'accès aux ressources.

5.4.2 Spécifications

5.4.2.1 Gestions des interruptions

Le WUC contient un contrôleur d'interruptions dont le but est d'indiquer au processeur le numéro de l'interruption la plus prioritaire qui s'est levée. Les interruptions ne peuvent pas se préempter pour des raisons de simplifications du matériel et parce que la préemption ne se prête pas bien à ce genre d'applications. La préemption serait utile pour des applications temps réel s'utilisant avec un RTOS[10] (FreeRTOS, ARM RTX...), or ici ce n'est pas le cas pour le WUC. Les interruptions sont triées par numéro d'interruption et n'ont pas de registre de configuration pour fixer leur priorité. Plus le numéro d'interruption est petit, plus elle est prioritaire. Il existe des registres de configuration pour permettre les interruptions et pour déclencher les interruptions de manière logicielle. Une routine d'interruption s'exécutera jusqu'à complétion. Le vecteur d'interruption qui sauve les adresses des routines d'interruptions est stocké dans le haut de la mémoire avec

6. Program Counter
7. Link Register
8. Stack Pointer
9. Program Counter
10. Real Time Operating System

l'adresse de la routine IT0 stockée à l'adresse 0xFFFE par défaut, puis l'adresse de la routine IT1 à l'adresse 0xFFFC et ainsi de suite. Si, pour des besoins applicatifs, plusieurs routines d'interruptions sont associées à une même interruption alors il existe un registre de configuration de l'offset du vecteur d'interruption pour que le processeur saute à un autre endroit dans la mémoire pour exécuter la routine.

5.4.2.2 Types de données gérées

Étant donné que ce processeur doit s'interfacer avec n'importe quel type de capteurs ou radio qui peuvent générer des données de tailles très différentes, alors il a été choisi de mettre un chemin de données sur 32 bits. Ainsi le WUC est capable de gérer des données sur 8, 16 et 32 bits en signé et non signé. Les opérations logiques et arithmétiques s'effectue sur 32 bits et ce sont ensuite les *load/store* qui vont permettre de charger ou sauver des données sur 8,16 ou 32 bit en signé ou non signé.

5.4.2.3 Modes de consommation

Le WUC est implémenté en logique asynchrone ce qui engendre un clock-gating naturel dès qu'il n'y a plus de données à traiter. le processeur est ainsi automatiquement en mode de veille dès qu'il a fini d'exécuter ses tâches. Deux modes de consommation existent donc pour le WUC, le mode actif dont la vitesse d'exécution des instructions dépendront de la tension d'alimentation du cœur et le mode veille dont la tension d'alimentation détermine la consommation statique du processeur.

5.4.2.4 *Reset* et *boot* du système

Le signal de *reset* est actif bas et permet d'initialiser les protocoles de communication. La logique asynchrone QDI [11] (section 6.1) utilise une porte spéciale appelé porte de Muller présenté section 6.1.2.1, qui est une porte séquentielle et qui nécessite d'être correctement initialisée pour que tout le système soit au repos après le *reset*. Dans le cadre de la logique WCHB [12] utilisée dans ce circuit, chaque porte de Muller implémentant un rendez-vous entre un signal de donnée et un signal d'acquittement devra être initialisé à 0 pour un signal de donnée et 1 pour le signal d'acquittement. Lorsque le signal de *reset* est relâché, le processeur ne va pas exécuter tout de suite le code de *boot*. Il faut déjà venir écrire dans la mémoire tout le programme, et c'est ensuite que le code de *boot* peut être exécuté en déclenchant de façon logicielle l'interruption 0. Ce code de *boot* a été mis sur l'IT0, l'interruption la plus prioritaire, et est en permanence permis.

5.4.2.5 Périphériques de la plateforme de réveil

L'implémentation des périphériques étant hors du travail de cette thèse, un module de déclenchement d'interruptions a été mis en place pour pouvoir déclencher les interruptions de manière logicielle. Elles sont connectées directement sur les lignes d'interruption et sont déclenchées en écrivant dans les registres d'interface pour simuler le déclenchement des interruptions par des périphériques. La plateforme de réveil a pour but d'accueillir tous les périphériques basses consommation nécessaires pour communiquer avec n'importe quel capteurs ou radios pour les tâches courantes d'un nœud de capteurs.

11. Quasi Delay Insensitive
12. Weak Condition Half Buffer

5.4.2.6 Débogage sur puce

Le débogage sur puce d'un processeur est aujourd'hui indispensable. Comme expliqué section 3.1, deux modes de *debug* existent, le mode intrusif appelé *Foreground Debug Mode* (FGDM) et le mode arrière plan appelé *Background Debug Mode* (BGDM). Le WUC intègre une solution de débogage intrusif pour stopper l'exécution du cœur à tout moment, poser des points d'arrêt (*Breakpoint*) et pour lire et écrire dans tout l'espace d'adressage du WUC et de ses registres internes. L'architecture et le principe de fonctionnement de la partie debug du WUC est présenté section 5.5.5.

5.4.2.7 Système d'alimentation

Les simulations de la consommation du microcontrôleur ont montré qu'il était intéressant de mettre en place une multitude de domaine d'alimentation pour agir à grain très fin sur la consommation. Ainsi il a été choisi dans le circuit de séparer les domaines d'alimentation de la mémoire (Vdd_mem) et la partie asynchrone (Vdd_wuc) pour pouvoir faire varier indépendamment leur tension et évaluer leur impact sur la consommation moyenne et les performances du circuit. La partie asynchrone possède aussi des tensions d'alimentation Vdds et Gnds pour la polarisation arrière du caisson (FBB car la technologie utilisée ici est du FDSOI en LVT). Des cellules de conversion de niveau de tension sont présentes entre chaque domaine d'alimentation. La tension du cœur pourra être plus basse que celle de la mémoire qui est caractérisée pour 0,6V.

L'objectif est de pouvoir faire fonctionner le processeur sur une très large gamme de tension de 0,3V à 1,2V et avoir une caractérisation du circuit en terme de consommation/-performance/réveil pour un grand nombre de points de fonctionnement.

5.4.3 Jeu d'instructions

Le jeu d'instructions du WUC est un jeu d'instructions RISC 16 bits ayant trois modes d'adressage : l'adressage indirect à registre, l'adressage base + index, l'adressage base + offset. Différents types d'instructions ont été mis en place avec des instructions à 3 opérandes, 2 opérandes sur les registres avec la possibilité d'utilisation d'immédiats en signés ou non signés.

Dans un premier temps les instructions vont être présentées au niveau fonctionnel en différents groupes (tableau 5.1). Il y a 7 groupes : les instructions mémoires, arithmétiques, logiques, de gestion des signes, de support d'appel aux fonctions, de branchements et les instructions spéciales.

	Mémoire	Arithmétique	Logique	Gestion signe	Fonction	Branchement	Spéciale
33 instructions	LDRB LDRH LDR LDRSB LDRSH STRB STRH STR MOVBL MOVBU MOV PCREL	ADD SUB COMP	AND XOR OR BIC BIT NOT LSL LSR	SXTB SXTH UXTB UXTH	PUSH POP BL	B	ENDIT BREAK

TABLE 5.1 – Récapitulatif des 33 instructions du *Wake Up Controller* regroupées par type

5.4.3.1 Instructions mémoire

Les instructions mémoire sont les instructions qui permettent de faire des chargements de données vers les registres généraux du WUC, des sauvegardes des registres généraux du WUC vers l'espace d'adressage et des déplacements de données dans les registres généraux. Le récapitulatif des instructions mémoire et de ce qu'elles exécutent sont présentés tableau 5.2.

Type	Instruction	Assembleur	Opération
MOV	MOVBL imm	MOVBL #imm, Rd	{24'b0 : [imm8]} -> Rd
	MOVBU imm	MOVBU #imm, Rd	{16'b0 : [imm8] : 8'b0} -> Rd
	MOV R	MOV Rs, Rd	Rs -> Rd
LOAD	LDRB imm	LDRB [Rs, #imm] , Rd	ZeroExtend([Rs+imm5][7 :0]) -> Rd
	LDRH imm	LDRH [Rs, #imm], Rd	ZeroExtend([Rs+(imm5 :0)][15 :0]) -> Rd
	LDR imm	LDR [Rs, #imm], Rd	[Rs+(imm5 :00)] -> Rd
	LDRSB imm	LDRSB [Rs, #imm] , Rd	SignExtend([Rs+imm5][7 :0]) -> Rd
	LDRSH imm	LDRSH [Rs, #imm], Rd	SignExtend([Rs+(imm5 :0)][15 :0]) -> Rd
	LDRB R-R	LDRB [Rs1, Rs2], Rd	ZeroExtend([Rs1+Rs2][7 :0]) -> Rd
	LDRH R-R	LDRH [Rs1, Rs2], Rd	ZeroExtend([Rs1+Rs2][15 :0]) -> Rd
	LDR R-R	LDR [Rs1, Rs2], Rd	[Rs1 + Rs2] -> Rd
	LDRSB R-R	LDRSB [Rs1, Rs2], Rd	SignExtend([Rs1+Rs2][7 :0]) -> Rd
	LDRSH R-R	LDRSH [Rs1, Rs2], Rd	SignExtend([Rs1+Rs2][15 :0]) -> Rd
	LDRB R	LDRB [Rs], Rd	ZeroExtend([Rs][7 :0]) -> Rd
	LDRH R	LDRH [Rs], Rd	ZeroExtend([Rs][15 :0]) -> Rd
	LDR R	LDR [Rs], Rd	[Rs] -> Rd
	LDRSB R	LDRSB [Rs], Rd	SignExtend([Rs][7 :0]) -> Rd
	LDRSH R	LDRSH [Rs], Rd	SignExtend([Rs][15 :0]) -> Rd
	PCREL	LDR Label, Rd	[PC + (offset8 :0)] -> Rd
STORE	STRB imm	STRB [Rs, #imm], Rd	Rd[7 :0] -> [Rs + imm5][7 :0]
	STRH imm	STRH [Rs, #imm], Rd	Rd[15 :0] -> [Rs+(imm5 :0)][15 :0]
	STR imm	STR [Rs, #imm], Rd	Rd -> [Rs+(imm5 :00)]
	STRB R-R	STRB [Rs1, Rs2], Rd	Rd -> [Rs1 + Rs2][7 :0]
	STRH R-R	STRH [Rs1, Rs2], Rd	Rd -> [Rs1 + Rs2][15 :0]
	STR R-R	STR [Rs1, Rs2], Rd	Rd -> [Rs1+Rs2]
	STRB R	STRB [Rs], Rd	Rd[7 :0] -> [Rs][7 :0]
	STRH R	STRH [Rs], Rd	Rd[15 :0] -> [Rs][15 :0]
	STR R	STR [Rs], Rd	Rd -> [Rs]

TABLE 5.2 – Récapitulatif des instructions mémoires du *Wake Up Controller*

Les périphériques et la mémoire de programme/données étant tous interconnectés sur le même bus de communication, il n'y a alors pas besoin d'instructions dédiées pour les périphériques et pour la mémoire. Pour gérer la taille des données, les instructions de chargement (*Load*) et de sauvegarde (*Store*) existent en version octet (*Byte*), demi-mot (*Half Word*) et mot (*Word*). Pour charger et travailler sur des nombres signés, les instructions *Load* existent aussi pour les nombres signés en étendant leur signe sur 32 bits. L'adressage dans le WUC s'effectue en *Little Endian* (l'adresse de l'octet, du demi mot ou du mot pointe sur l'octet de poids faible). La mémoire a une largeur de 32 bits et peut être écrite par octet, demi mot ou mot. Une lecture dans la mémoire renvoie automatiquement un mot de 32 bits. La logique doit donc sélectionner dans le mot la bonne partie et la décaler sur l'octet de poids faible pour le stocker dans le registre général. Pour une sauvegarde dans la mémoire, l'octet ou le demi-mot sont dupliqués sur le bus de 32 bits. Une opération de masquage s'effectue dans la mémoire en fonction de l'adresse où écrire et de la taille de la donnée à sauver.

Les instructions *Load* et *Store* existent sous les trois modes d'adressages expliqués dans la section 5.4.1 et sont possibles pour des octets, des demi-mots et des mots. Il y a des *Load* qui permettent d'étendre le signe d'une donnée signée stockée en mémoire dans le

cas d'un octet ou d'un demi-mot. Il existe aussi une instruction de chargement relatif au compteur programme `PCREL`, pour charger des données provenant d'une table ou *Literal Pool* qui permet de charger un mot de l'adresse courante à l'adresse courante + 510.

L'instruction `MOVBL` permet de charger un immédiat sur 8 bits directement dans un registre alors que l'instruction `MOVBU` permet de charger un immédiat de 8 bits sur l'octet de poids fort d'un demi-mot. En enchainant ensuite avec un une addition en immédiat ceci permet de créer des adresses sur 16 bits.

5.4.3.2 Instructions arithmétiques et logiques

Le jeu d'instructions du *Wake Up Controller* inclut des opérations classiques arithmétiques et logiques. Le tableau 5.3 référence toutes les opérations arithmétiques et logiques du WUC ainsi que leurs opérations et leur action sur les bits de statut du processeur qui sont utilisés par l'unité de branchement. Les opérations logiques sont très utiles pour faire des opérations de masquage afin de configurer des registres par exemple. Les opérations arithmétiques permettent de faire un minimum de calcul sur les données, de créer des adresses ou d'évaluer des prédicats dans le cadre d'exécution de machines d'états. De plus, si les nombres sont signés alors il est nécessaire d'étendre leur signe sur 32 bits si ce sont des données de 8 ou 16 bits (`SXTB`, `SXTH`). Une extension en nombre non signé peut être nécessaire quelque fois pour forcer la conversion de type (*cast*).

Type	Instruction	Assembleur	Mise à jour	Opération
Logique	AND	AND Rs, Rd	Z	Rs . Rd -> Rd
	XOR	XOR Rs, Rd	Z	Rs ^ Rd -> Rd
	OR	OR Rs, Rd	Z	Rs | Rd -> Rd
	BIC	BIC Rs, Rd	Z	$\overline{Rs}$ & Rd -> Rd
	BIT	BIT Rs, Rd	Z	Rs & Rd
	NOT	NOT Rs, Rd	Z	$\overline{Rs}$ -> Rd
	LSL	LSL #shift, Rs, Rd	N,Z,C	$\{Rs << shift5 : shift5'b0\}$ -> Rd
	LSR	LSR #shift, Rs, Rd	N,Z,C	$\{shift5'b0 : Rs >> shift5\}$ -> Rd
Signe	SXTB	SXTB Rs, Rd	N,Z	SignExtend(Rs[7 :0]) -> Rd
	SXTH	SXTH Rs, Rd	N,Z	SignExtend(Rs[15 :0]) -> Rd
	UXTB	UXTB Rs, Rd	Z	ZeroExtend(Rs[7 :0]) -> Rd
	UXTH	UXTH Rs, Rd	Z	ZeroExtend(Rs[15 :0]) -> Rd
Arith	ADD Imm	ADD #imm, Rd	N,Z,C,V	Rd + imm8 -> Rd
	ADD R	ADD Rs, Rd	N,Z,C,V	Rd + Rs -> Rd
	SUB Imm	SUB #imm, Rd	N,Z,C,V	Rd - imm8 -> Rd
	SUB R	SUB Rs, Rd	N,Z,C,V	Rd - Rs -> Rd
	COMP R	CMP Rs, Rd	N,Z,C,V	Rd - Rs

TABLE 5.3 – Récapitulatif des instructions arithmétiques et logiques du *Wake Up Controller*

Ces instructions ont la plupart d'entre elles deux opérandes provenant du banc de registres `R0` à `R15`, et viennent mettre à jour les bits d'état du processeur `N`, `Z`, `C`, `V` qui sont les bits qui indiquent si le résultat de l'opération est négatif, nul, s'il a engendré une retenue (*Carry*) et si le résultat a entrainé un débordement (*Overflow*) quand le signe du résultat diffère du signe des opérandes.

Ces instructions sont les seules instructions à modifier les bits de statut du processeur qui sont ensuite utilisés par l'unité de branchement pour évaluer si une branche doit être prise ou non.

5.4.3.3 Instructions de branchements

Les instructions de branchements permettent d'évaluer le résultat précédent pour savoir s'il faut sauter à un autre endroit dans le programme ou s'il faut continuer à l'instruction

suivante. Ici la gestion des branchements est simple à mettre en place puisque chaque instruction s'exécute jusqu'à complétion avant de charger l'instruction suivante. Ainsi il n'y a pas besoin d'avoir d'instructions dites de *delay slot* après un branchement retardé comme il peut y en avoir dans les processeurs pipelinés pour remplir le pipeline le temps que le branchement soit évalué, ou carrément de stopper le remplissage du pipeline le temps que l'instruction de branchement soit exécutée.

Instruction	Assembleur	Opération	Test
BEQ		Branch if Z == 1	x == y
BNE		Branch if Z == 0	x != y
BHS		Branch if C == 1	uint x $\geq$ uint y
BLO		Branch if C == 0	uint x $<$ uint y
BN		Branch if N == 1	x $<$ 0
BPZ		Branch if N == 0	x $>$ 0
BVS		Branch if V == 1	Branch if Overflow
BVC	*B{cond}Label*	Branch if V == 0	Branch if No Overflow
BHI		Branch if C == 1 and Z == 0	uint x $>$ uint y
BLS		Branch if C == 0 or Z == 1	uint x $\leq$ uint y
BGE		Branch if N == V	int x $\geq$ int y
BLT		Branch if N != V	int x $<$ int y
BGT		Branch if Z == 0 and N == V	int x $>$ int y
BLE		Branch if Z == 1 or N != V	int x $\leq$ int y
BA		Branch always	Branch Always

TABLE 5.4 – Récapitulatif des instructions de branchement du *Wake Up Controller*

Lorsque la condition est vraie alors le PC prend l'adresse de branchement qui est égale à la valeur du PC courant plus un offset signé
(`@Branch = PC + (SignedExtend(Offset9):0)`) ce qui permet de faire des sauts en mémoire de -512 à 511 octets. Si le branchement n'est pas pris alors le `PC` s'incrémente de 2 pour sauter à l'instruction suivante.

Le processeur travaille sur des valeurs signées et non signées et les conditions à tester peuvent être de multiples sortes dans un programme c'est pourquoi les conditions à tester doivent prendre en compte les 4 tests plus grand que, plus grand ou égal, plus petit que et plus petit ou égal pour des nombres signés et non signés, ainsi que les tests égal, non égal, positif et négatif. Dans le WUC, les mêmes conditions que dans le jeu d'instructions ARMV6-M ont été utilisées et sont présentés tableau 5.4 avec l'opération de test sur les bits de statut et la condition mathématique équivalente testée.

5.4.3.4 Instructions d'appels aux fonctions

Lorsqu'une fonction est appelée dans le code il faut alors sauvegarder le compteur programme qui suit la fonction quelque part, mais aussi sauvegarder le contexte des registres généraux dans la mémoire, pour ensuite commencer à exécuter le code de la fonction qui travaille sur les registres généraux. Pour cela une fonction de branchement spéciale *Branch and Link* (`BL`) est implémentée pour sauvegarder l'adresse de retour dans un registre spécial appelé *Link Register* (`LR`) et pour faire sauter le programme à l'adresse de la fonction qui est stocker dans un registre général. L'instruction de retour d'une fonction est émulée par l'instruction `MOV LR, PC`. Pour la sauvegarde et la restauration du contexte lors d'appels de fonctions, des instructions `PUSH` et `POP` ont été implémentées.

Ces instructions prennent en opérande une liste de registres où les 1 signifient qu'il faut sauvegarder en mémoire ou restaurer le registre depuis la mémoire. Ainsi, en une instruction, le processeur est capable de sauvegarder 9 registres en mémoire et d'en restaurer 8 tout en ayant la possibilité de faire brancher le `PC` au précédent `LR` sauvé. Cela évite d'utiliser une multitude de *load* et *store* et accélère les appels et sorties de fonctions. Un

Instruction	Assembleur	Opération
BL	BL Rd	PC + 2 -> LR ; Rd & 0xFFFE -> PC
RET*	RET	LR -> PC
PUSH	PUSH opt, Reglist	Push LR(if opt == 1) and Register onto stack ([R13 R12 R11 R10 R9 R8 R5 R4]) where Reglist[i] == 1
POP	POP opt, Reglist	Pop data from memory to Register ([R13 R12 R11 R10 R9 R8 R5 R4]) where Reglist[i] == 1 and (if opt == 1) load data to PC

*émulée par MOV

TABLE 5.5 – Récapitulatif des instructions de support d'appels aux fonctions du *Wake Up Controller*

exemple de `PUSH` et `POP` et de l'état de la pile après leur exécution est présenté figure 5.3. Lors d'un `PUSH` le pointeur de pile est décrémenté alors qu'avec un `POP` il s'incrémente.

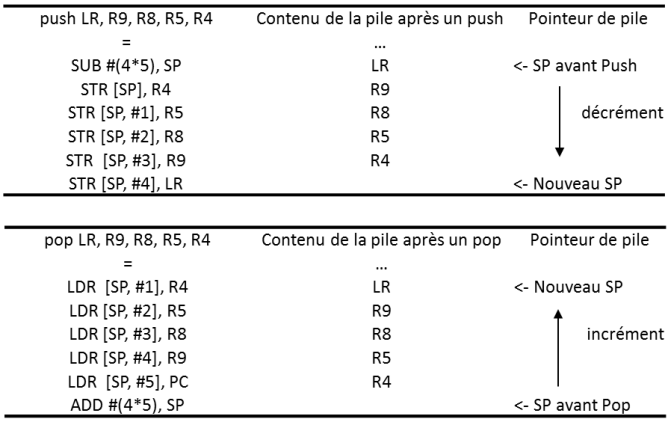


FIGURE 5.3 – Opération d'une instruction `PUSH` et `POP` et état de la pile en mémoire

5.4.3.5 Instructions spéciales

Le processeur ne fonctionnant que sur interruptions. Il a donc besoin de connaitre la fin de la routine d'interruption. Pour cela une instruction `ENDIT` a été implémentée pour que le processeur arrête de charger des instructions et pour que le contrôleur d'interruption soit informé afin qu'il évalue l'interruption la plus prioritaire (tableau 5.6). Dans le cas où une nouvelle interruption est en attente il l'indique immédiatement au cœur sinon celui ci se met en état de veille.

Instruction	Assembleur	Opération
ENDIT	ENDIT	Indique la fin d'une interruption
BREAK	BREAK	met le cœur en attente jusqu'à ce qu'il reçoive un signal de continuation

TABLE 5.6 – Récapitulatif des instructions spéciales du *Wake Up Controller*

Dans le cadre du débogage sur puce, il est très intéressant de pouvoir poser des points d'arrêt, c'est pourquoi une instruction `BREAK` a été implémentée pour que le cœur s'arrête et ne charge plus de nouvelles instructions. L'instruction `BREAK` est écrite à la place de l'instruction sur laquelle on veut s'arrêter. Dès que l'instruction `BREAK` est décodée, le cœur est en attente de commandes pour relancer le cœur ou faire du pas à pas sur le code. Le principe de fonctionnement sera expliqué dans le détail section 5.5.5. Avant de

relancer le cœur il faut veiller à bien réécrire l'instruction qui a été remplacée par le Break en mémoire.

5.4.3.6 Encodage du jeu d'instructions

Afin de simplifier au maximum le format et le décodage des instructions, celles-ci sont toutes codées sur 16 bits. Le tableau 5.7 liste tous les formats d'instructions, ainsi que les instructions concernées par ces formats. Le WUC possède huit sous formats d'instructions regroupés dans quatre gros formats (séparés par les doubles lignes dans le tableau), ce qui permet de faire le pré-décodage sur les trois premiers bits de l'instruction et de distribuer l'instruction au bon décodeur de format ensuite. Le pré-décodage permet aussi d'identifier immédiatement les instructions spéciales pour en informer la machine d'état du décodeur.

16 bits				Instructions
op (5 bits)	imm5	Rs (3 bits)	Rd (3 bits)	(LDR Imm)*3, (LDRS Imm)*2, (STR Imm)*3, LSL, LSR
op (5 bits)	Imm8		Rd (3 bits)	MOVBU, MOVBL, PCREL, ADD Imm, SUB Imm
op (7 bits)	- (1 bit)	Rs (4 bits)	Rd (4 bits)	MOV R, (LDR R)*3, (LDRS R)*2, (STR R)*3, AND, XOR, OR, BIC, BIT, NOT, (SXT)*2, (UXT)*2, ADD R, SUB R, COMP R
op (7 bits)	Rs1 (3 bits)	Rs2 (3 bits)	Rd (3 bits)	(LDR R-R)*3, (LDRS R-R)*2, (STR R-R)*3
op (7 bits)	option (1 bit)	Reglist (8 bits)		PUSH, POP
op (3 bits)	Cond (4 bits)	SignedOffset (9 bits)		B<Cond>
op (5 bits)	- (7 bits)		Rd (4 bits)	BL
op (5 bits)	- (11 bits)			ENDIT, BREAK

TABLE 5.7 – Huit différents formats d'encodage des instructions du *Wake Up Controller*

Une documentation complète existe sur l'encodage du jeu d'instructions pour plus de détail.

5.5 Architecture de la plateforme de réveil

Cette section va présenter l'architecture globale de la plateforme de réveil ainsi que le schéma d'exécution d'une routine d'interruption avec sa séquence d'instructions. Ce processeur exécute une instruction jusqu'à complétion avant de passer à l'autre. Il a été choisi de partir sur ce principe de fonctionnement dans un premier temps pour avoir une logique et une méthode de débogage la plus simple possible.

5.5.1 Architecture du processeur

L'architecture de la plateforme de réveil développée dans cette thèse est présentée figure 5.4. Le module qui permet d'indiquer au processeur d'exécuter du code est le contrôleur d'interruptions. Celui-ci reçoit des signaux d'interruptions provenant des différents périphériques internes (*Timers*, périphériques de communication, accélérateurs matériel, module de contrôle des interruptions externes...) et indique au processeur (*WUC Core*) le numéro de l'interruption la plus prioritaire à exécuter.

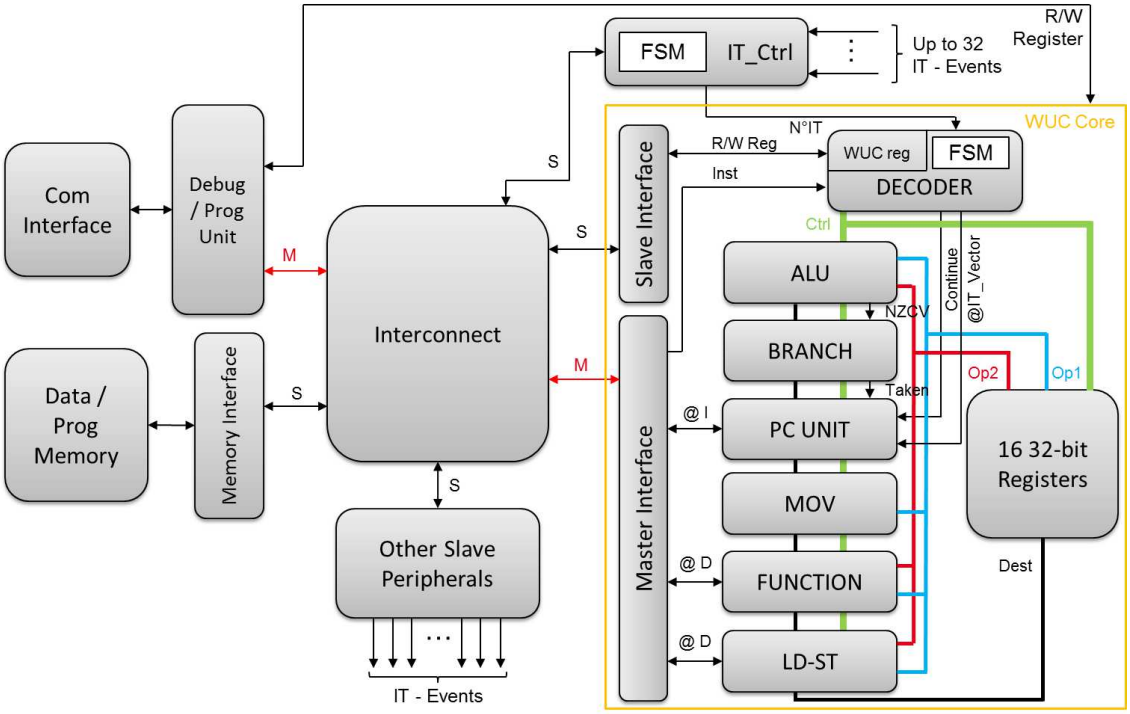


FIGURE 5.4 – Architecture de la plateforme de réveil

Le processeur alors en état de veille va évaluer le numéro de l'interruption et en déduire l'adresse du vecteur d'interruption qu'il envoie à l'unité de chargement des instructions, la PC UNIT. Celle-ci va tout d'abord charger le contenu du vecteur d'interruption depuis la mémoire en passant par l'interface maitre du processeur et le bus d'interconnexion pour connaitre l'adresse de la première instruction de la routine d'interruption puis charger la première instruction. Les instructions sont envoyées à l'unité de décodage (DECODER). Le décodeur élabore les commandes qui sont distribuées aux unités d'exécutions et aux contrôles du chemin de données. Les unité d'exécutions présentes dans le *Wake Up Controller* sont l'unité arithmétique et logique (ALU), l'unité de branchement (BRANCH), l'unité de déplacement (MOV), l'unité de chargement sauvegarde (LD-ST) et l'unité de support aux fonctions (FUNCTION). Les unités sont indépendantes dans leur fonctionnement et pourrait exécuter chacune quelque chose en parallèle, mais ceci a été rendu impossible en bloquant la PC UNIT qui est en attente d'un signal d'incrément du compteur programme (`PC`[13]) et qui est envoyé une fois que l'instruction est exécutée. Il n'y a pas de signaux de contrôle supplémentaires dans l'architecture étant donné qu'en logique asynchrone QDI, c'est le protocole physique de requête/acquittement qui le permet et pas un protocole fonctionnel. Ainsi pour exécuter une instruction, le décodeur va positionner les bons signaux de contrôle au niveau du chemin de données, du banc de registres et de l'unité d'exécution. L'unité d'exécution va travailler sur les données et une fois l'exécution finie, tous les signaux d'acquittements des signaux de contrôle au niveau du décodeur sont relâchés et le signal d'incrément qui correspond au type d'instruction qui vient d'être exécuté est envoyé pour charger la prochaine instruction. Le banc de registres contient des registres gérant le conflit de lecture et écriture car quelque fois une même instruction pourra lire et écrire sur le même registre.

13. Program Counter

5.5.2 Diagramme de temps d'une interruption et boucle Fetch-Decode-Execute

Cette section va présenter le déroulement d'une routine d'interruption ainsi que le déroulement de l'exécution d'une instruction dans la plateforme de réveil.

Le diagramme de temps d'une interruption est présentée figure 5.5. Quatre temps vont être mesurés pour les performances du cœur :

- t_1 : le temps que le contrôleur d'interruption prenne en charge l'interruption au moment ou elle se lève, de ressortir l'interruption la plus prioritaire et de l'envoyer à la PC UNIT ;
- t_2 : chargement du vecteur d'interruption en mémoire qui indique l'adresse de la routine d'interruption associée à l'interruption ;
- t_3 : exécution de la routine d'interruption jusqu'à détection de l'instruction ENDIT ;
- t_4 : ré-évaluation de l'interruption la plus prioritaire et mise en veille si aucune interruption est en attente.

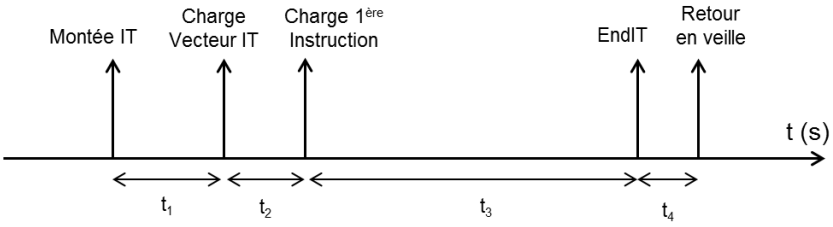


FIGURE 5.5 – Diagramme de temps d'exécution d'une seule interruption

Lorsque le processeur exécute la routine d'interruption dans le temps t_3, le processeur exécute la boucle Fech-Decode-Execute en une fois et possède une synchronisation au niveau de la PC UNIT grâce au signal `I_type` qui est le type d'instruction qui vient d'être exécuté afin d'incrémenter le `PC` en conséquence. La figure 5.6 montre l'activité de certains blocs composant le cœur du WUC et la plateforme de réveil pour l'exécution d'une instruction `ADD`.

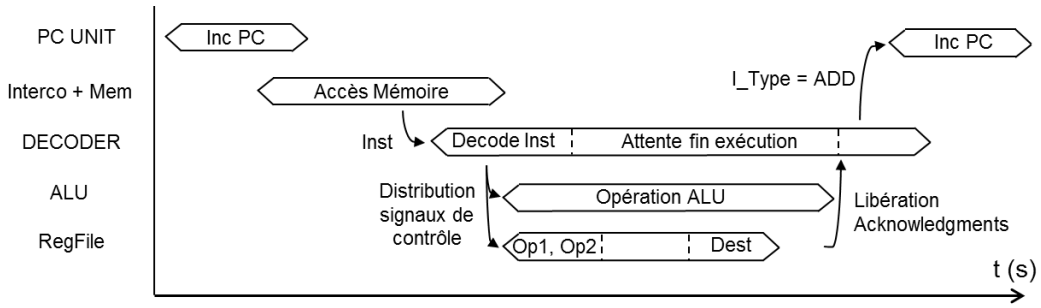


FIGURE 5.6 – Diagramme de temps d'exécution d'une instruction avec l'activité de chacun des blocs de l'architecture de la plateforme de réveil pour l'exécution d'une instruction ADD

Ce diagramme montre l'activité à partir du moment où le bloc reçoit les données jusqu'à ce qu'il revienne dans un état inactif lorsque tous les acquittements des signaux sont revenus à 1. Dans un premier temps la PC UNIT envoie l'adresse de l'instruction à charger. Le bus d'interconnexion prend bien en compte la demande et redirige les demandes sur la mémoire. Les signaux de la PC UNIT reviennent tous à un état inactif grâce aux

acquittements de la logique asynchrone, et la lecture en mémoire s'effectue. Ensuite le décodeur reçoit l'instruction, la décode et envoie les signaux de contrôle à l'ALU car ici un `ADD` s'exécute. Des signaux de contrôle sont aussi envoyés au banc de registres pour charger les opérandes 1 et 2 afin qu'elles soient transmises à l'ALU et prépositionne le numéro de registre de destination pour que l'ALU puisse écrire le résultat. L'opération s'effectue et écrit dans le registre de destination. Le banc de registres devient alors inactif, puis tous les signaux de contrôle envoyés par le décodeur deviennent inactifs (libération des acquittements). Le décodeur envoie alors le signal de contrôle du type d'instructions qui vient d'être exécuté à la PC UNIT, ici un `ADD` qui correspond à une incrémentation normale de +2 du `PC`. La PC UNIT incrémente le `PC` et repart pour une boucle Fetch-Decode-Execute.

5.5.3 Gestion des branchements

L'unité de branchements lit le registre interne des bits de statut de la dernière opération `N`, `Z`, `C`, `V`. L'offset du branchement pris est envoyé à la PC UNIT directement. La BRANCH UNIT évalue si le branchement est pris ou non et l'indique à la PC UNIT qui se charge du calcul du nouveau `PC`. Que ce soit un branchement conditionnel ou non-conditionnel (`BA`), cela suit le même schéma. D'autres branchements non-conditionnels existent et sont créés par les unités MOV, FUNCTION et LD-ST avec les instructions suivantes :

- `MOV Rs, PC`
- `LDRH [Rs], PC`
- `BL Rd`
- `POP LR, Reglist`

Ces instructions engendrent un nouveau `PC` envoyé à la PC UNIT, qui est alors écrit dans le registre `PC` et qui charge l'instruction de cette nouvelle adresse.

5.5.4 Gestion de la taille des données, contenu de la mémoire de programme/données et *memory map*

Le chemin de données qui inclut le banc de registres, l'interconnexion et les unités d'exécution sont sur une largeur de 32 bits. Les adresses et les instructions manipulées sont sur 16 bits. Tous les périphériques sont capables de gérer des données sur 8, 16 et 32 bits grâce à un signal sur le bus indiquant la taille des données transférées (BW [14]).

Un exemple de *memory map* pour un état très avancé de la plateforme de réveil ainsi que le contenu de la mémoire de 4KB est présenté figure 5.7.

Cette figure présente le *memory map* tel qu'il serait s'il y avait beaucoup de périphériques développés. Dans le cas du circuit développé dans cette thèse seul le WUC, L'IT_CTRL et la mémoire sont présents dans le *memory map*. Les périphériques tels que la WUR [15], l'AES [16] ou l'*Energy Ctrl* sont développés par d'autres personnes et seront intégrés plus tard au SoC.

Le contenu de la mémoire est séparé en trois zones qui sont le vecteur d'interruptions contenant les adresses de chacune des routines d'interruption et qui peut être ré-alloué grâce à un registre de configuration de l'offset de la table de vecteur d'interruption dans le WUC, une zone pour la pile placée juste en dessous le vecteur d'interruption dont la taille est définie dans le fichier d'édition de liens et une zone pour le programme, les données en lecture seule et les données.

14. Byte Width
15. Wake Up Radio
16. Advance Encryption Standard

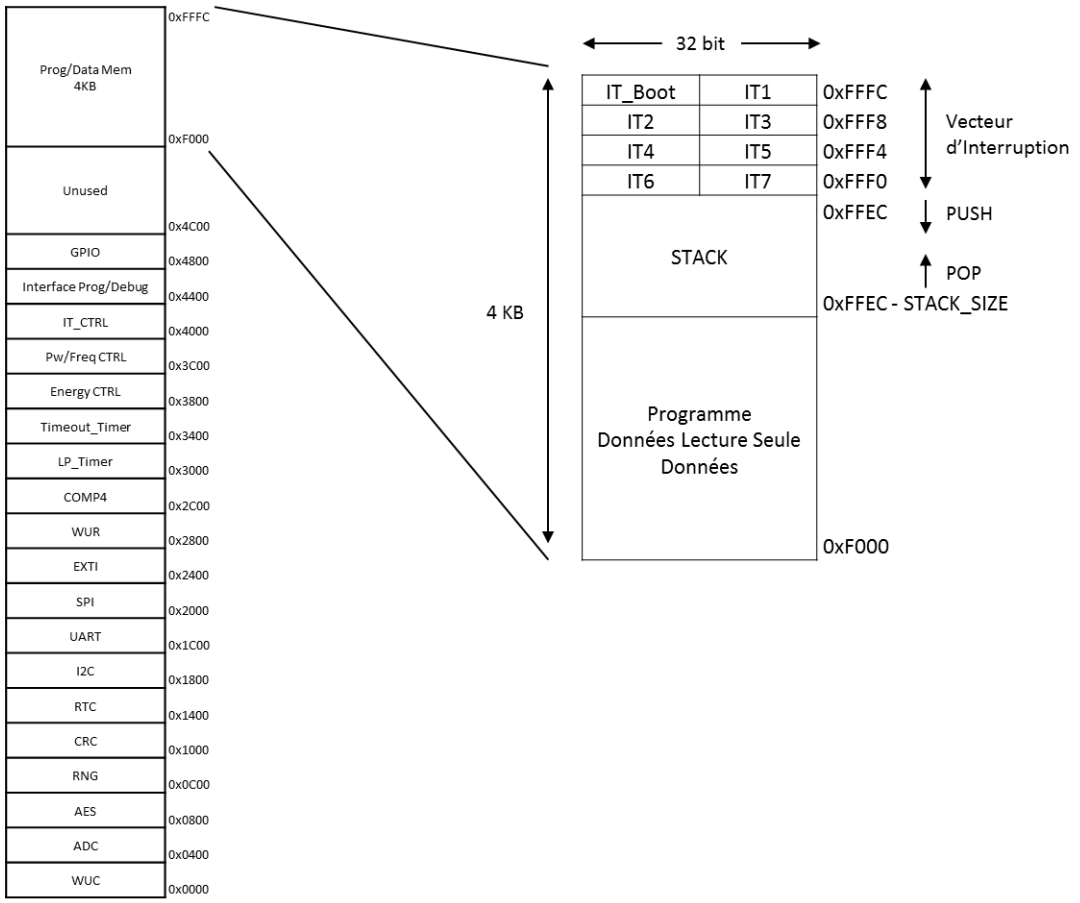


FIGURE 5.7 – Exemple de *Memory map* pour un état avancé de la plateforme de réveil et contenu de la mémoire de 4KB implémenté dans le circuit

5.5.5 Débogage sur puce du processeur

Cette section va présenter le principe de fonctionnement du debug dans la plateforme de réveil, la manière de stopper le cœur, de le relancer et de faire du pas à pas sur l'exécution du code.

Toute la stratégie de debug du cœur se passe dans le signal Continue (figure 5.8) venant de la machine d'état du DECODER. Ce signal est juste un signal *Single Rail* (SREVENT : 1 fil de signal + 1 fil d'acquittement) (voir section 6.1) qui permet de signaler à l'unité de calcul du *Program Counter* (PC UNIT) qu'il peut prendre en compte le nouveau type d'instruction qui s'exécute ou qui a été exécuté (I_type_ctrl dans Figure 5.8) pour pouvoir calculer le nouveau PC et charger la nouvelle instruction. En effet la PC UNIT est en attente de ce signal Continue avant de charger chaque instruction. Il y a donc un rendez-vous entre le SREVENT Continue et le canal dual rail I_type_ctrl. Ce rendez-vous est implémenté avec des portes de Muller (section 6.1.2.1). Ainsi il ne peut y avoir de chargement d'une nouvelle instruction tant que le signal Continue n'est pas envoyé. Ce signal Continue peut être généré soit par les évènements new_inst_ev, step_ev, run_ev ou new_irq_num_ev (figure 5.8). Le cœur est donc naturellement stoppé. Suivant les évènements qui arrivent de l'extérieur ou de l'intérieur, celui-ci va se mettre dans un certain mode et sera sensible à des évènements différents (voir machine d'états contenue dans le décodeur figure 5.9).

La machine d'état du décodeur (figure 5.9) est constituée de deux états : un état dans lequel il est en attente d'une interruption (WAIT_IT) pour pouvoir rentrer dans le second

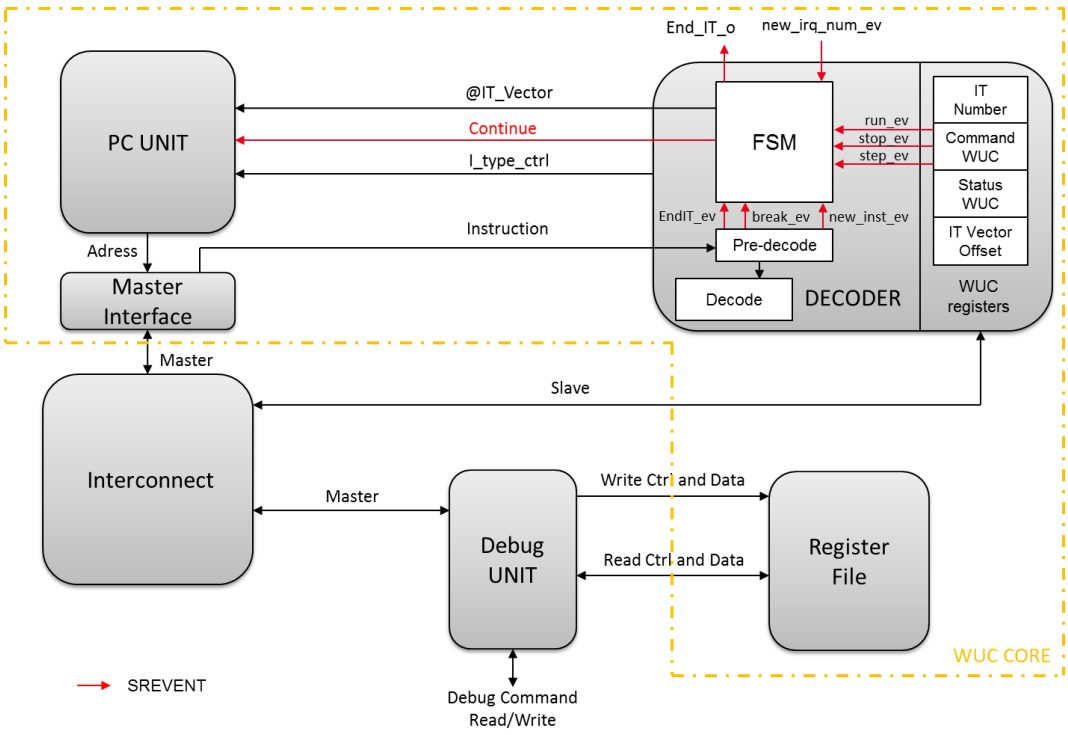


FIGURE 5.8 – Principe de fonctionnement du debug dans le WUC

état d'exécution (WAIT_INST). Dans l'état WAIT_IT la machine d'état configure le bit WAIT du registre d'état du WUC WUCSR (figure 5.10) dans l'état WAIT. Cet état est alors sensible aux évènements new_irq_num_ev, step_ev, run_ev, break_ev ou stop_ev. Lorsque l'évènement new_irq_num_ev se lève, le bit WAIT du registre WUCSR est mis à l'état NOT_WAITING, envoie un signal Continue à la PC UNIT et change l'état de la machine d'état à WAIT_INST. L'adresse contenue dans le vecteur d'interruption est chargée par la PC_UNIT et la première instruction est chargée. Un évènement new_inst_ev est alors généré par le pré-décodeur du DECODER à la réception de l'instruction. Dans l'état WAIT_INST de la machine d'états, cela va vérifier si le cœur est en mode STOP ou RUN grâce à une variable interne et envoie le signal Continue si le processeur est dans un mode RUN ou se met en attente des signaux step_ev ou run_ev si le cœur est dans le mode STOP. A tout moment les signaux step_ev, run_ev et stop_ev peuvent être envoyés grâce à une écriture dans le registre CMDR (figure 5.10). La variable interne de la machine d'états est alors mise à jour avec les signaux run_ev, break_ev et stop_ev.

Lorsque le cœur est en mode STOP, qu'une nouvelle instruction est arrivée et qu'un évènement run_ev ou step_ev est généré, alors le signal Continue est envoyé à la PC UNIT. Lorsque le pré-décodeur détecte l'instruction `ENDIT`, il génère aussi l'évènement EndIT_ev pour que la machine d'état retourne à l'état WAIT_IT.

Dans le cas d'un *Software Breakpoint* posé dans le code, le pré-décodeur détecte l'instruction `BREAK` et envoie un signal break_ev à la machine d'états qui change alors le mode du processeur au mode STOP. Le pré-décodeur envoie alors l'évènement new_inst_ev et le type d'instruction à la PC UNIT spécifiant de recharger la même instruction. La machine d'états est alors à ce moment en attente des évènements run_ev et step_ev venant de l'utilisateur. Celui-ci doit d'abord remplacer dans la mémoire de programme à l'adresse où est écrit le `BREAK` l'instruction originelle puis doit écrire dans le registre de contrôle CMDR RUN ou STOP pour exécuter la vraie instruction. Ainsi grâce à cette machine d'états, on

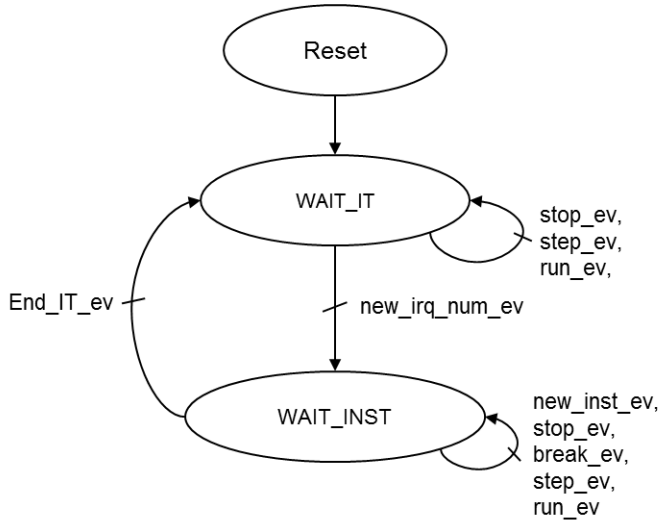


FIGURE 5.9 – Machine d'état du décodeur

peut être capable de mettre le processeur soit en mode STOP ou RUN, regarder le numéro de l'IT qui est exécutée, voir si le processeur est en attente ou non d'interruptions, et faire du pas à pas sur l'exécution du code. Il est possible de voir aussi le statut des bits `N` (*Negatif*), `Z` (*Zero*), `C` (*Carry*), `V` (*Overflow*), et de changer l'offset du vecteur d'interruption pour exécuter un autre code en mémoire correspondant à une même interruption.

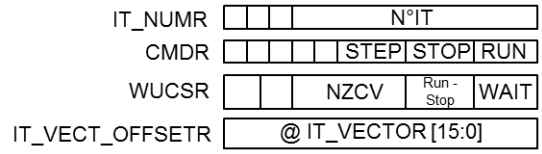


FIGURE 5.10 – Registre du WUC pour le support de debug, son état et sa configuration

Maintenant que les spécifications du système ont été déterminées, la microarchitecture des différents modules va pouvoir être présentée.

Chapitre 6

Microarchitecture de la plateforme de réveil

En se basant sur les spécifications du chapitre 5 et sur la méthode de conception asynchrone exposée dans ce chapitre, section 6.1, la microarchitecture de la plateforme de réveil va être présentée dans le détail. Ce chapitre va présenter la microarchitecture de chacun des blocs de l'architecture de la plateforme de réveil de la figure 5.4 et les différents processus qui les composent pour respecter le jeu d'instructions. En effet il a fallu décomposer chaque bloc en des processus élémentaires faisant apparaitre les dépendances entre les différentes ressources et l'échange de données entre les blocs fonctionnels.

Dans un premier temps la méthode de conception asynchrone va être présentée, puis l'architecture de la mémoire ainsi que son interface asynchrone-synchrone seront exposées. Nous détaillerons ensuite dans la section 6.3 les différentes unités d'exécution : le contrôleur d'interruption, le décodeur, l'unité PC, le banc de registres, le bus de communication, l'unité *load-store*, l'unité arithmétique et logique, l'unité de branchement et de déplacement et enfin l'unité de support aux fonctions. Nous finirons par une description de l'environnement logiciel ainsi que l'environnement de conception et de simulation.

6.1 Méthode de conception asynchrone

Cette section est dédiée à l'explication des bases de la logique asynchrone, ses différentes possibilités d'implémentation avec des exemples de synthèse d'opérateurs utilisant l'une des portes principales de la logique asynchrone QDI [1], la porte de Muller. Des exemples de codage SystemVerilog utilisant les librairies de Tiempo de modules simples et utilisés dans le processeur de réveil seront présentés.

Il existe deux types de circuits, les circuits dit synchrones qui sont séquencés par un signal périodique global appelé horloge et les circuits dit asynchrones qui ne possèdent pas de signal de synchronisation global mais qui utilisent plutôt des synchronisations locales entre opérateurs ce qui implique un comportement évènementiel de la logique. La théorie des circuits asynchrones a commencé à être étudiée dans les années 1950 par Muller et Bartky. Dans l'année 1968 Huffman créé la première machine à états asynchrone alors qu'en 1966 Clark [64] montre qu'il est possible de concevoir des machines spécialisées complexes à partir de blocs asynchrones fonctionnels. Enfin, Sutherland publie en 1989 un article sur les *micropipelines* [149] et contribue largement à l'intérêt de la communauté scientifique et industrielle sur la conception de circuits asynchrones.

Depuis, plusieurs circuits en logique asynchrone ont été développés dans le domaine du commerce ou dans la littérature. On peut citer Amulet 3i [80], MiniMIPS [115], Aspro [164],

1. Quasi Delay Insensitive

80C51 de NXP [162], TAM16 de Tiempo [24], ou encore récemment le neuro-processeur d'IBM [13].

6.1.1 Concepts de base des circuits asynchrones

Les circuits intégrés synchrones numériques se basent sur deux hypothèses : les signaux manipulés sont binaires et le temps est discrétisé. La binarisation permet une implémentation électrique simple grâce à l'algèbre de Boole, et la discrétisation du temps grâce à une horloge permet de s'affranchir des boucles combinatoires, ainsi que des transitions électriques. Les circuits intégrés asynchrones ne se basent pas sur la discrétisation du temps à un pas contraint par le pire cas de traitement de données mais sur des synchronisations locales ce qui permet à des évènements d'être traités dès leur arrivée à la vitesse maximale de l'opérateur qui les traite.

En asynchrone les niveaux des signaux ne sont plus échantillonnés, ce sont leurs transitions qui sont détectées. L'information est donc codée dans les transitions qui permettent de synchroniser le fonctionnement du circuit.

6.1.1.1 Synchronisation locale entre les modules de contrôle

Comme présenté précédemment, le transfert de données entre deux opérateurs asynchrones sont basés sur une synchronisation locale. Chaque opérateur doit donc remplir les fonctions suivantes :

1. écouter les communications entrantes,
2. signaler aux opérateurs émetteurs que les données ont bien été reçues,
3. réaliser le traitement des données quand toutes les informations sont disponibles,
4. attendre que l'opérateur suivant soit prêt à recevoir les données,
5. produire et transmettre les données à l'opérateur suivant.

Ainsi pour permettre un fonctionnement indépendamment du temps entre deux opérateurs, le contrôle local doit s'effectuer de manière bidirectionnelle. Cette communication est implémentée sous forme de canal et est dite à poignée de mains [2] car celui-ci contient les données/requêtes allant de l'émetteur vers le récepteur et un autre signal d'acquittement allant du récepteur vers l'émetteur pour confirmer la bonne réception des données. Ce signal d'acquittement permet la synchronisation des données entre les opérateurs et la causalité des évènements au niveau local et donc la cohérence fonctionnelle du système dans son ensemble. Ce mécanisme de communication bidirectionnel sous forme de canaux est illustré figure 6.1.

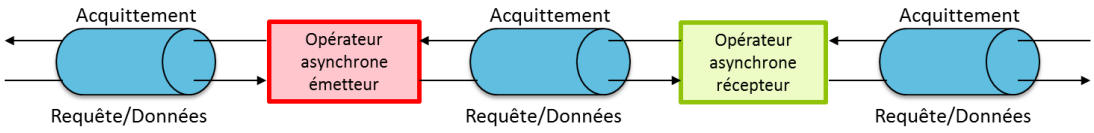


FIGURE 6.1 – Mécanisme de synchronisation locale entre opérateurs asynchrones grâce aux communications de types requêtes/acquittements

- Protocole de communication asynchrone

Ces canaux de communication peuvent être implémentés de deux manières différentes : le protocole deux phases appelé aussi NRZ [3] et le protocole quatre phases appelé aussi

2. Handshake en anglais
3. Non Retour à Zéro

RTZ [4]. Ces protocoles de communication sont illustrés figure 6.2.

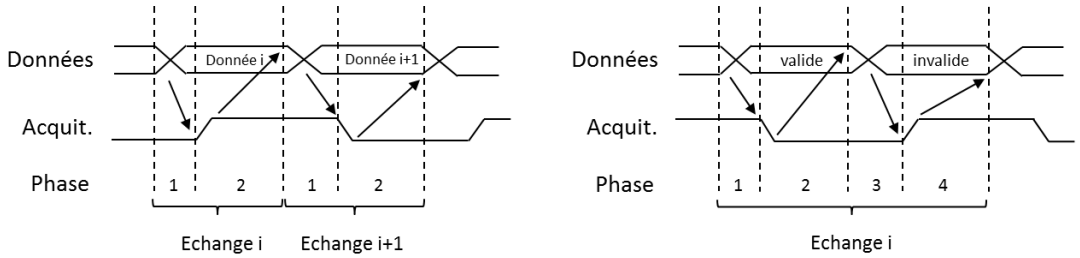


FIGURE 6.2 – Protocole de communication asynchrone 2 et 4 phases

Le protocole deux phases fonctionne de la manière suivante :

— Phase 1 : le récepteur détecte la présence de nouvelles données, effectue le traitement et bascule le signal d'acquittement ;

— Phase 2 : l'émetteur détecte la transition sur le signal d'acquittement et émet de nouvelles données si disponibles.

Pour faciliter l'implémentation, le protocole quatre phases ajoute une phase d'invalidation sur les données et sur l'acquittement pour retourner toujours dans un état de référence. Ainsi pour le protocole quatre phases, les phases suivantes sont observées :

— Phase 1 : Le signal d'acquittement généré par le récepteur est dans sont état initial, l'émetteur génère des données et le récepteur génère une transition sur le signal d'acquittement ;

— Phase 2 : l'émetteur détecte l'acquittement et invalide les données (retour à zéro) ;

— Phase 3 : le récepteur détecte que les données sont passées dans un état invalide et place le signal d'acquittement dans son état initial ;

— Phase 4 : l'émetteur détecte que l'acquittement est dans son état initial et est autorisé à émettre une nouvelle donnée si disponible.

Le protocole deux phases requière un matériel souvent plus important que le protocole quatre phases car il doit détecter des transitions pures et non des transitions entre un état connu et un état de données valides. Ainsi le protocole quatre phases est majoritairement utilisé dans la conception de circuits asynchrones et va être utilisé dans cette thèse.

• Codage des données

La détection des données au seins du protocole réside dans le codage des données elles-mêmes. Afin de détecter la valeur d'un bit, un seul fil de signal ne suffit pas sinon deux mêmes valeurs consécutives de ce bit ne seraient pas détectées. C'est pourquoi il existe deux types de solutions pour détecter l'arrivée d'une nouvelle donnée : la création d'un signal de requête associé aux données encodé en binaire normal (appelé codage de données groupées [5] ou l'utilisation d'un codage insensible aux délais bifilaire ou double-rail [6] par bit de donnée. L'encodage par données groupées rajoute par contre une hypothèse temporelle [149]. Il existe deux types de codage insensibles aux délais, le codage à 3 états et le codage à 4 états comme présentés figure 6.3.

Dans le codage trois états, un fil encode la valeur '0' et l'autre fil encode la valeur '1'. Lorsque les deux fils sont à zéro, cela encode l'état invalide. Cet encodage est par conséquent utilisé pour le protocole quatre phases. L'état 11 est interdit, ainsi chaque changement de valeur implique un passage par l'état invalide 00.

4. Retour à Zéro
5. Bundled Data
6. dual rail

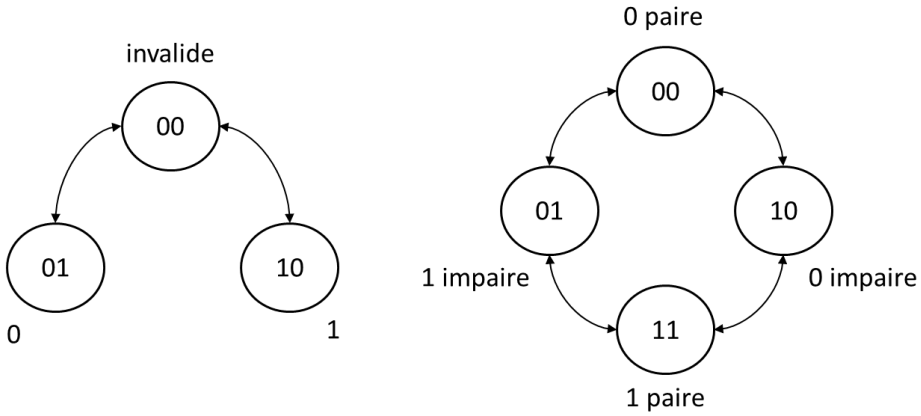


FIGURE 6.3 – Codage 3 états et 4 états double-rail ainsi que leur diagramme de transition

Le codage quatre états code la donnée avec un état pair et un état impair. Á chaque donnée émise, la parité est changée. Ce codage permet de ne pas repasser par l'état invalide, ce qui est parfaitement adapté au protocole deux phases.

Plusieurs bits peuvent être encodés dans un canal, c'est ce qui est appelé le codage *multi-rail m-of-n*. Le codage le plus intéressant en terme de réduction de la consommation est l'encodage 1-of-4 (en one-hot encoding) dont l'encodage est présenté ci-dessous car il lui suffit que deux transitions pour deux bits d'information contrairement à quatre transitions pour deux bits d'information en double-rail [40].

- 0000 = invalide
- 0001 = '00'
- 0010 = '01'
- 0100 = '10'
- 1000 = '11'

Un canal asynchrone peut aussi transporter une information purement évènementielle encodée avec juste un fil de donnée et un fil d'acquittement. Ce type d'évènement a été présenté précédemment dans la section 5.5.5 figure 5.8 avec le signal Continue qui transporte juste un évènement pour permettre le calcul du nouveau PC par exemple. Les outils de Tiempo définissent ce genre de signal un *SREVENT* pour *Single-Rail Event*.

6.1.1.2 Caractéristiques des opérateurs asynchrones

Les opérateurs asynchrones étant indépendants les uns des autres et fonctionnant à leur vitesse maximale sans être cadencés par une horloge globale calée sur le chemin combinatoire le plus long, ceux-ci ont une sémantique de synchronisation plus riche que les opérateurs synchrone. Ils sont caractérisés par quatre paramètres : leur latence, leurs temps de cycle, leur profondeur de pipeline et leur protocole de communication.

- Latence

Le temps de latence correspond à la chaine combinatoire la plus longue pour qu'une sortie soit l'image fonctionnelle d'une entrée comme présenté figure 6.4. Le temps de traversée peut être variable en fonction des données d'entrée et dépend de l'algorithme et de l'implémentation. La latence est indépendante du fait que l'opérateur soit combinatoire ou à mémoire. En asynchrone, un opérateur de mémorisation se comporte comme un opérateur combinatoire, la donnée ne fait que la traverser. C'est pourquoi la vitesse maximale d'un opérateur est mesurée à vide, lorsque toutes les ressources de mémorisation sont inoccu-

pées. Comme montré figure 6.4, les chaines combinatoires peuvent être différentes suivant les données d'entrées, ce qui engendre des latences différentes. Ainsi, des latences minimales, moyennes et maximales peuvent être déterminées pour un opérateur asynchrone.

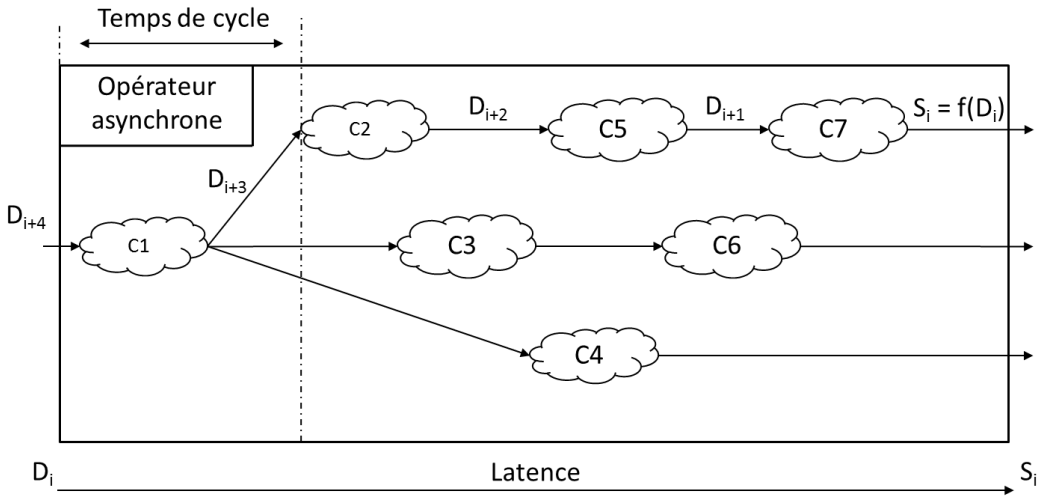


FIGURE 6.4 – Définition de la latence, temps de cycle, profondeur de pipeline d'un opérateur asynchrone

- Temps de cycle

Le temps de cycle correspond au temps minimum qui sépare l'acceptation de deux informations en entrée de l'opérateur. Ainsi dans l'exemple de la figure 6.4, si sur le chemin qui sépare chaque chaine combinatoire, un opérateur de mémorisation est présent, alors le temps de cycle est égal au temps de traversé de la logique C1 et de l'élément de mémorisation. En général cela correspond au temps pour échanger une donnée entre deux ressources de mémorisation successives et caractérise la bande passante de l'opérateur. Le temps de cycle vu de l'entrée de l'opérateur peut varier et est au minimum égal au temps de cycle de l'étage d'entrée et au maximum égal au temps cycle de l'étage le plus lent de l'opérateur. Le temps de cycle dépend de l'occupation des ressources interne, car s'il est utilisé de temps en temps, le temps de cycle est égal au temps de cycle de l'entrée et s'il est utilisé en continu, le temps de cycle est égal au temps de cycle le plus lent en interne.

- Profondeur de pipeline

La profondeur de pipeline d'un opérateur asynchrone caractérise le nombre maximal de données que l'opérateur peut mémoriser. Cela correspond à la différence maximale entre l'indice de la donnée présente en sortie et l'indice de la donnée présente à l'entrée lorsque toutes les ressources de mémorisation sont occupées. Dans la figure 6.4, si sur chaque chemin entre deux opérateurs combinatoires il y a un élément de mémorisation, alors la profondeur de pipeline est de 4. Dans un opérateur asynchrone, contrairement au synchrone, le nombre de données présentes dans le pipeline n'est pas imposé et varie dynamiquement au cours de l'exécution.

- Protocole de communications

Les protocoles de communications caractérisent les interfaces des opérateurs asynchrones afin qu'ils puissent se synchroniser entre eux et échanger des données. Plusieurs protocoles peuvent être utilisés dans un même circuit. Il faut néanmoins qu'ils assurent la détection des données en entrée, la signalisation comme quoi la donnée a été consommée en entrée et qu'une information est disponible en sortie. Une telle implémentation de circuit est qualifiée de flot de données.

6.1.2 Circuits asynchrones quasi-insensibles aux délais (QDI)

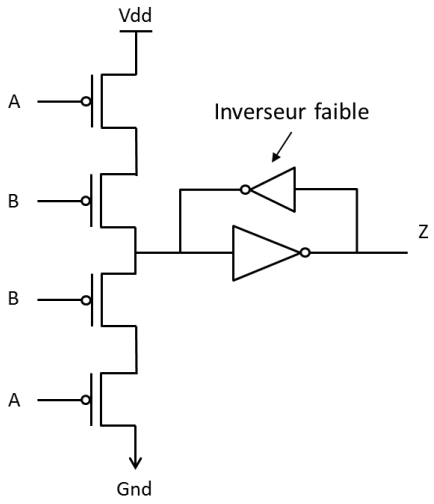
Le domaine de conception de circuit en logique asynchrone a fait naître aussi de nouveaux outils pour faciliter la description des architectures asynchrones et la synthèse de celles-ci. Parmi eux on peut citer le langage CHP [7] qui a été développé à Caltech et utilisé pour de nombreux circuits tel que Aspro [164]. Dans le cadre de cette thèse l'outil de synthèse ACC [8] développé par Tiempo [9] a été utilisé. Les outils de Tiempo permettent de développer les différents modules de l'architecture en langage standardisé SystemVerilog dont des exemples de description seront vus section 6.1.4. L'outil ACC permet de synthétiser en logique QDI [10] WCHB [11] la description en SystemVerilog du matériel. La classe des circuits quasi insensibles aux délais est basée sur un modèle de délais de propagation et d'interconnexion non-bornés mais ajoute la notion de fourche isochrone [12]. Une fourche est par définition un fil qui connecte un émetteur à plusieurs récepteurs. Cette fourche est isochrone lorsque le délais entre l'émetteur et les différents récepteurs est identique. Alain Martin a montré que cette hypothèse est la plus faible à ajouter aux circuits insensibles aux délais pour les concevoir avec des portes logiques standards [116]. En pratique il suffit de vérifier que le temps de propagation dans un fil d'une fourche ne soit pas supérieur au temps de propagation dans une autre branche additionné du temps de traversé de l'opérateur auquel ce signal est connecté. Le protocole WCHB synchronise la phase de remise à zéro des deux protocoles. Si l'entrée est redescendue et que la sortie est acquittée, la sortie est remise à zéro et l'acquittement est repositionné à 1 pour la donnée suivante.

Dans cette section l'une des portes logique les plus utilisée dans les circuits asynchrone QDI, la porte de Muller (section 6.1.2.1) va être présentée, ainsi qu'un exemple de propagation d'évènements dans un pipeline asynchrone.

6.1.2.1 Porte de Muller

La logique asynchrone met en place des rendez-vous d'évènements en entrée d'opérateurs de traitements. Ceux ci permettent d'être sûr que tous les signaux nécessaires à produire une sortie sont bien présents. C'est le principe de la synchronisation locale entre opérateurs. La porte de Muller introduit dans les années 1950 par David E. Muller implémente ce rendez-vous d'évènements. La figure 6.5 présente le schéma transistor de la porte de Muller à deux entrées, sa table de vérité et son symbole. D'après cette figure, il est possible de voir que cette porte transmet la valeur de ses entrées uniquement quand les deux sont égales, sinon la sortie conserve sa valeur courante. Cette porte est constituée de 4 transistors en série (deux PMOS et deux NMOS) de telle manière que la sortie soit en haute impédance si les entrées ne sont pas identiques. Une cellule mémorisante est placée en sortie pour maintenir la valeur de la sortie à la valeur courante. Dans le cadre de la synthèse de circuit asynchrone en technologie FDSOI 28nm, STMicroelectronics a développé une bibliothèque de portes asynchrones qui peuvent directement être utilisées par le synthétiseur ACC.

7. Concurrent Hardware Process
8. Asynchronous Circuit Compiler
9. http ://www.tiempo-secure.com/
10. Quasi Delay Insensitive circuits
11. Weak Condition Half Buffer
12. isochronic fork



A	B	Z
0	0	0
0	1	Z^{-1}
1	0	Z^{-1}
1	1	1

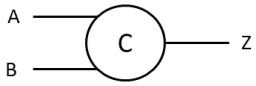


FIGURE 6.5 – Porte de Muller à deux entrées : schéma transistor, table de vérité et symbol

6.1.2.2 Exemple de propagation d'évènements dans un pipeline asynchrone et exemple de porte logique en logique QDI

En prenant la structure du *Half-Buffer* qui est très utilisée dans les architectures asynchrones QDI, nous allons voir comment un évènement se propage dans la logique. La figure 6.6 présente la mise en série de trois *Half-Buffer* ainsi que le diagramme en fonction du temps de l'activité aux différents étages du pipeline pour les données et les acquittements. Au début tous les signaux sont dans leur état invalide du protocole quatre phases. Une donnée appelée jeton arrive sur l'entrée et se propage dans les *Half-Buffer*. Lorsque deux *Half-Buffer* sont mis en série, cela s'appelle un *Full-Buffer* capable de contenir un jeton en son sein pour ainsi mémoriser la valeur du jeton et avoir les signaux d'interface à l'état invalide. On voit sur cette figure que lorsque le pipeline est vide initialement, il faut à l'entrée un temps minimale de $3t_c + 2t_{nor}$ pour que le protocole revienne à son état invalide et accepte une nouvelle donnée, ceci due à la logique de retour de l'acquittement.

6.1.3 Avantages et inconvénients de la logique asynchrone

6.1.3.1 Pipeline élastique

Le pipeline, que ce soit en logique synchrone ou asynchrone permet d'effectuer plusieurs tâches en parallèle. L'avantage en logique asynchrone, c'est que celui-ci a une capacité variable grâce aux mécanismes de synchronisation locale. Ainsi les données peuvent se déplacer dans le pipeline aussi loin qu'elles ne rencontrent pas de ressources occupées et ne sont pas forcément séparées par le même nombre d'étage au court du temps.

6.1.3.2 Circuit adapté à des évènements non déterministes

Les évènements non déterministes peuvent faire référence aux interruptions dans un processeur par exemple. En synchrone il est nécessaire d'échantillonner le signal d'interruption et le resynchroniser avec l'horloge. Ici le signal d'interruption est considéré comme un évènement normal et est traité directement par la logique QDI, ce qui est un avantage certain pour notre application qui exécute du code uniquement sur interruptions. De plus les temps de réveil sont instantanés en logique asynchrone car aucune horloge n'est à

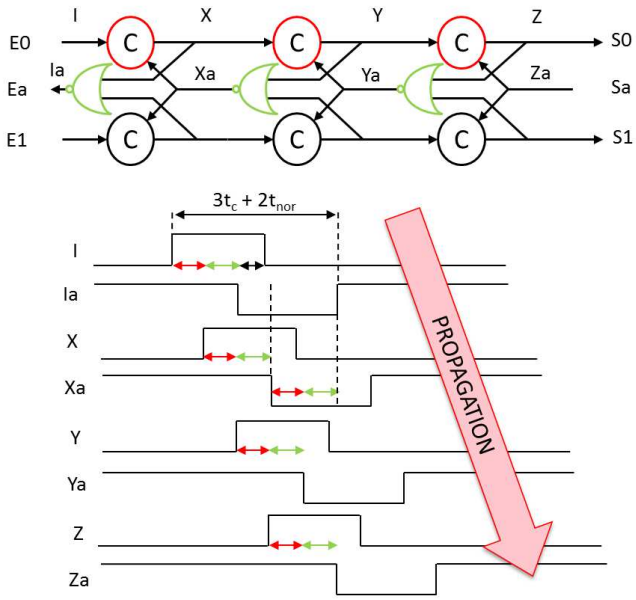


FIGURE 6.6 – Exemple de propagation d'un signal dans un pipeline de simple buffer QDI WCHB

réveiller et la mise en veille s'effectue de manière intrinsèque par la logique.

6.1.3.3 Calcul en temps minimum et temps moyen

Nous avons pu voir qu'un opérateur pouvait avoir une latence variable (figure 6.4) suivant les données d'entrée et le chemin utilisé. Cette latence est comprise entre une borne minimum et une borne supérieure. En asynchrone le calcul s'effectue au plus tôt suivant la fonction à évaluer, les données et le chemin parcouru, en d'autre terme il fonctionne à la vitesse maximale permise par le dispositif. Contrairement au synchrone, les performances globales ne sont pas contraintes par le pire cas d'un bloc. Il faut alors optimiser les temps de calcul moyen des blocs en moyenne les plus utilisés.

6.1.3.4 Absence d'horloge

L'avantage considérable des circuits asynchrones est de ne pas avoir d'horloge globale. L'arbre d'horloge dans un circuit synchrone représente une grande partie de la consommation. En asynchrone les éléments de synchronisation sont distribués dans l'ensemble du circuit et leur conception est nettement plus facile. Il n'y a plus de problèmes de temps de *skew* ou temps de *hold* à respecter, le temps d'arrivée d'un signal d'un bout à l'autre du circuit n'a pas d'importance, il faut juste respecter le principe de fourche isochrone.

6.1.3.5 Consommation évènementielle et lisse

Le fonctionnement évènementiel des circuits asynchrones impacte sa consommation. En effet, l'absence d'horloge supprime la consommation due aux transitions dans les bascules et les chargements de l'arbre d'horloge. Les aléas de la logique combinatoires sont supprimés en asynchrone et représentent une consommation non négligeable en synchrone.

La consommation dynamique du circuit asynchrone varie proportionnellement au nombre d'évènements à traiter. De plus les transitions apparaissent uniquement quand des données sont à traiter, pas comme en synchrone ou les blocs commutent tous au signal d'horloge

même si cela est inutile. Ceci engendre un *clock gating* naturel de la logique asynchrone à tous les niveaux de granularité lorsqu'il n'y a rien à traiter.

Grâce au fonctionnement évènementiel, la consommation des opérateurs est répartie sur le temps au fur et à mesure que les évènements se propagent dans la logique. De plus, il n'y a pas de pic de courant à chaque coup d'horloge comme en synchrone, ce qui implique une consommation en courant nettement plus lisse que pour un circuit synchrone. Ceci implique par la même occasion une réduction des émissions électromagnétiques du circuit et est donc moins sensible à des attaques de type étude des émissions électromagnétiques du circuit dans le domaine de la sécurité.

6.1.3.6 Robustesse des circuits asynchrones et souplesses d'alimentation

L'un des grands avantages de la logique asynchrone est sa robustesse aux variations de tension et de température. En effet la logique asynchrone est correcte par construction et ainsi, en réduisant la tension d'alimentation, on augmente les temps de propagation dans la logique mais cela n'impacte pas le fonctionnement du circuit. Ainsi en asynchrone il est aisé de diminuer la tension d'alimentation pour diminuer sa consommation de puissance tout en perdant d'un autre côté des performances.

6.1.3.7 Augmentation de la surface du circuit

L'un des problèmes majeurs de la logique asynchrone est par contre l'augmentation de la surface du circuit. En effet, par rapport à un équivalent synchrone, la surface du circuit asynchrone peut être doublée, ce qui augmente la consommation statique du circuit. Mais heureusement, comme nous l'avons vu, il est très aisé de pouvoir diminuer les tensions d'alimentation du circuit pour réduire la consommation de puissance tout en garantissant son fonctionnement. C'est pourquoi dans notre cas, durant les longues périodes de veille il sera nécessaire de diminuer la tension de la logique asynchrone à sa tension minimale de rétention et de l'élever lorsqu'un évènement est à traiter. Il peut être aussi envisageable de couper l'alimentation d'une partie de la logique asynchrone pour réduire encore plus la consommation statique pendant ces longues phases de veille.

6.1.4 Exemples de description de modules asynchrones en SystemVerilog avec les outils de Tiempo

Cette section a pour but de présenter la manière de décrire des modules en SystemVerilog grâce aux outils de Tiempo. La description de ces modules sont basées sur des méthodes de lecture et d'écriture sur les canaux asynchrones pour obtenir une synchronisation locale entre les modules. L'annexe B présente des structures asynchrones très utilisées dans le WUC. L'exemple d'un banc de registres avec uniquement quatre registres va être détaillé dans cette section.

Il faut dans un premier temps décrire les ports d'entrées/sorties du module à l'aide de canaux *push_channel_bitx* comme montré ci-dessous.

```
1 module reg_file_chan
2 (
3   /***********Data in ***************/
4   //Register Number
5   push_channel_bit2.in     Op1_i,
6   push_channel_bit2.in     Op2_i,
7   push_channel_bit2.in     Dest_i,
```

```
8    //Unit Data Write
9    push_channel_bit32.in    Units_i,
10
11   /***********Data out ***************/
12   //Bus 1 and 2 Read
13   push_channel_bit32.out   Bus1_o,
14   push_channel_bit32.out   Bus2_o,
15
16   (*ACC_Reset*) input bit    resetn
17 );
```

Ensuite les différents processus se déclarent toujours entre un *always begin* et *end*. Ces processus ouvrent des canaux grâce à la méthode *BeginRead()*, effectuent un travail sur les données entrantes pour écrire sur d'autres canaux grâce à la méthode *Write()* et referment les canaux ouverts avec la méthode *EndRead()*. L'exemple ci-dessous montre le processus de lecture dans le banc de registres pour écrire l'opérande 1 sur le bus 1.

```
1  always
2    begin: Read_reg_1
3    bit2   op1;
4    bit32  data;
5
6    Op1_i.BeginRead(op1);
7    unique case(op1)
8      `R0:begin
9        r0.BeginRead(data);
10       Bus1_o.Write(data);
11       r0.EndRead();
12     end
13     `R1:begin
14       r1.BeginRead(data);
15       Bus1_o.Write(data);
16       r1.EndRead();
17     end
18     `R2: begin
19       r2.BeginRead(data);
20       Bus1_o.Write(data);
21       r2.EndRead();
22     end
23     `R3: begin
24       r3.BeginRead(data);
25       Bus1_o.Write(data);
26       r3.EndRead();
27     end
28   endcase
29   Op1_i.EndRead();
30 end
```

Chaque processus est ainsi déclaré de cette manière et les canaux de lecture se ferment uniquement quand les écritures sur les canaux à l'intérieur du processus ont été acquittées.

Les registres sont décrits à l'aide de *Shared Variable* et sont synthétisés avec des cel-

lules de registre. Différents types de conflits peuvent être gérés par la *Shared Variable* : aucun conflit, lecture-écriture, écriture-écriture, tous les conflits. Voici comment déclarer un registre en asynchrone :

```
1 shared_variable_bit32 #(.INITIAL_VALUE(32'h00000000), .
    ACCESS_CONFLICT(E_READ_WRITE_CONFLICT_C)) r0 ();
```

6.2 Architecture de la mémoire de programme et données

6.2.1 Caractéristiques de la mémoire

Le WUC possède une seule mémoire pour les données et le programme avec un seul port de lecture/écriture. Pour la première version du WUC il n'a pas été prévu d'implémenter une mémoire asynchrone, c'est pourquoi la réalisation d'une interface entre la logique asynchrone et synchrone est nécessaire et présentée section 6.2.3. La mémoire utilisée pour le WUC est une mémoire créée par une équipe du laboratoire et caractérisée pour une tension de 0,5V avec 12 ns de temps d'accès. La capacité d'un bloc mémoire est de 4KB (1024 mots de 32 bits) et le WUC en instancie par conséquent qu'un seul étant donné que cette capacité suffit pour les applications visées. Cette mémoire n'a pas de capacité de *Back Biasing* mais sa tension peut théoriquement varier de 0,5V à 1V.

La figure 6.7 illustre l'interface de la mémoire ainsi que l'organisation de la macro mémoire. Celle-ci possède les signaux classiques d'une mémoire : un signal d'adresse, de masque pour écrire ou pas un octet sur une ligne, un signal de lecture/écriture, un *Chip Select* utile au cas ou plusieurs macro mémoires sont présentes, un signal de donnée à écrire et un signal de donnée lue ainsi qu'un reset. La macro mémoire est divisée en quatre bancs possédant des IOs partagées entre deux bancs, des drivers pour les *Word Line*, un contrôleur d'IO et enfin un contrôleur pour verrouiller les signaux d'entrée et s'occuper du clock gating interne. Les lectures et écritures s'effectuent sur 32 bits.

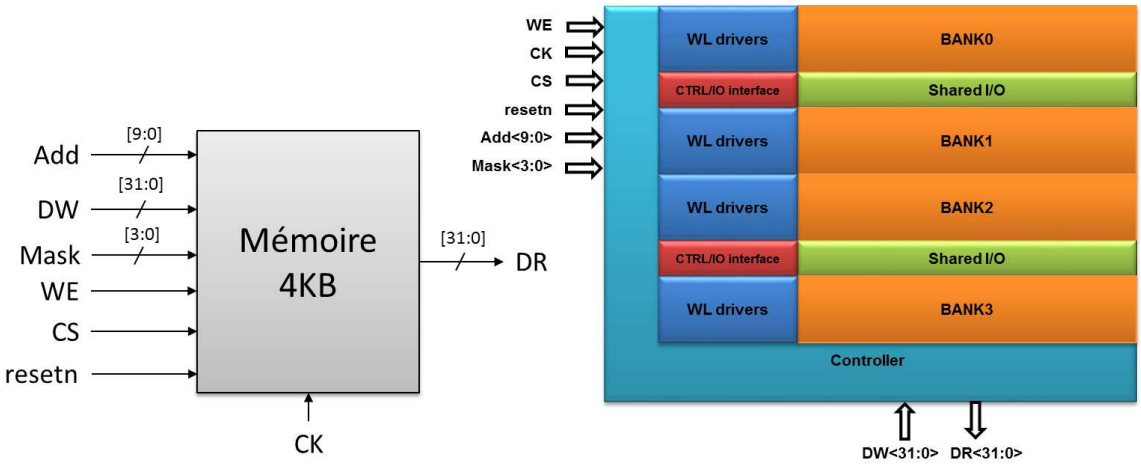


FIGURE 6.7 – Interface et architecture interne de la mémoire contenant le programme et les données utilisées pour la plateforme de réveil

6.2.2 Protocole d'accès à la mémoire

La figure 6.8 montre l'évolution des signaux de l'interface mémoire en fonction du temps pour une écriture et une lecture. Dans un premier temps, le *Chip Select* (CS) est placé au

niveau haut pour sélectionner cette macro mémoire, puis le signal d'écriture/lecture (WE) est placé à 1 pour une écriture et 0 pour une lecture, ainsi que les signaux de masquage, d'adresse et donnée à écrire (DW) dans le cas d'une écriture. Le signal de masquage est sur 4 bits et indique si un octet peut être écrit ou pas (1 implique que l'octet n'est pas écrit et 0 qu'il est écrit). Il faut alors respecter un temps de *setup* avant d'avoir un front montant sur l'horloge (CK) et que les entrées soient maintenues pendant un temps de *hold* avant de les basculer. Un temps entre deux accès doit aussi être respecté.

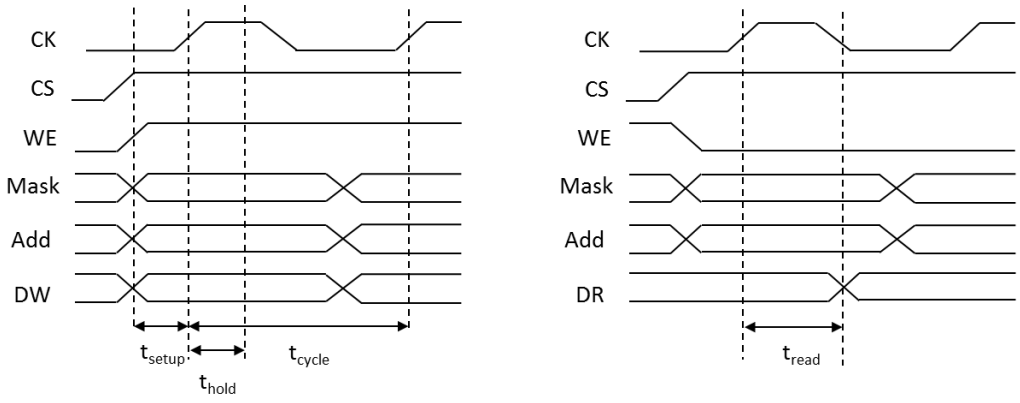


FIGURE 6.8 – Protocole d'accès à la mémoire en écriture et lecture

6.2.3 Architecture de l'interface mémoire

Chaque bloc asynchrone connecté au bus de communication (dont l'architecture sera présentée figure 6.3.5) possède les canaux de communication suivant en dual-rail :

— adresse sur 16 bits,
— taille de la donnée sur 2 bits,
— signal de lecture écriture sur 1 bit,
— signal de donnée à écrire sur 32 bits,
— signal de donnée lu sur 32 bits.

La macro mémoire possède quant à elle les signaux d'interface vus dans la section précédente et encodés en simple rail. Il faut de plus respecter les contraintes en timing (temps de *hold*, *setup*, accès en lecture/écriture et temps de cycle). L'interface doit aussi générer le signal d'horloge localement lorsque des données arrivent sur les canaux asynchrones, ceci pour activer la mémoire lorsque nécessaire afin de réduire la consommation.

L'interface asynchrone-synchrone qui a été développée pour la mémoire est présentée figure 6.9. Elle est séparée en deux parties, une partie asynchrone utilisant les outils de Tiempo pour transformer des canaux de communication en signaux simple rail de type *Bundled Data*, et une partie synchrone implémentant toute la logique des délais de la mémoire, la génération du signal d'horloge et la génération des acquittements de chaque canal asynchrone. Les modules *Push_bitx_2hsk* transforment les canaux double rail en *Bundled Data* alors que le module *Sig_2push* transforme un signal *Bundled Data* en double rail.

Pour une écriture il y a un rendez-vous entre les requêtes du masque, de l'adresse, de la donnée à écrire et du signal de demande d'écriture (WE). Cela génère une requête d'écriture qui passe dans une chaine de délais (Delay W) dont l'architecture va être présentée dans la prochaine section. Le signal de sortie de cette chaine de délais, génère l'acquittement pour le masque, l'adresse et la donnée écrite. Pour la lecture, le rendez-vous s'effectue entre la

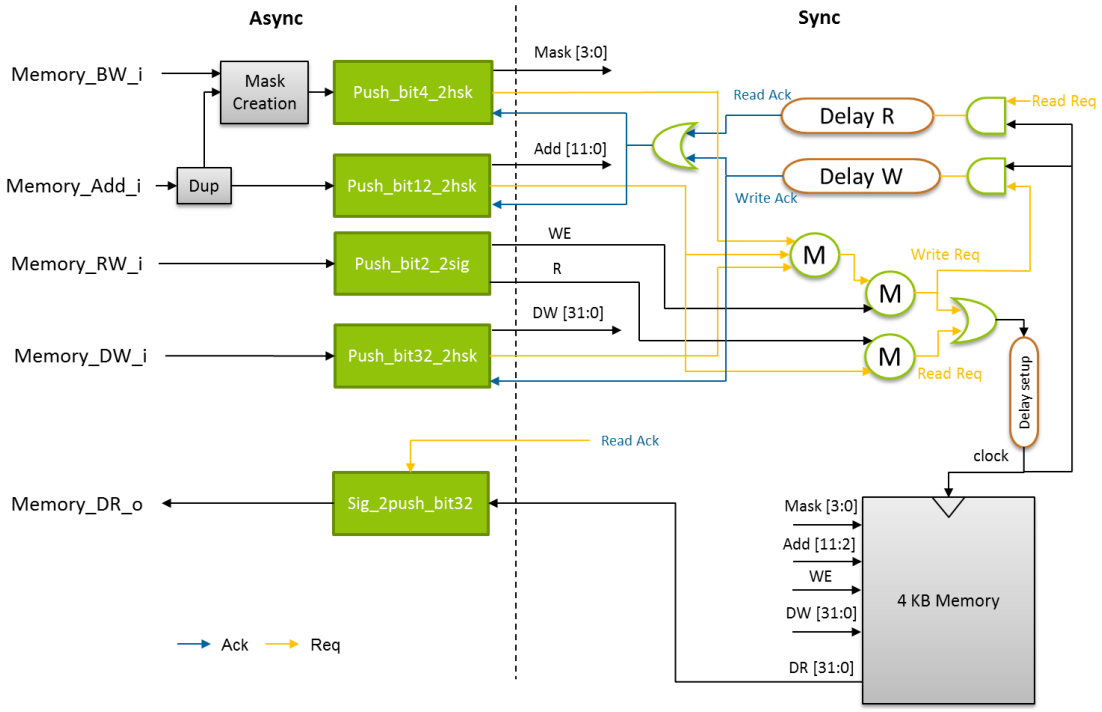


FIGURE 6.9 – Interface asynchrone-synchrone de la mémoire

requête d'adresse et le signal de demande de lecture (R). Cette requête de lecture passe dans une chaine de lecture (Delay R) et génère en sortie l'acquittement pour le masque, l'adresse et devient une requête pour que la donnée positionnée en sortie de la mémoire sur le signal DR soit prise en compte par le module *Sig_2push_bit32* pour générer un jeton sur le canal Memory_DR_o.

La figure 6.10 présente le diagramme en fonction du temps de l'activité sur les canaux de communication pour une lecture et montre l'enchainement du basculement des signaux de requête et d'acquittement en fonction de la validité des canaux. En effet lorsque les trois canaux d'entrée sont valides en entrée, Read_Req monte à 1. Après un temps t_{read}, Read_Ack passe à 1, ce qui implique un jeton sur Memory_DR_o. Read_Ack permet d'invalider toute les entrées, ce qui fait baisser la requête Read_Req et Read_Ack par conséquent. Memory_DR_o est aussi consommé par le bus ce qui implique son invalidation.

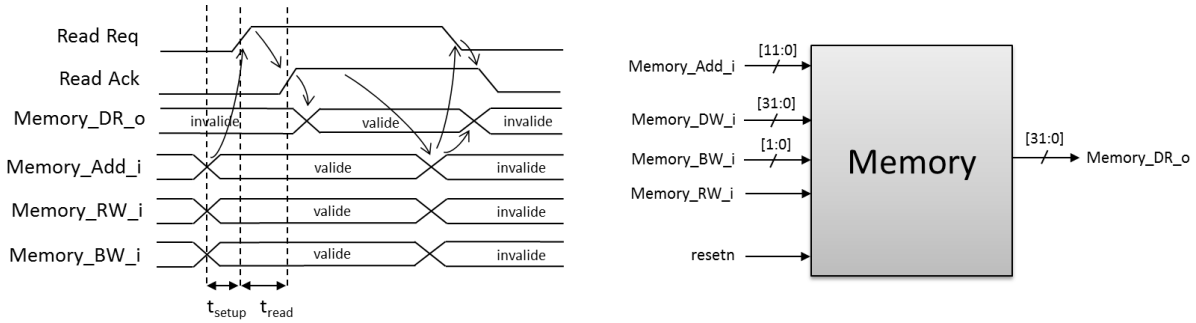


FIGURE 6.10 – Protocole de l'interface asynchrone synchrone de la mémoire pour une lecture et entrées/sortie du bloc mémoire connecté au bus de communication

6.2.4 Architecture des chaines de délai

L'interface mémoire intègre des chaines de délai afin de respecter les timing de la mémoire et générer les acquittements. Les chaines de délai implémentées sont des chaines de délai dites asymétriques car la remise à zéro de la sortie est plus rapide que la montée à 1. Ceci pour avoir uniquement un délai dans la phase montante du calcul et non lors de la remise à zéro du protocole. La figure 6.11 illustre bien le phénomène. Lorsque la sortie de l'arbre de Muller est à 1 (à l'état inactif) et qu'un signal en entrée se lève à 1, celui-ci va se propager dans tous les buffers de délai D et les portes ET au fur et à mesure, ceci grâce aux portes ET avec le signal E_0. Lorsque le 1 est à la sortie de la chaine de délai alors la sortie de l'arbre de Muller est à 0 grâce à tous les inverseurs. Comme il a été vu, la sortie de la chaine de délai est en fait l'acquittement des canaux asynchrones ce qui a pour effet de baisser la requête à 0 en entrée de la chaine. Comme la sortie de l'arbre de Muller est à 0 alors la Muller d'entrée permet au 0 de se propager dans toutes les chaines de délais D élémentaire en même temps grâce aux portes ET. Á ce moment là l'entrée IN ne prend pas en compte de nouvelles requêtes tant que la sortie de l'arbre de Muller n'est pas revenu à 1. Ceci assure bien le respect de la logique QDI, en effet l'arbre de Muller vérifie bien que la requête précédente est redescendue (que tous les 0 se sont propagés dans tous les buffers de délais pour revenir à un état stable) avant d'en autoriser une autre.

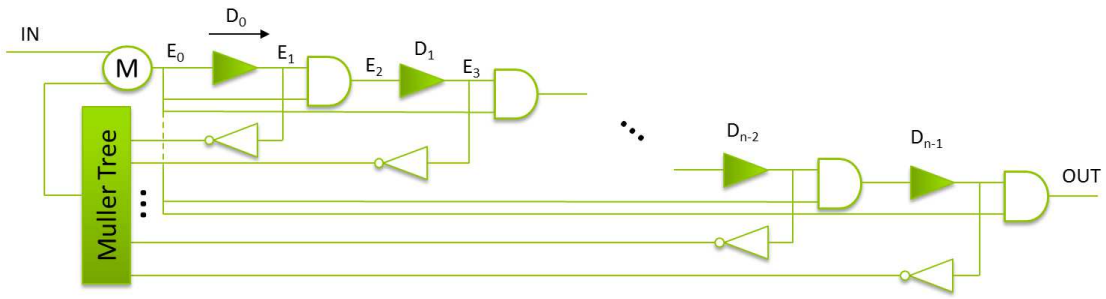


FIGURE 6.11 – Chaine de délai asymétrique avec remise à zéro rapide

Dans le cadre du circuit, les chaines de délai ont été rendues configurables afin d'adapter finement le timing de chacune des phases $t_{setup}, t_{read}, t_{write}, t_{cycle}$. Ces chaines de délai sont configurables via une interface registre et intégrées dans les drivers du circuit pour avoir une configuration depuis l'application.

6.3 Microrchitecture des modules internes

Durant la description des différentes unités, la microarchitecture sera représentée par de petits blocs qui représenteront chacun un processus ouvrant des canaux de communication (grâce à la méthode BeginRead() des canaux push_channel vu dans la section 6.1.4) et effectuant certain rendez-vous entre eux (grâce aux fork-join), écrivant sur certains autres canaux (grâce à la méthode Write()) et refermant la communication des canaux ouverts pour signifier la fin de la synchronisation local entre deux processus (grâce à la méthode EndRead() des canaux push_channel). Nous utilisons dans la représentation de la microarchitecture le nom *Split* pour désigner un demultiplexeur et *Merge* pour désigner le module au sens asynchrone ou il est en attente d'un jeton sur un canal de ces entrées pour l'écrire sur la sortie ou bien un multiplexeur qui est en attente de plusieurs entrées mais aussi d'un signal de contrôle pour rediriger une des entrées sur la sortie. De plus,

chaque flèche présente dans la représentation de la microarchitecture décrit un canal de communication asynchrone QDI (donnée + acquittement).

6.3.1 Contrôleur d'interruptions

Le contrôleur d'interruption est une partie centrale du WUC car c'est lui qui permet l'exécution du code en fonction des évènements et qui trie les évènements entrants. Le choix de l'évènement le plus prioritaire est déterminé par son numéro d'interruption par ordre décroissant ainsi l'interruption numéro 0 est plus prioritaire que l'interruption numéro 3 par exemple. De plus, comme expliqué dans les spécifications, les interruptions sont non préemptibles : lorsqu'une interruption commence à s'exécuter, elle s'exécute jusqu'à complétion.

Il a été choisi d'intégrer 8 interruptions générales au WUC pour le circuit de cette thèse. Les interruptions sont gérées sur niveau par le WUC, c'est à dire que ce ne sont pas de purs évènements SREVENT qui ne contiennent pas de valeur. Un signal d'interruption (IT) est transporté sur un canal double rail possédant une valeur (0 signifie l'interruption n'est pas en attente et 1 l'interruption est en attente). En effet, les périphériques montent à 1 leur signal d'interruption lorsque cela est nécessaire et permis et le maintiennent tant que le matériel ou le programme n'a pas effacé le drapeau d'interruption [13]. De plus, il est possible que le périphérique regénère l'interruption dûe à des évènements internes ou externes alors même que celle-ci n'a pas encore été effacée. Un signal d'IT en SREVENT ne permettrait pas de savoir si l'interruption est montée ou descendue. Il est nécessaire aussi d'avoir une configuration registre pour permettre les interruptions [14], les déclencher pour les mettre en attente [15]. Ainsi, la relation qui lie les interruptions entrantes, le registre des interruptions en attente déclenchées par le programme et le registre de permission des interruptions est :

```
1  irq_in_en_pend_r = (irq_in_r | irq_pend_r) & irq_en_r
```

La figure 6.12 présente l'architecture du contrôleur d'interruptions. Les interruptions pouvant a priori arriver en même temps, elles passent en premier par un mutex pour les rendre mutuellement exclusives, et un processus écrivant dans un registre IRQ_IN_R [16] et générant un évènement pour signaler que le registre a été modifié. Par la même occasion, depuis l'interface registre connectée au bus de communication, il est possible d'écrire dans les registres pour permettre ou non certaines interruptions depuis le registre IRQ_EN_R et déclencher des interruptions au niveau logiciel depuis le registre IRQ_PEND_R. Ces processus d'écriture des registres engendrent aussi des évènements pour signifier qu'ils ont été modifiés. Les trois évènements générés peuvent aussi avoir lieu au même moment, c'est pourquoi ils passent eux aussi dans un mutex pour les rendre mutuellement exclusifs afin que chaque évènement soit pris en compte par le processus de masquage des ITs pour écrire dans un registre qui sauvegarde uniquement les interruptions permises et en attente. Ce processus effectue le masquage vu plus haut et envoie un évènement au processus qui va générer les entrées de l'arbre de comparaison. Ce processus génère sur chaque entrée des comparateurs les vecteurs sur cinq bits suivants :

```
1  tree_in[i] = {irq_in_en_pend_r[i], {1'b0, 3'di}}
2  no_it = {1'b1, {1'b1, 3'd7}}
```

13. Flag Interrupt
14. Enable Interrupt
15. Pending Interrupt
16. Registre de sauvegarde des interruptions entrantes

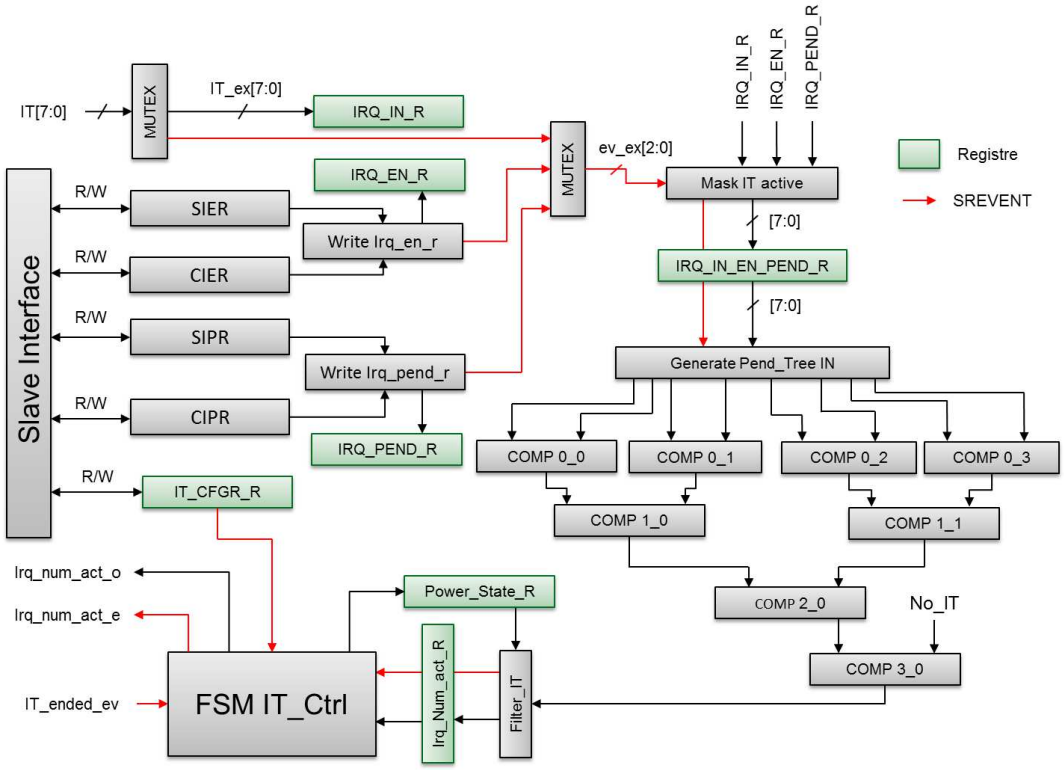


FIGURE 6.12 – Microarchitecture du contrôleur d'interruptions

Ces vecteurs sont une concaténation du numéro d'interruption et de son bit d'interruption d'activité provenant du registre irq_in_en_pend_r. Le comparateur va alors vérifier si l'interruption de l'entrée avec le numéro d'interruption le plus faible est actif ou pas. Son algorithme est le suivant :

```
1 if(in1[4] == 0)
2   out = in2;
3 else
4   out = in1;
```

Ainsi, si l'interruption au numéro le plus faible (in1) est active alors la sortie du comparateur prend son vecteur sinon elle prend le vecteur de la seconde entrée in2. Le bit d'activité du cas ou il n'y a pas d'interruption active est toujours à 1 pour avoir le cas quand aucune interruption n'est active d'avoir effectivement le numéro de non activité fixé à 4'b1111 de sortir de l'arbre de comparaison. Une fois le numéro d'interruption sorti de l'arbre, celui-ci est filtré par un processus afin d'envoyer ou non un évènement à la machine d'états en fonction de l'état d'activité du WUC et pour le sauvegarder dans un registre Irq_num_act_R. Cet évènement est envoyé uniquement si le WUC n'exécute rien et si au moins une IT est active et est autre que le numéro d'interruption *no_it*.

La machine d'état dont l'encodage en SystemVerilog a été présenté en section 6.1.4 est illustrée 6.13. Il a été choisi de mettre l'IT0 comme IT d'amorçage [17] du système. Juste après un *reset*, l'IT0 est en attente et, dès qu'un *Enable* est écrit dans le registre de configuration du contrôleur d'interruption alors, le numéro IT0 est envoyé au décodeur et se met en attente de la fin du code de boot. Une fois fini, la machine d'états se met dans

17. Boot

l'état *SLEEP* et est en attente des ITs entrantes.

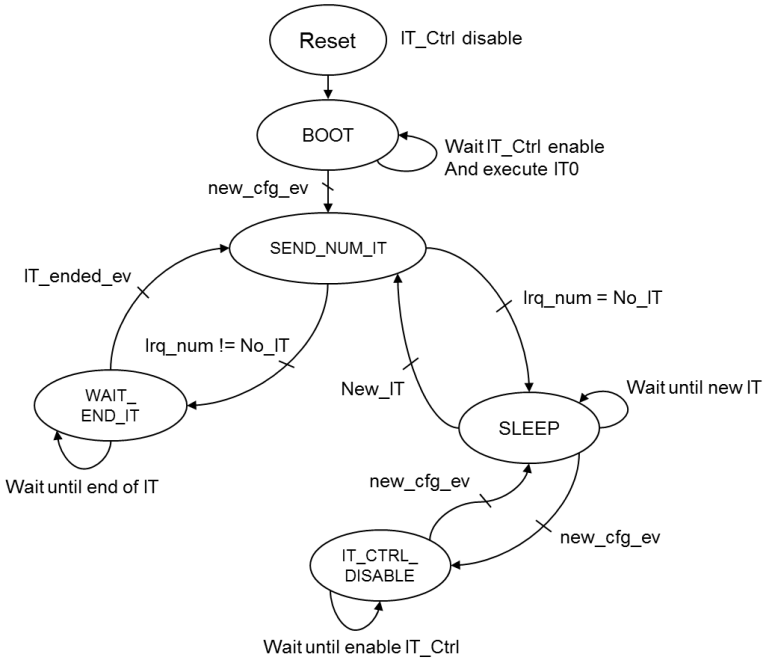


FIGURE 6.13 – Machine d'états du contrôleur d'interruptions

6.3.2 Décodeur

Vu l'encodage présenté tableau 5.7, il est facile de déterminer rapidement quel est le format de l'instruction reçue. C'est pourquoi un pré-décodage de l'instruction est effectué sur, au maximum, les trois premiers bits de l'instruction pour connaître le format Fx de l'instruction. Comme nous avons pu le voir, il y a quatre gros formats d'instructions, non pas séparés par leurs types d'opérandes, mais par le nombre d'instructions à encoder pour ce type d'opérandes. C'est pourquoi les sous formats d'instructions sont regroupés par rapport à la taille de leur opcode. Le pré-décodage sert aussi comme expliqué section 5.5.5 à envoyer les évènements pour le debug et filtrer les instructions spéciales (`BREAK`, `ENDIT`). Le fonctionnement du principe de débogage du WUC, des registres du WUC et de la machine d'états du décodeur est expliqué section 5.5.5 et ne va pas être ré-expliqué dans cette partie. La figure 6.14 montre la microarchitecture du décodeur.

Le pré-décodeur distribue l'instruction au bon processus de décodage de format Fx. Ce sont dans ces processus qu'une séquentialité est mise en place pour envoyer dans un premier temps tous les signaux de contrôle et les différents champs de l'instruction vers l'unité d'exécution concernée. Une fois l'exécution terminée, ce qui se traduit par la fermeture de tous les canaux de communication de contrôle et d'opérandes, le signal de contrôle du type d'instructions qui vient d'être exécuté (`I_type_ctrl`) est envoyé à la PC UNIT pour calculer le nouveau compteur programme. Ce signal est encodé sur 3 bits car il y a 6 possibilités de modification du `PC` qui seront vues dans la prochaine section.

Les signaux générés par les processus de décodage des formats F1, F2, F3 et F4 génèrent les signaux des numéros de registres vers le banc de registres pour les opérandes à partir des registres généraux, les valeurs immédiates, les signaux de contrôle des demultiplexeurs des registres vers les unités d'exécution, et les signaux de commande ayant un *opcode* reconstruit avec des champs directement utilisable par l'unité d'exécution pour contrôler

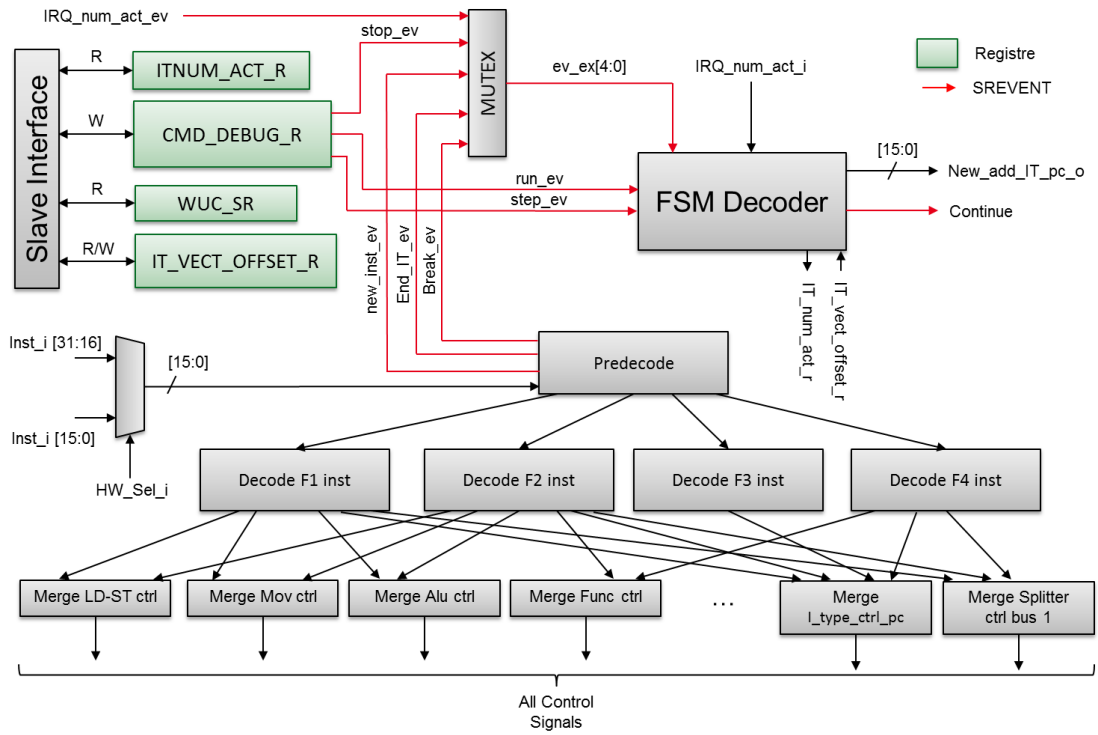


FIGURE 6.14 – Microarchitecture du décodeur d'instructions

leur processus internes. Par exemple les signaux de commande qui sont envoyés à l'ALU et à l'unité *Load-Store* sont :

```
opcode_alu = {unit_op[4:3], inst_op[2:0]}
opcode_ld_st = {ld_or_st[5], bw[4:3], add_mode[2:1], u_or_s[0]}
```

Dans le cas de l'ALU, celle-ci reçoit un opcode composé d'un encodage de l'unité sur 2 bits (unité arithmétique, unité logique, unité d'extension, et unité de décalage) et d'un encodage de l'instruction qui est un ré-encodage des instructions exécutables par chaque unité interne et encodé sur 3 bits. L'unité *Load-Store* reçoit un *opcode* sur 6 bits pour indiquer si l'instruction à exécuter est un *load* ou un *store*, la taille du transfert, le mode d'adressage (immédiat, registre-registre, registre ou PC-relative), et enfin si le transfert est une donnée signée ou non signée dans le cas d'un *load* pour étendre le signe. De cette façon ces champs d'*opcode* sont directement utilisables par l'unité d'exécution pour les propager dans ses processus internes.

Tous ces signaux de contrôle sortant des différents décodages de formats passent par des processus de *Merge* pour être distribués vers les unités d'exécution.

6.3.3 Unité PC

L'unité PC doit mettre à jour le compteur programme et charger une nouvelle instruction en fonction du type d'instruction émis par le décodeur. Pour les instructions courantes (ALU, Load-Store...) le compteur programme est incrémenté de 2 octets pour charger l'instruction suivante en mémoire. Lorsqu'une instruction de branchement a lieu, le `PC` est soit incrémenté avec l'offset signé intégré dans l'instruction de branchement (`Offset9`) si le résultat du branchement est pris, soit incrémenté normalement s'il n'est pas pris. Pour les quelques instructions qui viennent modifier le `PC` directement pour faire un saut, un signal

d'adresse sur 16 bits en entrée vient des différentes unités concernées (MOV, LD-ST et FUNC Unit). Lorsqu'une nouvelle interruption se déclenche, c'est le décodeur qui envoie une adresse pour charger en premier le contenu du vecteur d'interruption. Cette nouvelle adresse est dirigée vers l'unité PC et celle-ci écrit cette adresse comme `PC` de départ et charge la première instruction.

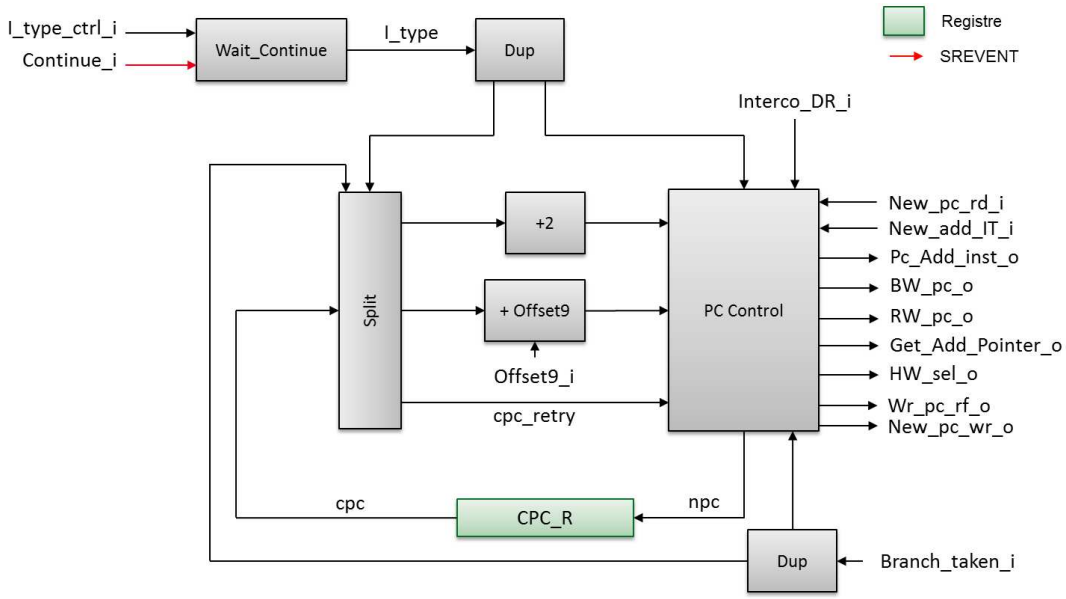


FIGURE 6.15 – Microarchitecture de l'unité de chargement d'instruction (PC Unit)

La figure 6.15 présente la microarchitecture de l'unité PC. Le processus de contrôle du `PC` (PC Control) récupère les signaux de nouveaux `PC` provenant soit du bus de communication, soit des unités d'exécution, soit du décodeur ou soit des calculs d'incrément pour modifier la valeur du registre courant du `PC` CPC_R. Par la même occasion elle vient écrire la valeur du nouveau `PC` dans le registre `R15` du banc de registres et fait une demande de lecture sur le bus de communication pour lire l'instruction à cette adresse.

Lorsqu'une instruction a été exécutée, le décodeur envoie le signal I_type_ctrl pour indiquer au `PC` l'instruction qui vient d'être exécutée. Il y a 6 possibilités pour I_type_ctrl :

- INC_PC : pc = pc + 2
- B_COND_PC : si Branch_taken = 1 alors pc = pc + signed(offset9 :0) sinon pc = pc + 2
- NEW_PC : pc = New_pc_rd_i (possible avec BL, POP, MOV et LDRH)
- NEW_INT : pc = mem[New_add_inst_i]
- END_IT_PC : pc = 0xF000
- PC_RETRY : pc = CPC_R

Il a été choisi d'écrire le nouveau pc dans un registre et non dans une série de *Half Buffer* car, dans le cas de l'instruction Break, il faut ré-exécuter le même `PC` une fois que l'instruction `BREAK` a été remplacée par l'instruction originelle. Or si le `PC` était stocké dans une série de *Half Buffer*, le jeton aurait été consommé pour lire l'instruction Break et n'aurait pas été capable de relire le même `PC` pour exécuter l'instruction originelle.

6.3.4 Banc de registres

Le banc de registres est constitué de 16 registres de 32 bits avec deux ports d'accès en lecture pour extraire les opérandes 1 et 2 des instructions contenues dans des registres et un

port d'écriture pour que les unités d'exécution puissent écrire le résultat de l'instruction.

La figure 6.16 présente l'architecture du banc de registres. Les registres ont été implémentés avec des *Shared Variable* avec gestion des conflits lecture-écriture. Ce conflit est le conflit minimal à implémenter pour avoir le meilleur rapport surface-fonctionnalité. Il est possible alors qu'une lecture se passe simultanément avec une écriture sur le registre. Ceci est géré au niveau de la synthèse pour que cela reste fonctionnel au niveau du registre.

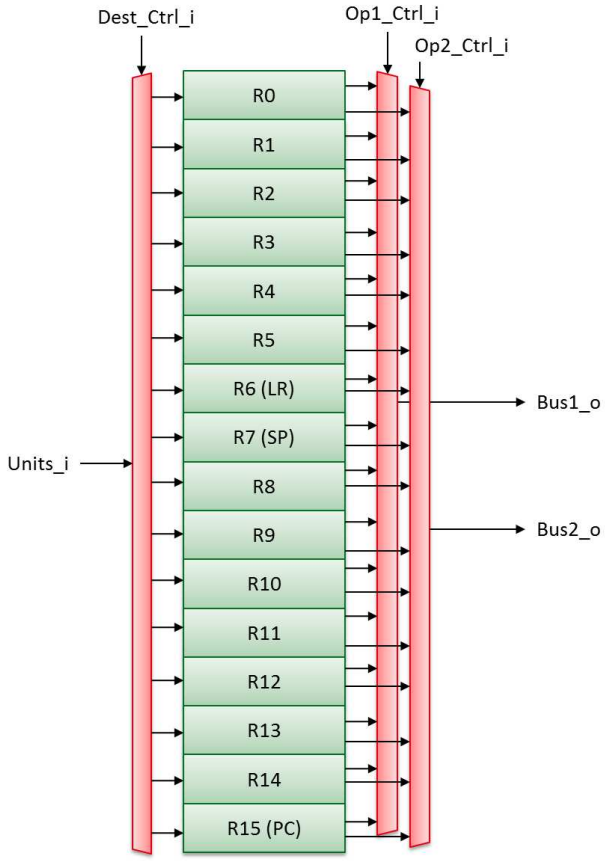


FIGURE 6.16 – Microarchitecture du banc de registres

Il a été choisi de mettre le *Link Register* (LR) sur R6 et le *Stack Pointer* (`SP`) sur `R7` car beaucoup d'instructions manipulent des registres sur les *Low Registers* (`R0->R7`). Il est alors plus simple pour le compilateur d'utiliser les *Low Registers* que les *High Registers*.

6.3.5 Bus de communication

Le bus de communication relie tous les modules maîtres (*masters*) aux modules esclaves (*slaves*). Les signaux transportés par ce bus sont les suivants :

— adresse sur 16 bits
— taille du transfert sur 2 bits (00=octet, 01=demi-mot, 10=mot)
— lecture/écriture sur 1 bit (R=0, W=1)
— donnée à écrire sur 32 bits
— donnée lue sur 32 bits

Dans le circuit, le bus a été implémenté avec deux maîtres (WUC et Debug Unit) et quatre esclaves (IT_Ctrl, WUC, mémoire et un module déclencheur d'IT). La figure 6.18 présente la microarchitecture générique du bus de communication. Ici les deux maîtres

peuvent faire des requêtes sur le bus simultanément. La sélection de la requête du maître est faite grâce au signal d'adresse. Les adresses passent par un *Mutex* pour rendre les requêtes mutuellement exclusives. Ensuite les autres signaux du bus sont sélectionnés grâce à des *Merge*. Une fois les signaux de requête pour un même maître sélectionné, ceux-ci sont distribués au bon esclave grâce à un décodage de l'adresse. Si la requête est une lecture alors il faut que la donnée lue soit renvoyée au bon maître grâce à un signal de contrôle du *Split* de chaque *Slave*. Ces signaux de donnée provenant de chaque *Slave* passent alors par un *Merge* pour ressortir un seul signal de donnée lu vers le maître.

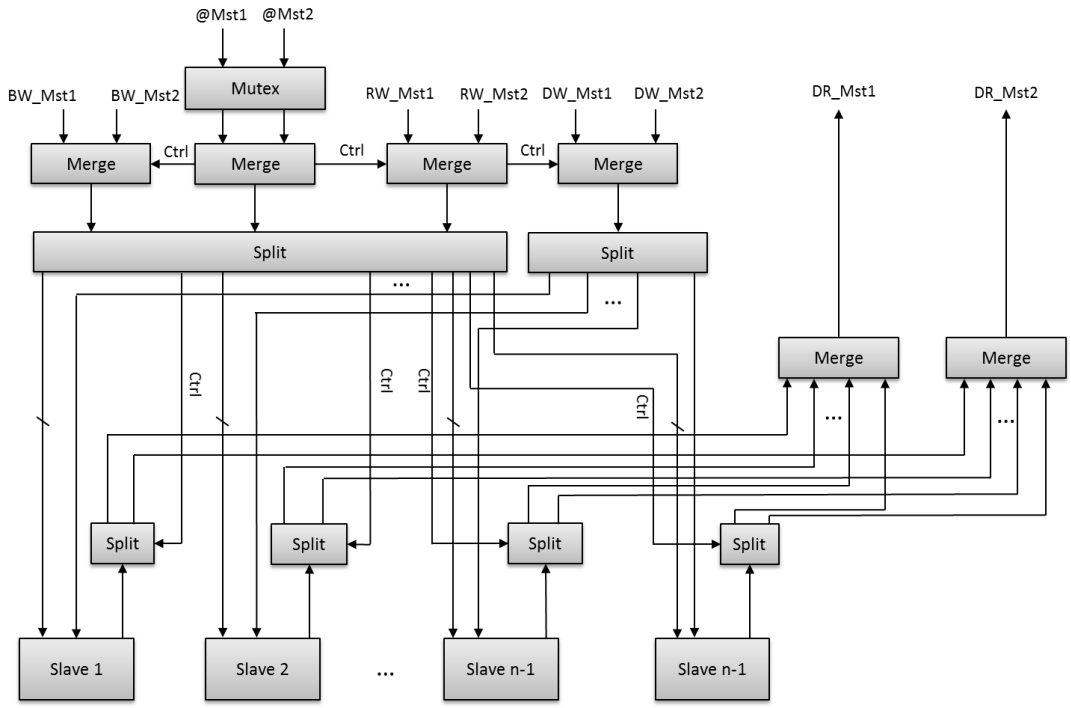


FIGURE 6.17 – Microarchitecture du Bus de communication de la plateforme de réveil

Lorsqu'un maître ou un esclave est connecté sur le bus, il doit respecter son protocole en lecture et en écriture. Ainsi un maître doit respecter les règles suivantes en écriture et en lecture.

— En écriture, si un octet ou un demi-mot est écrit sur un esclave, alors le maître doit dupliquer la donnée sur le bus autant de fois pour remplir le bus. L'octet est dupliqué sur les quatre octets du signal de donnée à écrire et le demi-mot deux fois.

— En lecture, le maître doit reconstruire la donnée en fonction de la taille du mot lu et des deux bits d'adresse de poids faible pour savoir quelle partie du mot est à sélectionner et à réaligner.

D'un autre coté l'esclave doit respecter le protocole suivant.

— En écriture, un esclave va créer un masque d'écriture sur 4 bits pour masquer une partie de la donnée et écrire les octets au bon endroit. De cette manière des écritures sur 8, 16 et 32 bits sont possibles dans l'esclave.

— En lecture, un esclave renvoie toujours le mot de 32 bits de l'adresse voulue alignée sur 32 bits (en masquant les 2 bits de poids faible de l'adresse).

Si les maîtres et esclaves respectent bien ces règles alors un maître peut émettre une requête en écrivant uniquement sur les bons signaux du bus (écriture : Add, RW, BW et DW ; lecture Add, RW, BW). Un rendez-vous ayant lieu dès l'entrée du bus sur ces signaux,

la requête se fera uniquement quand ils seront tous présents.

6.3.6 Unité *load - store*

L'unité de chargement-sauvegarde s'occupe de charger des données dans les registres généraux pour travailler dessus et sauvegarde les données contenues dans les registres généraux sur l'espace d'adressage (mémoire ou périphériques). Cette unité doit gérer trois modes d'adressage :

— l'adressage indirect à registre (`[Rs] -> Rd` ou `Rd -> [Rs]`)
— l'adressage base + index (`[Rs1 + Rs2] -> Rd` ou `Rd -> [Rs1 + Rs2]`)
— l'adressage base + offset (`[Rs + #imm] -> Rd` ou `Rd -> [Rs + #imm]`)

Ces instructions doivent donc calculer l'adresse à partir d'une ou plusieurs valeurs contenues dans les registres généraux ou avec un immédiat. Mais aussi, dans le cas d'un *load*, faire l'accès mémoire avec l'adresse calculée et stocker la donnée dans le registre de destination. Et dans le cas d'un *store*, charger la donnée depuis un registre pour la stocker dans un endroit de l'espace d'adressage.

Le *load* dans les différents modes d'adressage ne pose pas de problèmes d'accès au banc de registres alors que dans le cas du *store* il y a un problème d'accès. En effet, le *store* dans le mode d'adressage base + index doit effectuer trois accès en lecture au banc de registres. Or il y a uniquement deux bus de lecture au banc de registres. Il faut donc faire intervenir une séquentialité dans l'accès au banc de registres pour charger les deux opérandes afin de construire l'adresse et de charger la donnée à sauvegarder. Pour des raisons de régularité du matériel, le décodeur envoie dans un premier temps la lecture de la première partie de l'adresse. Une fois l'accès fini, les signaux de contrôle pour lire la deuxième partie de l'adresse sont envoyés sur le bus 2 du banc de registres et la lecture de la donnée est faite sur le bus 1 du banc de registres.

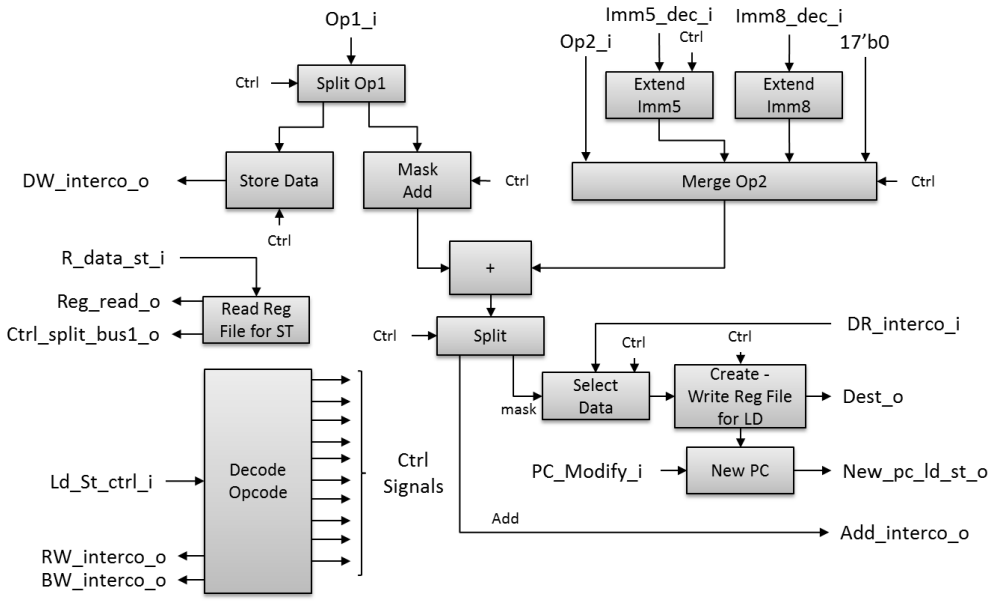


FIGURE 6.18 – Microarchitecture de l'unité Load-Store

La figure 6.18 présentant la microarchitecture de l'unité *Load-Store* montre bien que l'opérande 1 provenant du bus 1 du banc de registre peut être à la fois une adresse et une donnée dans le cas d'un *store*. Un *Merge* est présent pour la sélection de l'opérande 2

nécessaire au calcul de l'adresse pour un *load* ou un *store* en fonction du mode d'adressage. Lors d'un *store*, il y a toujours une séquentialité de la lecture sur l'opérande 1 quelque soit le mode d'adressage. En premier l'opérande 1 est utilisé pour calculer l'adresse et ensuite l'opérande 1 représente la donnée à sauvegarder. Le module de décodage de l'*opcode* distribue les signaux de contrôles en sélectionnant les bonnes parties de l'*opcode* vers les différents processus internes. En effet, nous avons vu que le décodeur créé l'*opcode* de manière à ce que l'unité *Load-Store* n'ait plus qu'à sélectionner et distribuer la partie de l'*opcode* et l'envoyer en tant que signal de contrôle.

6.3.7 Unité arithmétique et logique

L'unité arithmétique et logique (ALU) exécute les instructions du tableau 5.3. Elles se répartissent en quatre sous unités d'exécution : l'unité arithmétique qui exécute les instructions `add/sub/comp`, l'unité logique qui exécute les instructions logique bit à bit, l'unité d'extension pour étendre le signe ou pas d'un nombre sur 8 ou 16 bits et l'unité de décalage qui exécute les instructions de décalages.

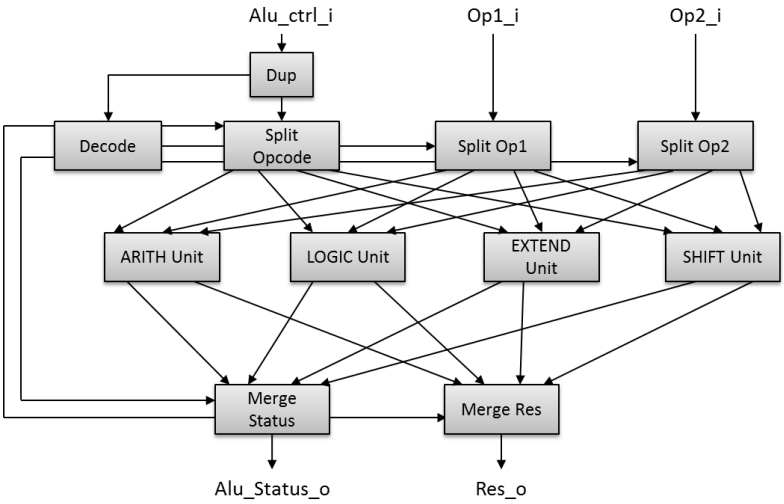


FIGURE 6.19 – Microarchitecture de l'unité arithmétique et logique

La figure 6.19 présente l'architecture de l'ALU. Celle-ci récupère l'*opcode* créé par le décodeur (section 6.3.2), distribue les signaux de contrôle vers les différents *Split* pour rediriger les opérandes vers les sous-unités et un signal de contrôle pour que les sous-unités sachent quelle instruction est à exécuter. Nous avons vu que les sous-unités comme l'unité arithmétique génère à la fois le résultat et les bits de statut `N`, `Z`, `C`, `V`. Ceux-ci sont ensuite envoyés au *Merge* pour ne ressortir qu'un seul résultat et un seul vecteur de bits de statut.

L'ALU n'a besoin que d'une addition pour effectuer l'addition et la soustraction car le calcul est effectué en complément à deux. En effet l'opérande à soustraire est inversée et la valeur 1 est ajoutée au calcul pour avoir le résultat final de la soustraction. Ce circuit possède donc une dynamique comprise entre $[-2^{n-1}, 2^{n-1} - 1]$ pour un nombre signé et une dynamique de $[0, 2^n - 1]$ pour un nombre non signé. L'addition-soustraction est synthétisée par ACC à l'aide de 32 cellules *Full Adder* en architecture de type propagation de la retenue (*ripple carry adder*). Tous les *Full Adders* sont mis en série. Il faut donc que la retenue se propage dans les 32 *Full Adders* séquentiellement pour avoir le résultat final. Ceci est l'additionneur le plus basique car il engendre le moins de matériel mais, est

par conséquent, le plus lent. Un autre type d'additionneur pourrait être mis en place pour accélérer le calcul, c'est l'additionneur à retenue anticipée (*carry-lookahead adder*), mais il n'a pas été jugé utile de l'implémenter dans ce circuit afin d'avoir la surface la plus petite.

6.3.8 Unité de branchement

L'unité de branchements prend en entrée la condition de branchement de l'instruction, les bits de statut de l'opération précédente qui les a modifié et génère le signal Branch_taken afin que l'unité PC fasse un saut dans le programme avec un *offset* ou sans. Les bits de statut à la sortie de l'ALU sont dupliqués vers le décodeur pour les stocker dans le registre d'état du WUC et pour le stocker dans le registre de l'unité de branchement.

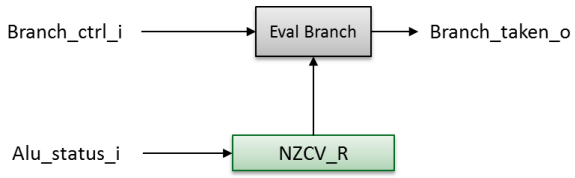


FIGURE 6.20 – Microarchitecture de l'unité de branchement

6.3.9 Unité de déplacement

L'unité de déplacement (figure 6.21) va déplacer une valeur de registre ou un immédiat dans un autre registre. Cette unité est relativement simple puisqu'elle lit le bus 1 du banc de registre ou un immédiat sur 8 bits et écrit sur le bus d'écriture la nouvelle valeur. Elle indique aussi au PC la nouvelle adresse ou sauter s'il y a une écriture sur `R15`.

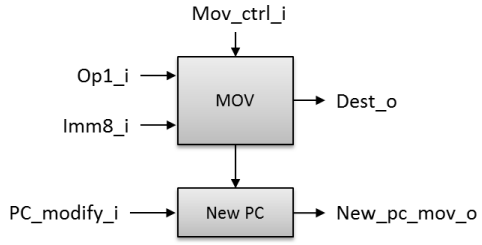


FIGURE 6.21 – Microarchitecture de l'unité de déplacement

6.3.10 Unité de support aux fonctions

L'unité de support aux fonctions s'occupe des trois instructions `BL`, `PUSH`, `POP` du tableau 5.5. Sa microarchitecture présentée figure 6.22 s'occupe des algorithmes illustrés figure 5.3.

Le module *Cmd Read Reg* prend en entrée la liste de registre et indique au processus *Read Rx* la valeur du bit correspondant contenu dans la liste de registre. S'il est à 1 alors le processus *Read Rx* envoie un évènement au processus *Read Request*. Celui-ci envoie une requête de lecture sur le bus 1 du banc de registre et dès qu'elle est effectuée et consommée, *Read Rx* envoie un évènement au prochain processus *Read Rx* qui teste si le bit est à 1 et ainsi de suite. De cette manière pour un `PUSH`, séquentiellement les requêtes sont faites au banc de registres et les valeurs sont récupérées par le processus *Read Write Reg* pour venir

effectuer des accès d'écriture dans la mémoire à travers le bus de communication. Entre chaque requête d'écriture en mémoire un décrément du pointeur de pile est réalisé.

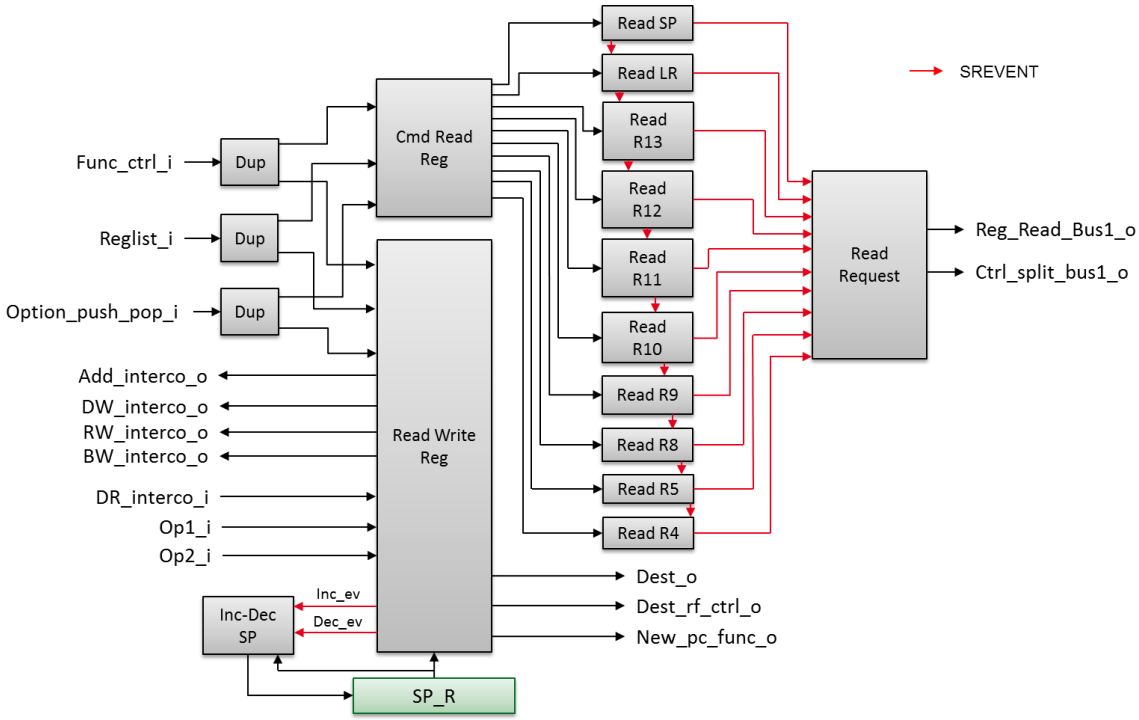


FIGURE 6.22 – Microarchitecture de l'unité de support aux fonctions

Dans le cas du `POP`, le registre `SP` est tout d'abord lu (via *Cmd Read Reg* puis *Read SP* puis *Read Request*) et incrémenté dans *Read Write Reg* pour effectuer la première lecture en mémoire. Dans le processus *Read Write Reg*, si le bit équivalent dans la liste de registre est à 1 alors une requête de lecture au pointeur de pile courant est effectuée et la donnée lue est écrite dans le registre équivalent au bit de la liste de registre. Ensuite le pointeur de pile locale `SP_R` est incrémenté et la prochaine requête peut être réalisée.

L'instruction `BL` est exécutée par le processus *Read Write Reg*. En effet, il reçoit sur le bus 1 du banc de registre le `PC` actuel et sur le bus 2 du banc de registres le nouveau `PC`. Ce processus alors incrémente le `PC` courant de 2 pour sauvegarder l'adresse de retour dans le `LR` en écrivant sur le bus d'écriture du banc de registres. Puis, une fois l'écriture finie, il écrit sur le bus d'écriture du banc la valeur du nouveau `PC` et indique à l'unité PC la valeur du nouveau `PC`.

6.3.11 Unité de debug

Cette unité est la deuxième unité maître sur le bus. Elle s'occupe de la liaison avec l'extérieur pour programmer et debugger le système. Seules deux commandes élémentaires existent pour venir effectuer n'importe quelle opération sur le système, la lecture et l'écriture. De plus, les registres généraux du WUC peuvent être lus ou écrits et les lectures ou écritures sur le bus peuvent se faire sur 8, 16 ou 32 bits. Cela implique une commande de lecture ou écriture sur 4 bits comme montré figure 6.23. Dans la trame envoyée depuis l'extérieur qui arrive en parallèle dans l'unité de debug, s'en suit une adresse sur 16 bits ou un numéro de registre, et une donnée sur 32 bits.

Un processus de contrôle va alors effectuer la requête sur le bon bus en fonction du type

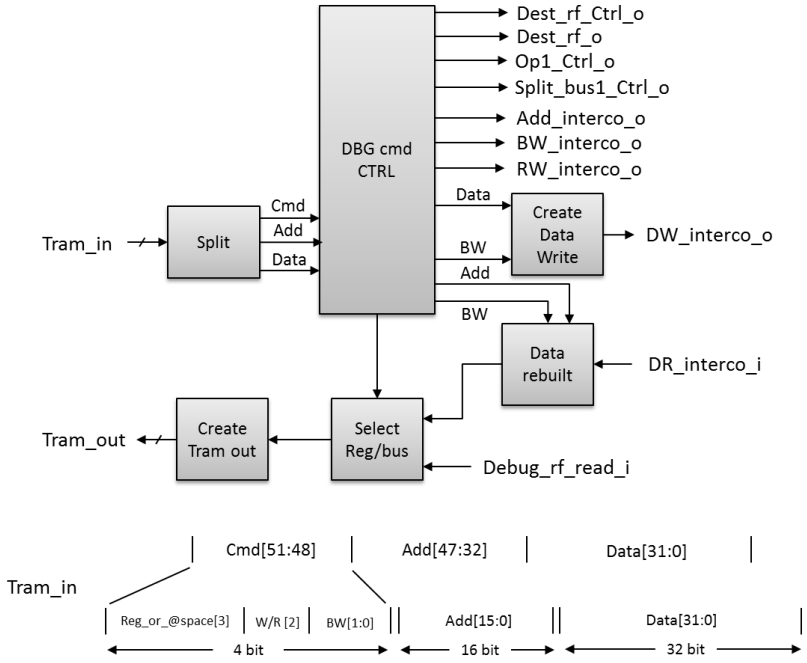


FIGURE 6.23 – Microarchitecture de l'unité de debug et trame de debug

de requête lecture-écriture et si l'accès est un accès aux registres généraux ou à l'espace d'adressage. Si l'accès est sur l'espace d'adressage, il respecte alors les règles vues dans la section de la microarchitecture du bus (section 6.3.5).

6.4 Environnement logiciel

Comme cela a été précisé, le jeu d'instructions et l'architecture sont entièrement customisés. Par conséquent, aucune suite de logiciel n'existe (assembleur, compilateur, debugger) pour ce processeur. C'est pourquoi un travail a été fait au laboratoire LIALP du CEA LETI pour mettre en œuvre le compilateur, assembleur et désassembleur. Le compilateur du WUC a été développé à partir du framework LLVM [18]. Ce framework permet de séparer :

- le *front-end* : un par langage, qui traduit un code source en une représentation intermédiaire générique (IR = *Intermediate Representation*) ;
- le *back-end* : un par architecture matérielle, qui traduit une représentation intermédiaire en un assembleur ou binaire spécifique à un processeur.

Le framework LLVM fournit également tout un panel d'optimisations sur l'IR (la suppression de code mort, propagation de constante, mutualisation de portions de code...). Le développement du compilateur pour le WUC a consisté à ajouter un nouveau *back-end*. Parmi les étapes principales de ce développement, on peut citer :

- description des registres du WUC ;
- description du jeu d'instructions : format binaires, entrées/sortie, dépendances implicites avec des registres de statut du cœur, scheduling, etc.
- développement des transformations non triviales de l'IR vers les instructions machines ;

18. anciennement Low Level Virtual Machine

- optimisations spécifiques au WUC (exploitation des bits `N`, `Z`, `C`, `V` pour un `ADD` suivi d'un `CMP`).

Le *back-end* a été validé sur des tests unitaires (une fonction C qui additionne deux entiers par exemple), puis par des tests de stress pour vérifier que l'étendue de l'IR était couverte.

6.5 Environnement de conception et simulation

Nous allons finir ce chapitre avec la partie sur l'environnement de simulation et de conception mixte asynchrone-synchrone. Le but est de montrer au lecteur les différentes étapes dans la conception mixte et les différents niveaux de hiérarchie.

6.5.1 Hiérarchie des modules du circuit pour la simulation mixte asynchrone-synchrone

Il existe trois niveaux de simulation, les simulations comportementales, les simulations post-synthèses et les simulations post-backend. La figure 6.24 présente ces trois niveaux de simulation avec les différentes hiérarchies mises en place pour les simulations.

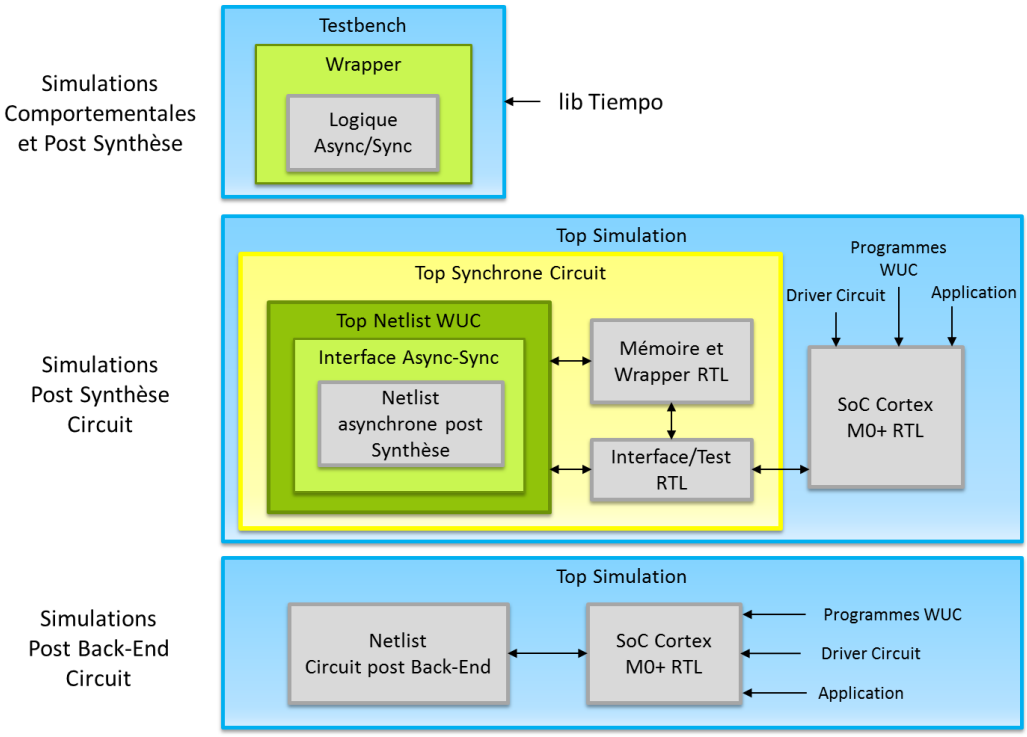


FIGURE 6.24 – Hiérarchies des modules pour les différents niveaux de simulation

La première étape dans la conception est d'effectuer les simulations comportementales pour voir si le module respecte bien le cahier des charges. Pour cela un TestBench est créé et vient piloter les signaux d'entrée et lire les signaux de sortie. Le comportement des canaux est décrit par les bibliothèques mises en place par Tiempo.

Une fois les simulations comportementales validées et l'étape de synthèse finie, des simulations post-synthèses peuvent être mises en place. Le même TestBench que pour les simulations comportementales peut être utilisé grâce à un *Wrapper* placé entre le TestBench et la netlist post-synthèse du module. L'étape de simulation post-synthèse est primordiale en asynchrone car elle fait apparaitre de nombreuses erreurs de design et des

problèmes de blocage dans le circuit. Ceci est dû à un manque de mémorisation intermédiaire et est résolu grâce à la mise en place de *Half-Buffer* en série sur certain canaux et qui n'apparaissaient pas dans les simulations comportementales. La figure 6.24 présente dans le cadre du circuit final la hiérarchie des modules du circuit pour les simulations post-synthèses. Cette *Top netlist* du WUC contient la netlist asynchrone synthétisée par ACC et toutes les interfaces asynchrones/synchrones pour communiquer avec l'extérieur. Le Top du circuit est alors piloté par un microcontrôleur dont le cœur de processeur est ici le Cortex M0+. Des *drivers* du circuit et principalement du WUC ont été développés en C pour pouvoir gérer le circuit depuis une application C exécutée par le Cortex M0+. De plus, différents programmes de test pour le WUC ont été compilés avec son compilateur ou créés en assembleur pour tester les différentes fonctionnalités du WUC. Une interface de debug a été développée dans les drivers pour que le Cortex M0+ puisse stopper le cœur du WUC, le redémarrer, poser des *breakpoints*, lire et écrire dans tout l'espace d'adressage du circuit, ainsi que dans les registres généraux du WUC pour voir si l'exécution de chaque instruction s'est bien passée.

Une fois les simulations post-synthèses validées et l'étape de Back-End finie, les simulations post-backend peuvent être réalisées. Les mêmes programmes de test que précédemment peuvent être utilisés sauf que cette fois ci c'est la netlist finale du circuit qui est simulée et pilotée par le Cortex M0+.

6.5.2 Flot de conception du circuit

Nous avons pu voir la hiérarchie des modules pour les simulations de circuits mixtes asynchrones-synchrones. Voyons maintenant le flot de conception mis en place pour le circuit et les différents outils utilisés. Ce flot est présenté figure 6.25.

Les modules asynchrones sont développés en SystemVerilog et synthétisés avec ACC pour obtenir une netlist asynchrone. Un fichier de contraintes (fichier sdc [19]) et un fichier des délais (fichier sdf [20]) sont aussi créés. Ensuite, Design Compiler est utilisé pour avoir une netlist de la macro asynchrone (netlist asynchrone + interface async/sync) en propageant les contraintes aux interfaces de la macro. Cela génère aussi un nouveau fichier de contraintes. Les autres parties du circuit sont synthétisées avec Design Compiler ou le FrontEnd Kit de STMicroelectronics qui fait appel à Design Compiler. Ensuite SocEncounter Kit, un autre outils de STMicroelectronics pour effectuer l'étape de Backend et faisant appel à Innovus de Cadence est utilisé pour générer la netlist finale du circuit, son fichier de délais sdf et le fichier à envoyer pour fabrication : le gds. Afin de définir les domaines de puissance, un fichier upf [21] est utilisé par Cadence. Il est possible alors de simuler cette netlist finale et d'en extraire l'activité dans un fichier vcd pour analyser sa consommation sur PrimeTime PX. On obtient alors les rapports de consommation du circuit. Les performances et temps de réveil du WUC sont directement analysés avec Questasim sur la netlist finale.

6.6 Conclusion

Nous avons présenté dans ce chapitre la microarchitecture du WUC en décomposant pour chaque unité les sous-blocs ou processus codés en SystemVerilog et synthétisables

19. Synopsys Design Constraints
20. Standard Delay Format
21. Unified Power Format

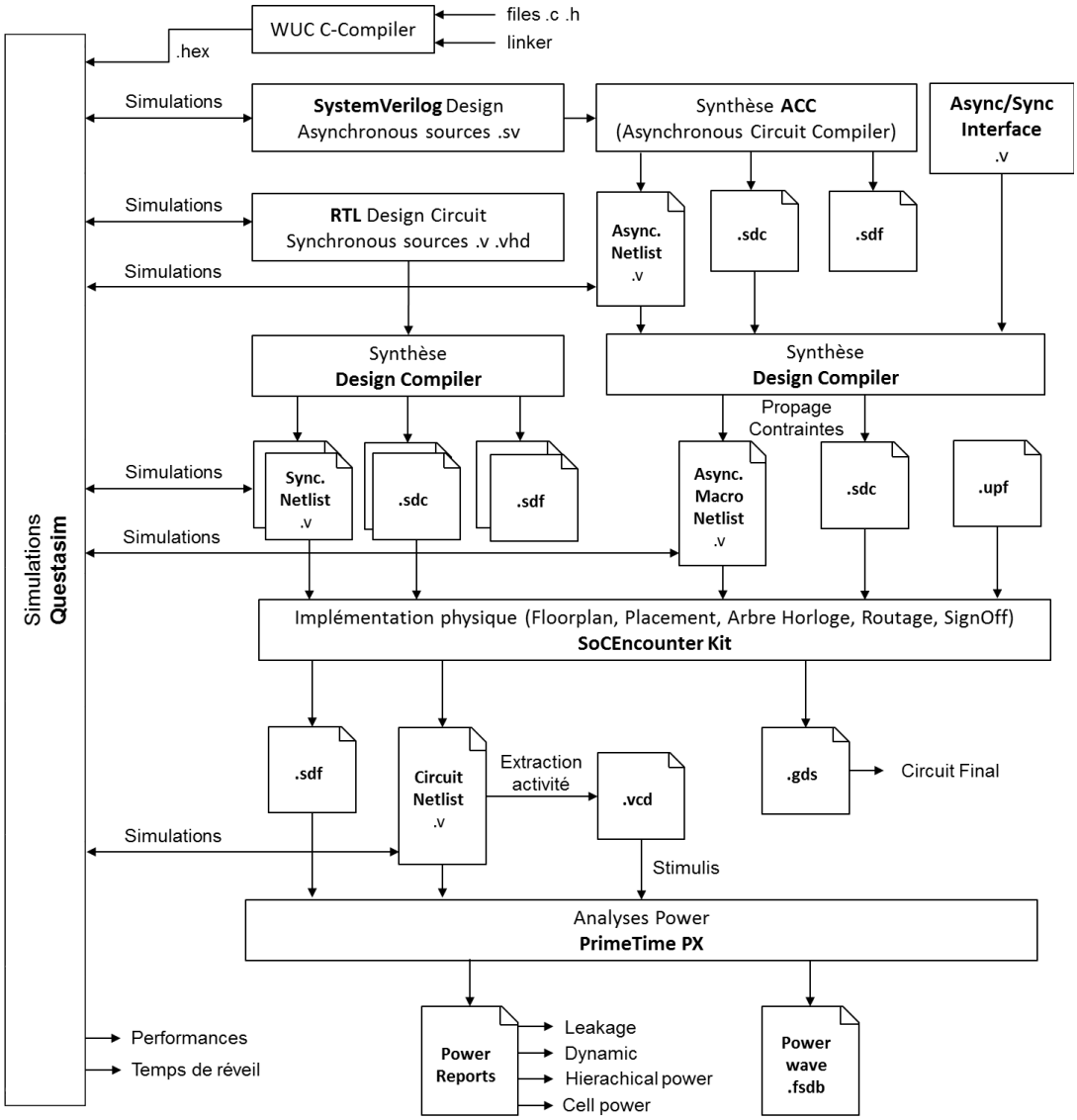


FIGURE 6.25 – Flot de conception du projet

par les outils de Tiempo (voir exemple de code en SystemVerilog de processus asynchrone section 6.1.4).

L'objectif est d'avoir le meilleur rapport surface/performance-fonctionnalité tout en privilégiant la surface pour éviter une consommation statique trop importante. Nous avons pu voir que mettre en place un processeur qui n'exécute qu'une instruction jusqu'à complétion ne nécessite pas de signaux de contrôle supplémentaires. En effet, ce sont l'envoi des données, leur rendez-vous et leur invalidation qui permettent la séquentialité grâce au protocole physique intrinsèque de la logique QDI qui engendre le blocage des modules tant qu'une opération n'est pas finie. Cette technique permet effectivement d'avoir le matériel minimum pour l'exécution d'un programme. Néanmoins, nous verrons au chapitre 7 que les performances en vitesse d'exécution de cette architecture (non optimisées) sont assez faibles.

Cette première microarchitecture peut être bien sûr améliorée au niveau des performances pour permettre un parallélisme entre les différentes unités et tirer parti des avan-

tages de l'asynchrone grâce au recouvrement des unités [164]. Par manque de temps, cette optimisation n'est pas faite dans cette première version. Maintenant qu'un flot de conception complet est mis en place, il va être aisé d'optimiser les différents modules en termes de débit/latence et ainsi envisager une architecture qui permette de paralléliser les tâches et d'obtenir une architecture à temps de calcul minimal. Toutes les optimisations possibles et perspectives d'améliorations et de réduction de la consommation seront vues dans la partie conclusion et perspectives.

La microarchitecture présentée dans cette partie a été implémentée en technologie STMicroelectronics UTBB FDSOI 28nm, dont les résultats en performances et consommations sont exposés au prochain chapitre.

Chapitre 7

Implémentation physique, tests, performances et consommation du processeur de réveil

7.1 Implémentation physique du circuit complet

Le circuit a été implémenté en technologie UTBB FDSOI 28nm LVT de STMicroelectronics. Comme expliqué section 3.3 cette technologie possède un grand degré de flexibilité. En effet cette technologie a un excellent contrôle électrostatique du canal, de faibles fuites de courant et une immunité à la fluctuation aléatoire des dopants. L'utilisation de la polarisation arrière permet dynamiquement d'augmenter les performances en vitesse en effectuant du FBB ou de diminuer les fuites de courants en effectuant du RBB. De plus, la surface silicium est par conséquent plus petite que pour les technologies utilisées habituellement pour l'IoT (65nm, 90nm, 130nm...) ce qui engendre pour des productions de masse des coûts plus faibles et une empreinte carbone beaucoup plus petite [50]. Comme cela a été vu section 6.5, la macro asynchrone a été synthétisée à l'aide d'ACC et est instanciée dans un *top* synchrone. Dans cette section l'architecture du circuit, de la macro asynchrone finale et les résultats en termes de surface vont être présentés.

7.1.1 Architecture globale du circuit et partitionnement des domaines de puissance

La figure 7.1 présente l'architecture *top* du circuit envoyée en fabrication. Il a été choisi de partager l'architecture en trois domaines de puissance.

- Domaine Macro Asynchrone : domaine de toute la logique asynchrone et dont la consommation peut être mesurée grâce à une alimentation isolée. Ce domaine a la possibilité d'avoir une polarisation arrière (FBB).
- Domaine Mémoire : contient le *cut* mémoire de 4KB et son *wrapper* (chaines de délai configurables et interface asynchrone-synchrone. La consommation de ce domaine peut être mesurée grâce à une alimentation isolée.
- Domaine *Always On* (AO) : contient l'interface de communication série (SPI), l'interface registres pour les chaines de délai configurables, les fifos GALS qui sont connectées à la macro asynchrone pour la programmation et le debug du WUC, un module de test contenant des timers pour la mesure des routines d'interruptions et de réveil et une FLL pour l'horloge du timer rapide.

Ce partitionnement en domaines de puissance permet de mesurer les consommations statiques et dynamiques de la logique asynchrone et de la mémoire indépendamment les unes des autres. Les domaines de puissance ont chacun leur alimentation. Le domaine

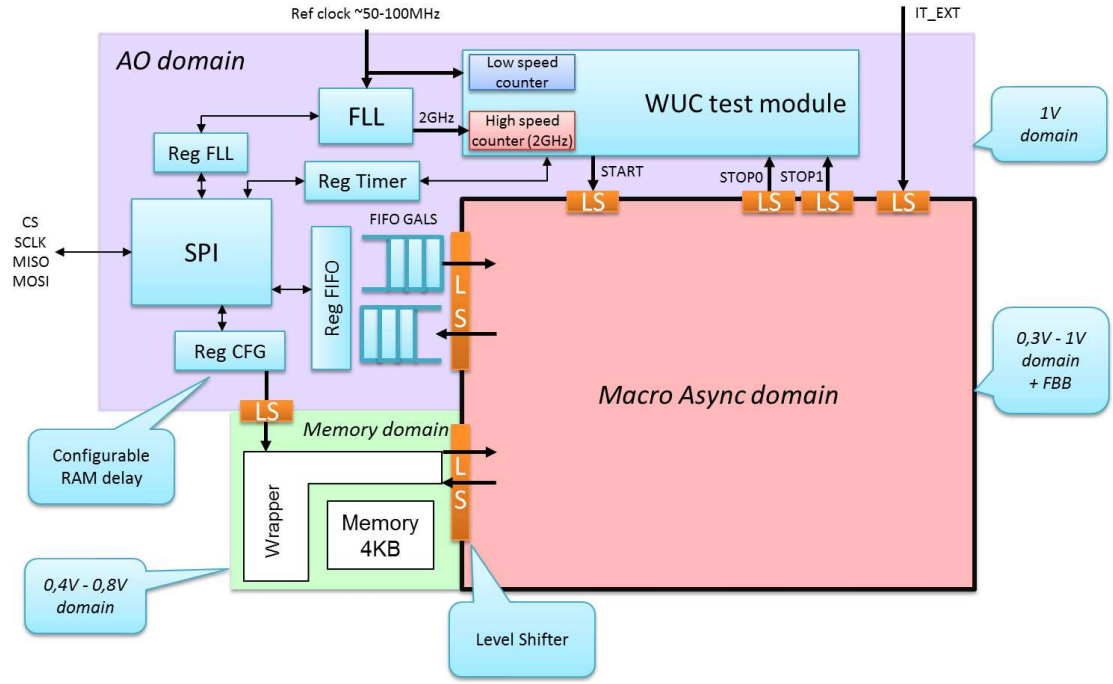


FIGURE 7.1 – Architecture du circuit WALPP

asynchrone possède des alimentations supplémentaires pour la polarisation arrière (FBB). Pour pouvoir mesurer les temps de réveil et les temps d'exécution d'une routine d'interruption afin d'obtenir les performances du circuit, des *timers* ont été implémentés : un *timer* très rapide pour mesurer les temps de réveil et un autre plus lent pour mesurer le temps d'une routine d'interruption. Il y a des *level shifter* pour que les signaux entre un domaine et un autre soient adaptés à la tension du domaine.

7.1.2 Architecture de la macro finale asynchrone

Nous allons présenter dans cette section l'architecture de la macro asynchrone finale qui a été envoyée en fabrication. La figure 7.2 présente cette architecture.

Les interfaces de la macro asynchrone sont présentes à trois endroits :

- au niveau de l'unité de Debug : un module de conversion transforme les signaux multi-rail 4 en signaux double rails arrivant en parallèles des fifos GALS [1] ;
- au niveau de l'interface mémoire : les signaux de type *Bundled Data* se connectent au *wrapper* mémoire du domaine mémoire ;
- au niveau du contrôleur d'interruptions et du décodeur : l'IT6 est connectée à une IT extérieure en passant par un détecteur de front pour transformer les fronts en une valeur sur un signal double rail. L'IT7 est connectée au signal START provenant du module de test en passant aussi par un détecteur de front. Les signaux STOP0 et STOP1 font références au premier fetch en mémoire et à la fin d'une routine d'interruption respectivement.

Les signaux START et STOP doivent respecter le protocole illustré figure 7.2 afin de mesurer les performances du circuit. Le signal START reste au niveau haut tant que le signal STOP ne monte pas à 1. Dès que le signal START passe sur niveau bas alors le signal STOP se baisse à son tour. Les signaux STOP sont des SREVENT dont le signal

1. Globally Asynchronous Locally Synchronous

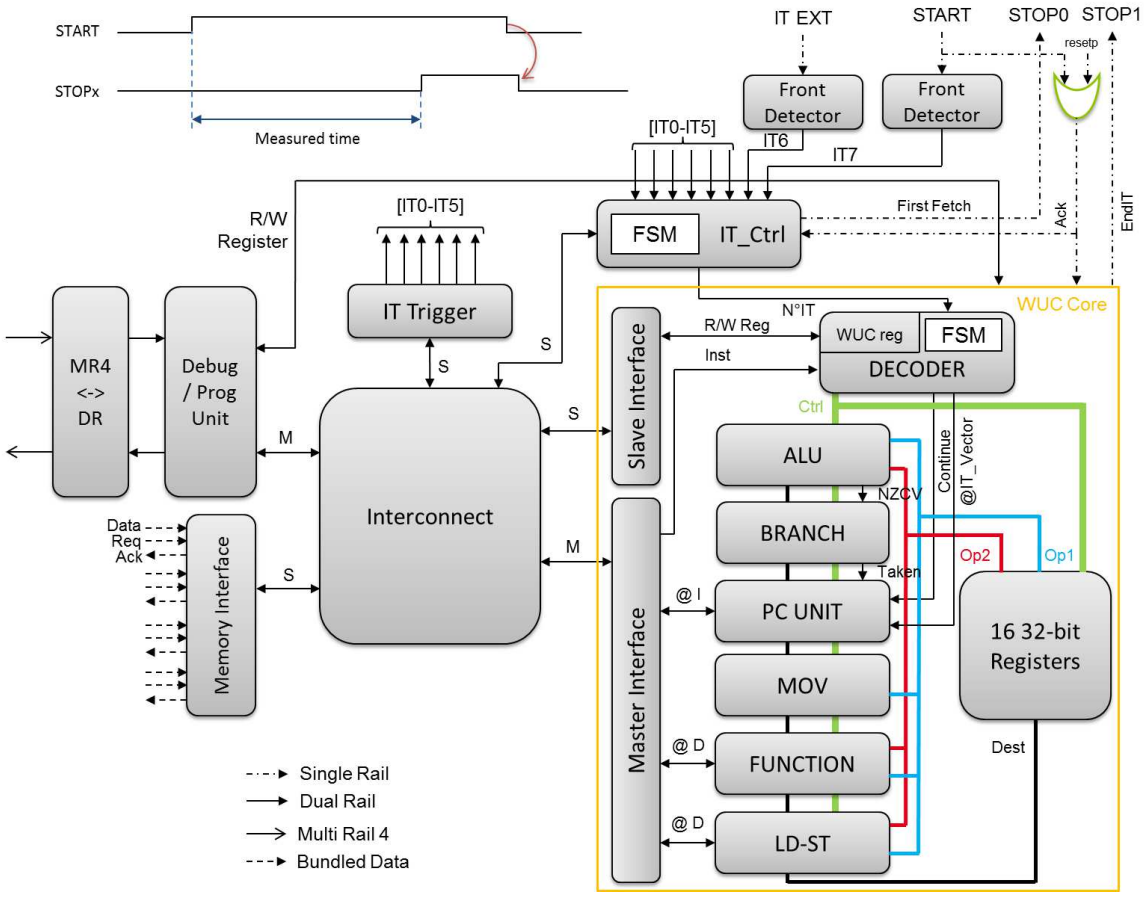


FIGURE 7.2 – Architecture de la macro asynchrone

d'acquittement est connecté au signal START.

7.1.3 Architecture des détecteurs de fronts

La figure 7.3 permet de transformer les signaux simples rail en signaux doubles rails en détectant ses fronts montants et descendants. Cette structure permet un retour à zéro de la sortie même si l'entrée reste à l'état haut, de cette manière la détection de front est décorrélée de la génération du jeton asynchrone. Ainsi le contrôleur d'interruptions peut consommer le jeton même si l'interruption n'est pas revenue à l'état bas.

7.1.4 Implémentation physique et résultats en surface

La macro asynchrone compte 32,1K cellules, dont 18,3K cellules asynchrones et 13,8K cellules combinatoires. La majorité des cellules proviennent du banc de registres, de l'ALU et de l'unité de support aux fonctions comme présenté dans le tableau 7.1. Leur surface représente 20,3% pour l'ALU, 18,7% pour le banc de registres et 13% pour l'unité de support aux fonctions. Ceci est en partie dû au nombre de cellules asynchrones présentes dans ces unités et tout particulièrement les cellules de Muller dont le nombre est conséquent (9645). Il y a 25% de plus de cellules asynchrones que de cellules combinatoires et les cellules asynchrones peuvent être nettement plus volumineuses que les cellules combinatoires (8 transistors pour une cellule de Muller 2 entrées figures 6.5).

Les résultats en surface présentés ici sont les résultats de la surface des modules sans

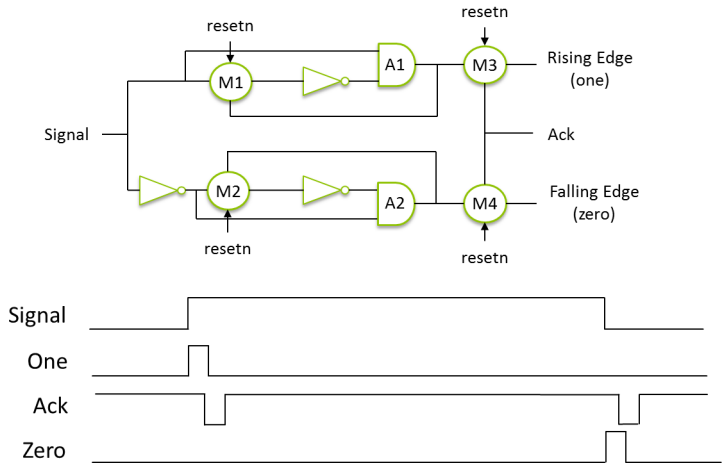


FIGURE 7.3 – Détecteurs de front montant et descendant QDI

Modules		Surface (μm^2)	Pourcentage	Total	Asynchrone	Combinatoire
WUC Core	Decode	4044,42	7,9%	2688	1401	1287
	PC	2213,80	4,3%	1268	783	485
	ALU	10326,64	20,3%	5654	3605	2049
	Ld-St	3499,98	6,9%	1947	1228	719
	Reg File	9535,44	18,7%	6094	2989	3105
	Functions	6609,92	13%	5200	2712	2488
	Mov	632,23	1,2%	409	261	148
	Branch	326,07	0,6%	264	136	128
	Ctrl Datapath	2596,83	4%	1926	898	1028
IT_Ctrl		3051,35	6%	1868	1080	788
Com Bus		2067,58	4,1%	1049	825	224
Debug		1476,96	2,9%	1061	694	367
Total		50639,49	100%	32140	18298	13842

TABLE 7.1 – Surface et répartition du nombre de cellules logiques (ST FDSOI28 LVT) dans la macro asynchrone

placement/routage (uniquement les surfaces des cellules sont prises en compte et additionnées). Le cœur de processeur du WUC représente à lui tout seul environ 26,8K cellules dont 14,9K cellules asynchrones pour une surface de $0,042mm^2$ sans le placement/routage. La macro asynchrone a été implémentée avec uniquement des cellules LVT PB16 pour avoir la consommation statique la plus faible possible.

La figure 7.4 présente le circuit envoyé en fabrication avec son anneau d'IO [2] ($2mm^2$) ainsi qu'un zoom sur la surface réelle du circuit. La macro asynchrone (nommée WUC core sur la photo) a une surface réelle de $0,081mm^2$ avec $267,38\mu m$ de large et $302,14\mu m$ de long. Cette photo illustre la répartition en surface de la macro asynchrone, du *cut* mémoire et du module de test/interface.

7.2 Test du circuit

7.2.1 Circuit

Le circuit a été fabriqué en technologie FDSOI 28nm ST Microelectronics et mis en boitier QFN36 qui contient 36 pattes d'entrées-sorties. La figure 7.5 présente la photo du circuit fabriqué.

2. Input Output

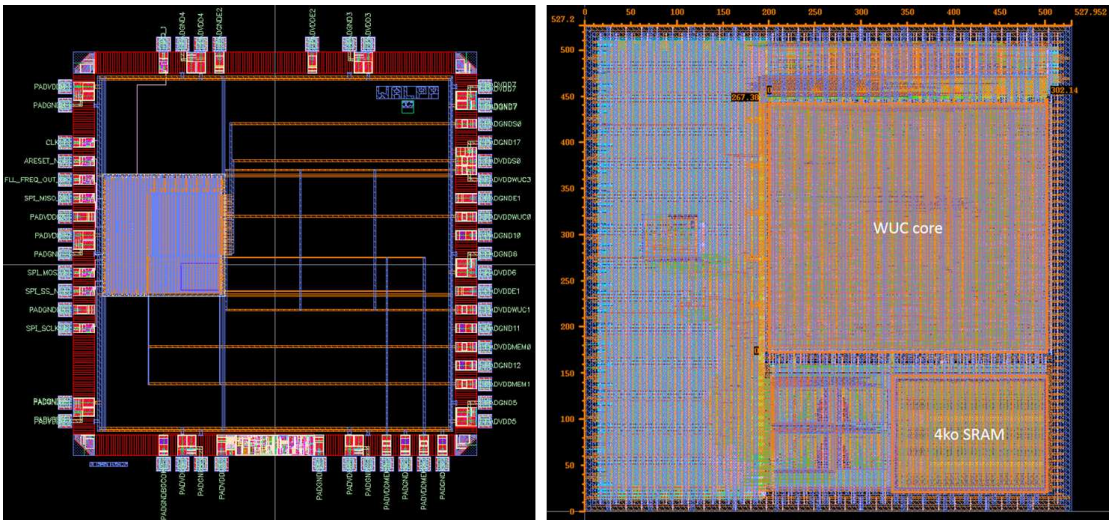


FIGURE 7.4 – Layout du circuit WALPP (à gauche : circuit avec IO ; à droite : zoom sur le circuit avec surface macro asynchrone et surface de la mémoire)

7.2.2 Carte de test

La carte de test de WALPP a été conçue de manière à ce qu'elle soit pilotable depuis un PC en passant par une liaison USB ou par une carte mère ayant les connectiques de type Arduino Uno. Pour la liaison avec le PC un circuit intégré de conversion entre l'USB et le SPI a été ajouté. Six tensions sont à générer pour WALPP et sont soit produites à partir de générateurs externes soit produites sur la carte grâce à des convertisseurs linéaires :

- Vdd (1V) : alimentation de la logique d'interface et de test de WALPP,
- Vdde (1,8V) : alimentation des *pads*,
- Vdd_mem (0-1,2V) : alimentation du domaine de puissance mémoire,
- Vdd_wuc (0-1,2V) : alimentation du domaine de puissance de la macro asynchrone,
- gnds_wuc et vdds_wuc : tension de *Back Biasing* du domaine de la macro asynchrone.

Du matériel a été ajouté sur la carte pour pouvoir mesurer la consommation des domaines de puissance de la macro asynchrone et de la mémoire. La figure 7.6 présente la carte qui a été fabriquée pour le test de WALPP. L'annexe C présente le schéma électrique de la carte et l'annexe D présente le *top* de la carte ainsi que sa sérigraphie.

7.3 Programmes de test et benchmark pour microcontrôleur de nœuds de capteurs communicants

Plusieurs programmes ont été mis en place pour tester les fonctionnalités du WUC et ses consommations. Nous allons présenter dans cette partie les programmes qui ont été utilisés ainsi que les *drivers* développés pour le circuit afin de le piloter depuis une application.

7.3.1 Drivers du circuit

Des *drivers* ont été développés pour que le circuit soit pilotable depuis un autre processeur. Pour le circuit il y a à chaque fois un fichier à inclure `wuc.h` et `walpp.h` qui contiennent la définition des registres, de leur adresse et leur déclaration. Les seconds fi-

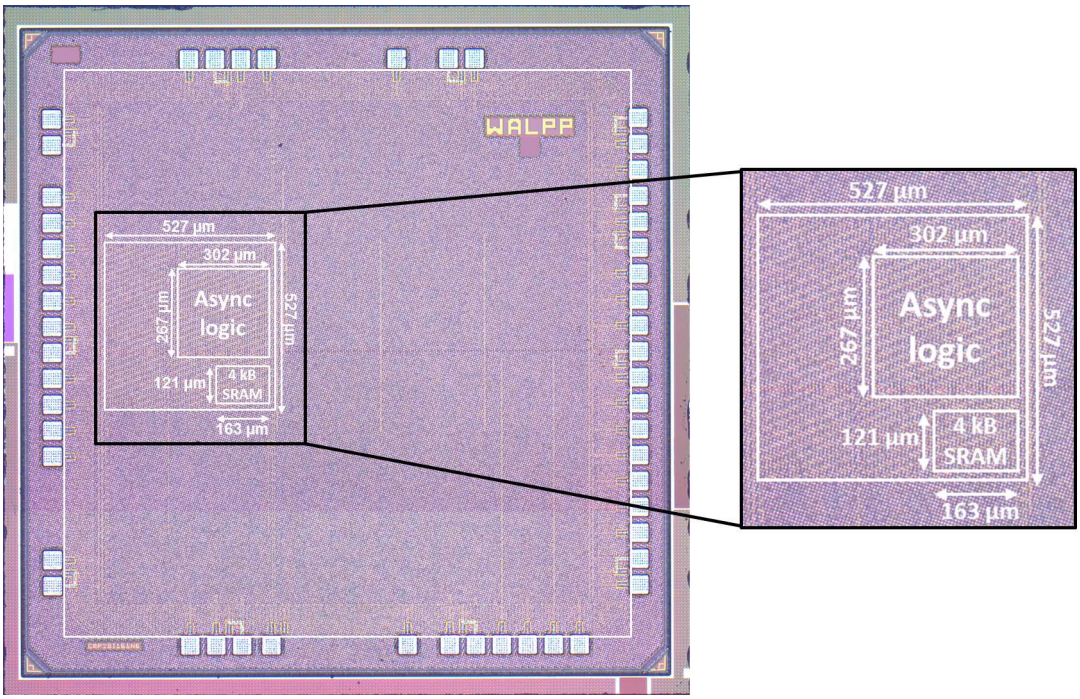


FIGURE 7.5 – Photographie du circuit WALPP. (gauche) circuit WALPP complet, (droite) zoom sur circuit WALPP

chiers `wuc_hal.h` et `walpp_hal.h` sont des couches d'abstraction du matériel. Il y a deux fonctions basiques dans `walpp_hal.h` pour venir lire et écrire dans le WUC, ce sont les fonctions suivantes.

```
1 void HAL_WALPP_FIFO_Write(WALPP_HandleTypeDef *walpp, uint32_t
    tram[]);
2
3 void HAL_WALPP_FIFO_Read(WALPP_HandleTypeDef *walpp, uint32_t
    tram[]);
```

Ensuite toutes les fonctions dans `wuc_hal.h` font appel à ces deux fonctions mais avec des commandes et des paramètres différents lors de l'appel. Ces *drivers* permettent ensuite un développement simple au niveau de l'application et des scénarios.

7.3.2 Programmes de test pour l'estimation de la consommation du processeur de réveil

Plusieurs programmes ont été développés pour tester à la fois toutes les fonctionnalités et pour tester le WUC sur des scénarios réalistes afin de tester les consommations du WUC.

Un premier programme entièrement développé en assembleur a été fait pour tester toutes les possibilités d'instructions du WUC ainsi que les possibilités de *debug* et la gestion de multiples interruptions arrivant en même temps. Ce programme n'a aucun intérêt mais permet juste de s'assurer qu'aucune faute n'est commise pendant l'exécution par rapport à ce qu'il doit exécuter. Pour s'assurer que le WUC exécute bien toutes les instructions, ce programme a été exécuté en mode pas à pas en simulation post-backend et le résultat de chaque instruction a été vérifié en allant lire le contenu des registres généraux. La trace

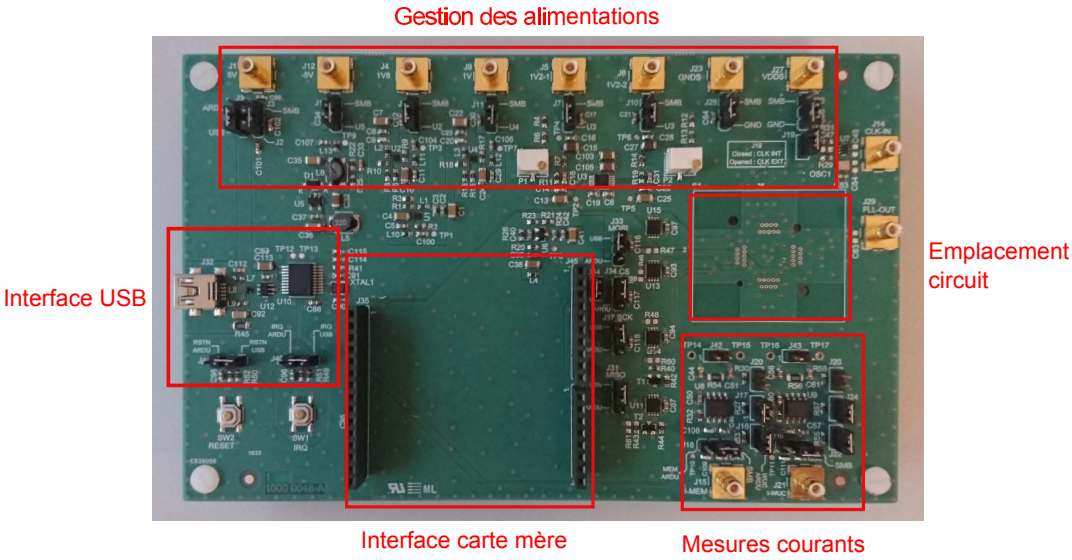


FIGURE 7.6 – Photographie de la carte de test du circuit

a été sauvegardée et comparée à la trace d'exécution que devait avoir le programme en simulation comportemental.

Un second programme a été développé cette fois-ci en language C pour tester les consommations et les performances du WUC sur un scénario probable qu'il pourrait exécuter. Ce programme consiste à effectuer des mesures sur des capteurs dont les données sont sur 8, 16 et 32 bits. Le WUC effectue 8 mesures par heure sur chaque capteur. Ces mesures brutes sont transformées en mesures réelles avec des calculs dont pourrait s'occuper le WUC et sont stockées dans des tableaux. Au bout de 8 mesures le WUC va extraire les valeurs minimales et maximales et calculer leur valeur moyenne. Ces valeurs sont stockées dans d'autres tableaux. Au bout de 24 heures, le WUC va vérifier que les valeurs minimales et maximales ne dépassent pas certaines limites, sinon il génère une alerte pour réveiller par exemple une radio et émettre un signal. Ce programme représente bien ce pourquoi le WUC a été développé. Ainsi, les résultats en consommation/performance et la quantité d'instructions ainsi que leur type seront réalistes. Ce programme a aussi été porté pour le Cortex M0+ pour avoir une vrai comparaison entre les deux processeurs.

7.3.3 Benchmark pour nœuds de capteurs communicants

Un benchmark existe aujourd'hui pour tester les microcontrôleurs basses consommation pour l'IoT. Le benchmark ULPBench™permet d'avoir une métrique des architectures low power pour des scénarios de l'IoT. Pour WALPP ce benchmark n'est pas utilisable car il manque beaucoup de périphériques dans l'architecture comme les *timers* mais aussi l'unité de *power management* pour effectuer du *power gating* et *clock gating* des différents modules et mettre le cœur et les périphériques dans le mode de consommation le plus bas pendant les phases de veille.

7.4 Performances et consommation de la plateforme de réveil et d'un circuit de comparaison

L'objectif de cette section est de présenter les résultats des simulations *post-backend* en termes de consommations et performances du circuit WALPP et de les comparer à l'état

de l'art pour en déduire les perspectives d'évolution du processeur et de la plateforme de réveil.

7.4.1 Estimation en performances de la plateforme de réveil

Le programme de test a été exécuté en simulation post-backend et les consommations statiques et dynamiques en ont été extraites. Ce programme fait exécuter au WUC 63479 instructions et génère 81507 accès à la mémoire ce qui implique 10915 instructions qui viennent lire en mémoire et 7113 instructions qui viennent écrire en mémoire. La mémoire et la logique asynchrone sont simulées à 0,6V. La figure 7.7 présente le diagramme en fonction du temps de l'exécution du programme. Le temps de réveil du WUC est ultra rapide puisqu'il se réveille fonctionnellement en 55,55ns pour charger le vecteur d'interruption et charge la première instruction à partir de 80,8 ns. À partir du moment ou l'instruction `ENDIT` est dans le décodeur, le processeur met 70,3 ns pour revenir à l'état de veille.

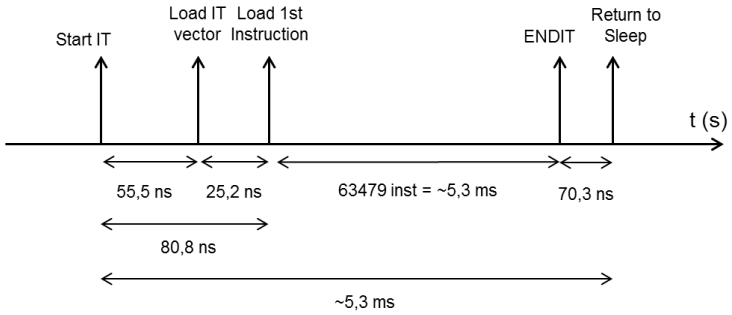


FIGURE 7.7 – Diagramme du temps de l'exécution du programme de test

Ces résultats permettent d'en déduire les performances du processeur. Il fonctionne en moyenne à 11,9 MIPS à 0,6V. La figure 7.8 montre le nombre d'instructions exécutées en fonction de leur temps d'exécution. les temps d'exécution des instructions vont de 49 ns à 115 ns.

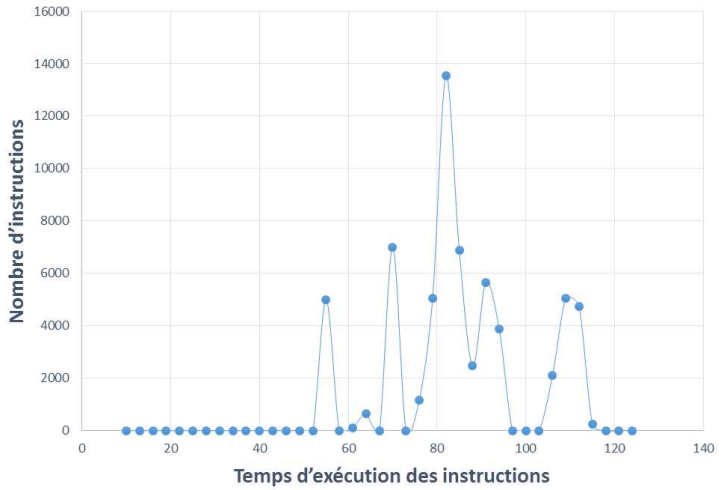


FIGURE 7.8 – Diagramme de distribution des instructions du programme en fonction du temps d'exécution des instructions

7.4.2 Estimation de la consommation de la plateforme de réveil

Sur ce circuit il est possible de mesurer indépendamment la consommation de la logique asynchrone et la consommation du domaine mémoire. Dans cette section les consommations statiques et dynamiques du circuit vont être analysées. La figure 7.9 montre la distribution de la puissance dans la logique asynchrone et le domaine mémoire. La logique asynchrone consomme une puissance due aux fuites de courant de $6,29\mu W$ et une puissance dynamique de $119\mu W$, ce qui correspond à $125\mu W$ de consommation de puissance moyenne en phase active. Cette puissance statique de $6,29\mu W$ correspond à la puissance consommée en mode *IDLE_LP* du tableau 4.3. La mémoire consomme quant à elle $6,6\mu W$ en phase de veille et $35,7\mu W$ de puissance moyenne en phase active. L'équation 7.1 permet de calculer l'énergie par instruction à ce point de tension.

$$E_{inst_{moy}} = \frac{P_{moy} \times T_{exec}}{Nb_{inst}} = \frac{P_{moy}}{MIPS} \tag{7.1}$$

Ces consommations impliquent une énergie par instruction moyenne $E_{inst_{moy}} = 10,5pJ/inst$ à 0,6V. Si la mémoire est comptée, l'énergie par instruction est de $E_{inst_{moy}} = 13,5pJ/inst$.

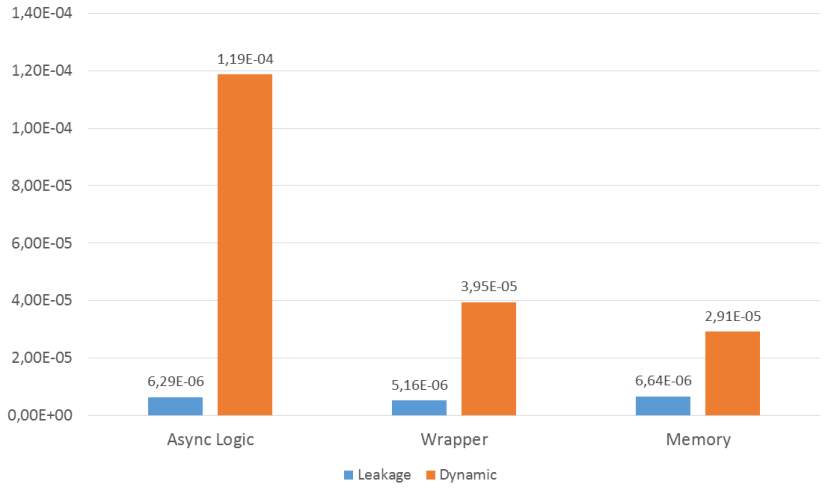


FIGURE 7.9 – Puissances dues aux au courants de fuites et puissance dynamique dans la macro asynchrone, le wrapper mémoire et la mémoire

En descendant dans la hiérarchie de la macro asynchrone, on peut extraire les puissances consommées des modules internes. La figure 7.10 présente la répartition de la consommation dans la macro asynchrone durant les phases de veille et les phases actives. Pendant la phase de veille, le cœur de processeur consomme $3,58\mu W$ pour 63% de la consommation statique. En phase active, le processeur consomme une puissance moyenne de $93,5\mu W$ (90% de la consommation) alors que le bus de communication en consomme $8,6\mu W$ (8%).

La macro asynchrone possède 18679 cellules asynchrones PB16 (57%), 13398 cellules standards PB16 (41%), 615 cellules standards PB0 (2%) et 29 cellules standards PB4. La figure 7.11 présente les puissances en fuites et dynamiques des cellules de la macro asynchrone. En terme de fuite de puissance, le peu de cellules PB0 consomment à elles seules 58% de la puissance alors que la consommation dynamique est majoritairement dues aux cellules asynchrones. Si l'on veut réduire la consommation statique il faudra donc

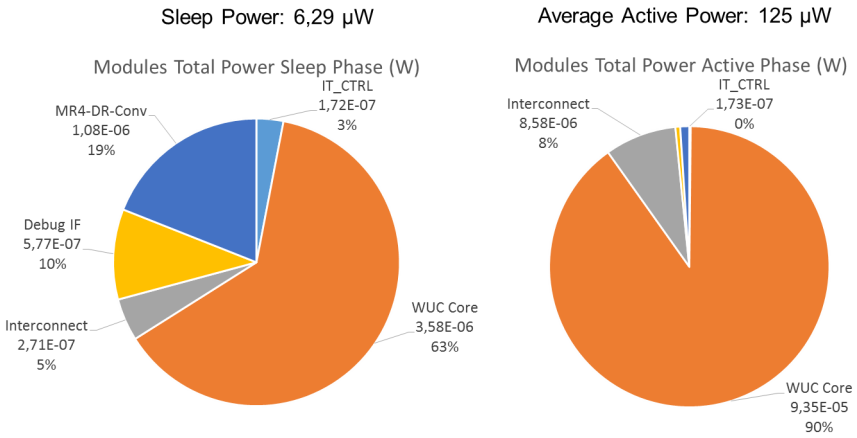


FIGURE 7.10 – Puissance consommée par les modules de la macro asynchrone

veiller à ne pas avoir trop de cellules PB0 et pour réduire la consommation dynamique il faudra réduire le nombre de cellules asynchrones.

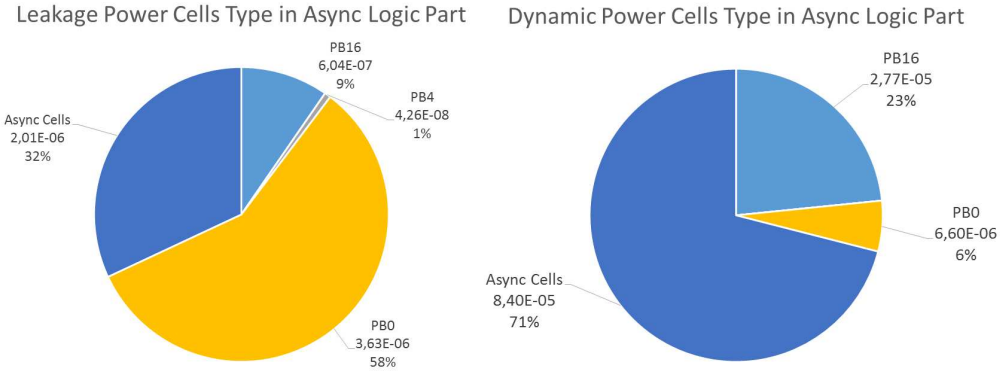


FIGURE 7.11 – Puissance consommée par les différents types de cellules dans la macro asynchrone

La figure 7.12 présente la consommation en puissance des fuites et en puissance moyenne des différents modules du cœur de processeur. En terme de puissance de fuites, le banc de registres, l'ALU, et l'unité de support aux fonctions sont les plus consommateurs. Lors de la phase active, ce sont majoritairement le contrôle du chemin de données, le décodeur, l'unité de PC et le banc de registres qui consomment le plus de puissance.

7.4.3 Étude du programme généré par le compilateur du WUC

Le programme de test pour le WUC et le Cortex M0+ a été compilé en optimisation "-O1". Malheureusement le compilateur du WUC n'intègre pas encore la possibilité d'utiliser les instructions `PUSH` et `POP`. L'analyse du code assembleur et des temps d'exécutions des instructions a permis de montrer que l'on pouvait gagner 4,1% d'énergie sur la tâche total en utilisant les instructions `PUSH` et `POP`.

À certain moment du programme, celui-ci a besoins d'exécuter un décalage arithmétique or cette instruction n'est pas implémentée dans le WUC, elle est donc émulée par d'autres instructions ce qui implique un coût supplémentaire par rapport au jeu d'instructions ARM. Une analyse du code a montré que l'implémentation de l'instruction de

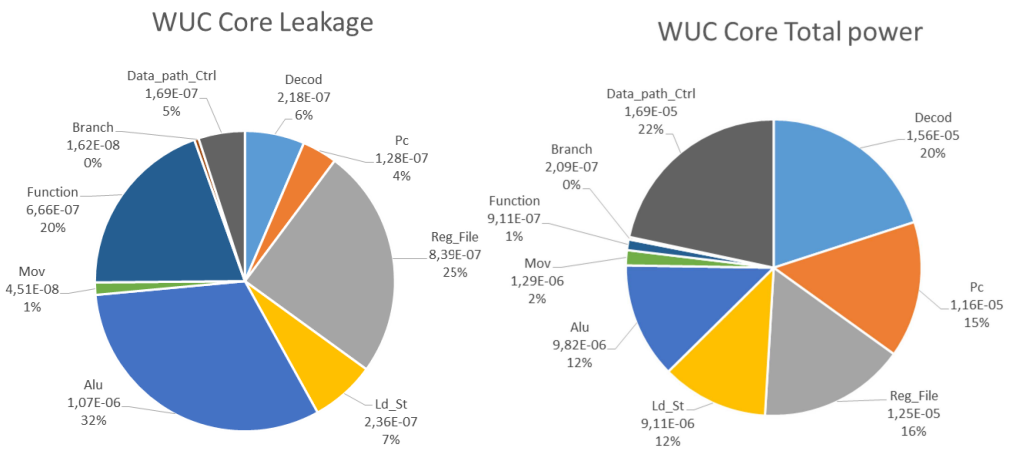


FIGURE 7.12 – Puissance consommée par les différents modules internes du cœur de processeur

décalage arithmétique permet de réduire l'énergie consommée par la tâche entière de 6,5%.

Ainsi, si le jeu d'instructions implémentait les décalages arithmétiques et si les instructions `PUSH` et `POP` étaient utilisées, alors 10% de l'énergie consommée par la tâche pourrait être sauvée.

Les instructions supplémentaires qui pourraient être ajoutées, ne modifient en rien l'énergie par instruction et les performances en terme de MIPS du processeur. Elles améliorent uniquement l'énergie consommée par la tâche.

La figure 7.13 présente l'estimation du nombre de MIPS qu'il faudrait atteindre pour avoir l'énergie par instruction voulue pour une architecture très peu modifiée et consommant en moyenne la même puissance active. Les mesures sur un circuit du laboratoire implémentant un Cortex M0+ dans la technologie FDSOI 28 nm LVT permettent d'atteindre pour les meilleurs circuits 5,46pJ/cycle soit 7,3pJ/inst à 0,6V pour une fréquence maximale de 111 MHz et une consommation dynamique de $5,56\mu W/MHz$ ou $7,44\mu W/MIPS$. Pour que le WUC soit meilleur énergétiquement parlant, il faudrait qu'il exécute du code entre 18 et 20 MIPS à 0,6V, contrairement au 12 MIPS actuel. Les différentes perspectives pour accélérer le processeur vont être vu en conclusion. Les circuits du Cortex M0+ de performance moyenne atteignent quant à eux des consommations de 8,9pJ/inst ou $8,9\mu W/MIPS$.

7.4.4 Positionnement vis-à-vis de l'état de l'art

Une comparaison avec l'état de l'art a été faite sur les principales caractéristiques des microcontrôleurs ultra basse consommation et dont un résumé est présenté tableau 7.2. Étant implémenté en logique asynchrone, le WUC peut voir sa tension varier de 0,3V à 1V. Nous avons pu simuler le circuit uniquement sur le point de tension 0,6V pour lequel le WUC atteint 11,9 MIPS et $10,5\mu W/MIPS$ pour une énergie de 10,5 pJ/inst. Il est de plus capable de se réveiller fonctionnellement en 55 ns, ce qui est très rapide par rapport aux microcontrôleurs de l'état de l'art et par rapport au circuit avec le Cortex M0+ du laboratoire qui se réveille du mode *IDLE_LP* en $2,3\mu s$ avec une fréquence de 10MHz soit un gain en temps de réveil de 42×. Nous n'avons pas pu mettre en place cependant la technique de *power gating* dans notre circuit avec des régulateurs d'énergie. Nous ne pouvons donc pas évaluer le temps de réveil du processeur d'un mode *OFF* ou l'alimentation est coupée, à un état d'activité. Or celui-ci serait nettement plus rapide que pour un processeur synchrone car dès que l'alimentation atteint une tension suffisante, aux alentours de 0,3V, le

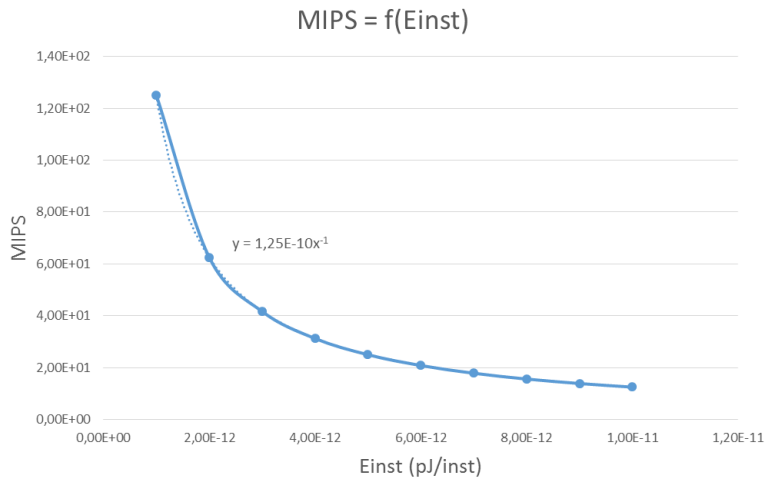


FIGURE 7.13 – Estimation du nombre de MIPS en fonction de l'énergie par instruction visée à 0,6V

processeur commence déjà à exécuter ses instructions. Plusieurs caractéristiques peuvent être utilisées pour comparer ce travail de thèse à l'état de l'art (Tableau 7.2) dont les principales sont la puissance dynamique, le point d'énergie minimum, la plage de fréquence, la puissance de rétention minimale et les temps de réveil depuis un mode de veille profond ou non. Néanmoins, n'ayant qu'un point de tension pour le WUC, il est très dur de comparer ce circuit à l'état de l'art, étant donné que les microcontrôleurs comme ceux d'Intel à VLSI 2016 [139], de ARM à l'ISSCC 2015 [120] et de Bol et al. [50] présentent des résultats sur des plages de tensions de 0,3V à 1V pour des énergies minimales de 11,7pJ/cycle pour une tension de 0,35V à 300KHz de fréquence maximale pour le microcontrôleur de ARM ou les 2,6pJ/cycle à 0,375V pour SleepWalker à 25MHz. Les résultats en énergie du WUC sont quand même meilleurs que ceux du processeur Cortex M0+ d'ARM à l'ISSCC 2015 puisqu'à 0,6V il fonctionne à une fréquence de 10MHz maximale pour une énergie de plus de 21pJ/cycle. Les concepteurs d'Intel arrivent à atteindre une énergie de plus de 26pJ/cycle à 0,6V pour une fréquence maximale à 100MHz et pour un point d'énergie minimal de 17,18pJ/cycle à 0,37V pour 3,5MHz de fréquence maximale. La comparaison la plus juste peut être faite avec le Cortex M0+ implémenté en FDSOI 28 nm dans notre laboratoire. Il consomme pour des circuits de performance moyenne 8,9pJ/inst à 0,6V pour une fréquence de 10MHz. Ce qui le rend meilleur en termes de consommation et performance mais pas en terme de réveil. De plus, pour le même mode de veille *IDLE_LP*, le circuit consomme $6,3\mu W$ alors que pour le même mode de consommation à la même tension, le circuit avec le Cortex M0+ consomme $41,6\mu W$, soit un gain en consommation de veille de 6×. Malheureusement nous n'avons pas encore les résultats des mesures silicium et il ne nous a pas été possible d'avoir les simulations en consommation du WUC pour une large gamme de tension. C'est pourquoi nous n'avons pas connaissance encore pour le WUC de son énergie minimale, et de ses performances maximales en augmentant sa tension d'alimentation et en utilisant les capacités de polarisation arrière.

7.4.5 Conclusion

L'architecture finale du circuit qui a été envoyé, ainsi que la carte de test développée pour extraire les résultats en termes de consommation et performance, les *drivers* du circuit et programmes de test pour obtenir ces valeurs ont été présentés dans ce chapitre.

Ce processeur nommé WUC implémenté en logique asynchrone QDI s'occupe des tâches régulières des nœuds de capteurs sans fil et possède un jeu d'instructions réduit 16-bit RISC pour effectuer ces tâches. Il est implémenté le plus simplement possible pour générer la surface minimale ($0,081mm^2$) en technologie FDSOI 28nm LVT. Une grande partie de la macro asynchrone a été développée avec des cellules PB16 (98%) pour minimiser les fuites de courant. Le WUC consomme 10,5pJ/inst à 0,6V pour une vitesse de 11,9MIPS. Sa consommation statique est par contre de $6,29\mu W$ sans power gating. Ce processeur est meilleur énergétiquement que celui d'Intel [139] ou ARM [120] dans l'état de l'art mais moins bon que l'implémentation du Cortex M0+ en technologie FDSOI 28nm. Pour une architecture du WUC consommant en moyenne en phase active $125\mu W$ il faudrait qu'il atteigne entre 18 et 20 MIPS pour être en dessous des 8,9 pJ/inst du Cortex M0+ en technologie FDSOI 28nm.

Les résultats silicium du WUC sont en train d'être obtenus et seront présentés à la soutenance.

Comme nous allons le voir dans les perspectives, plusieurs pistes peuvent être envisagées pour augmenter les performances, diminuer à la fois la consommation statique et dynamique et diminuer l'énergie consommée par tâche :

- ajouter les instructions manquantes qui sont aujourd'hui émulées par des instructions du jeu d'instructions et qui peuvent être appelées de nombreuses fois pendant le code comme l'instruction *arithmetic shift right* ;
- optimiser la logique des acquittements qui permettrait d'augmenter les performances de l'architecture actuelle sans trop d'effort et permettrait d'atteindre les 18 MIPS et être à un niveau équivalent au Cortex M0+ en terme d'énergie ;
- donner la possibilité au processeur de pouvoir charger/décoder et exécuter des instructions en même temps pour améliorer la vitesse du processeur tout en gardant une surface correcte et en gardant les possibilités de debug ;
- utiliser le protocole en multi-rail 4 pour le chemin de données afin de diminuer le nombre de transition sur les bus et réduire la consommation dynamique ;
- réduire la taille de l'ALU et de la *Function Unit* par la modification du jeu d'instructions et par une meilleure customisation de ces unités pour réduire la consommation statique ;
- éviter l'utilisation des cellules PB0 pour réduire la consommation statique (figure 7.11) ;
- Améliorer le compilateur pour utiliser au mieux les instructions `PUSH/POP` et pour utiliser au mieux le banc de registres car pour l'instant celui-ci est sous utilisé. S'il n'y a pas la possibilité d'utiliser les registres hauts (`R8-R14`) mieux qu'actuellement ou si les tâches programmées n'en n'ont pas besoins alors il serait envisageable de les supprimer ce qui permettrait par la même occasion d'avoir plus de bits pour l'encodage des instructions et donc d'ajouter d'autres instructions utiles.

	Ce travail	**Cortex M0+ Labo**	**Intel 2016** [139]	**ARM ISSCC** [120]	**SleepWalker** [50]	**TI CC2650** [156]
Technologie	FDSOI 28nm LVT	FDSOI 28nm LVT	14nm Tri-Gate CMOS	65nm CMOS	65nm CMOS	-
Vdd Cœur	0,3-1V	0,4-1V	0,308-1V	0,25-1,2V	0,32-0,48V	-
Forward Back Biasing	-0,2V-1,1V	-0,2V-1,1V	-	-	-	-
Logique	asynchrone	synchrone	synchrone	synchrone	synchrone	synchrone
CPU	16-bit RISC	32-bit ARM Cortex M0+	32-bit Intel Architecture	ARM Cortex M0+, Cortex M0, ARM Cortex A5	16-bit Compatible MSP430	ARM Cortex M3, ARM Cortex M0, Sensor Ctrl 16-bit
Surface logique	$0,081mm^2$	$0,140mm^2$	$0,076mm^2$	$0,154mm^2$	$0,42mm^2$	-
Chemin de données	32 bits	32 bits	32 bits	32 bits	16 bits	Sensor Ctrl : 16 bits
Mémoire	4 KB LVT SRAM	256 KB LVT SRAM	16KB Boot ROM, 64KB SRAM, 8KB DTCM, 8KB I$	8KB ULV SRAM, 16KB HV SRAM	18KB SRAM, 32*16bit I$	Sensor Ctrl : 2KB SRAM
Vdd Mémoire	0,6-1V	0,6-1V	0,297-1V	0,25-1,2V for ULV SRAM	1V	-
Périphériques	-	2 SPI, 2 UART, 2 Timer	2 SPI, 1UART, Timer, GPIO, I2C, PMU	RTC, PMU, GPIO, SPI, 128b AES	AVS, DC/DC, PMU, Clk gen, 2 SPI, Timer, UART, WDT, GPIO, TDC	TDC, ADC, CMP, 2 Timer, GPIO, Semaphore
Fréquences	11,9MIPS @ 0,6V	10MHz - 526MHz	500KHz-297MHz	>66MHz - 29KHz	25MHz	Sensor Ctrl : 24MHz max
Puissance Repos	$6,3\mu W$ @ 0,6V	$41,6\mu W$ @ 0,6V	-	4KB + CPU @ 0,25V : 80 nW (retention)	18KB + CPU @ $1,7\mu W$ (retention)	20KB SRAM + CPU + RTC @ $1\mu A$ (retention)
Puissance Dynamique min	$10,5\mu W/MIPS$ @ 0,6V	$8,9\mu W/MIPS$ ou $6,7\mu W/MHz$ @ 0,6V	$16,6\mu W/MHz$ @ 0,37V	$10\mu W/MHz$ @ 0,35V	$7\mu W/MHz$ @ 0,4V (SoC)	$8,2\mu A/MHz$
Emin (pJ/cycle)	10,5pJ/inst @ 0,6V, 11,9 MIPS	8,9pJ/inst ou 6,7 pJ/cycle @ 0,6V, 10MHz	17,18 @ 0,37V, 3,5MHz ; 26 @ 0.6V, 100MHz	11,7 @ 0,35V, 300KHz ; 21 @ 0.6V, 10,5MHz	2,6 @ 0,375V, 25MHz	-
Temps de réveil	55 ns @ 0,6V du mode *IDLE_LP*	23 cycles du mode *IDLE_LP* ou $2,3\mu s$ @ 10MHz	us du *Short Sleep*, ms du *Long Sleep*, s du *Deep Sleep*	$>1\mu s$	$33\mu s$ du mode *Stanby Power*	$151\mu s$ du mode Standby, $1015\mu s$ du mode Shutdown
Nb circuits mesurés	-	6	-	160	23	-
Nombre de domaines de puissance	2	2	5	14	3	10
Modes de puissance	*RUN*, *IDLE*	*RUN*, *IDLE*	*No Sleep*, *Short Sleep*, *Long Sleep*, *Deep Sleep*	*ALLON*, *LEASTON*, *ALLOFF*, *RET4KB*, *RET8KB*	*RUN*, *IDLE*, *STANDBY*	*ACTIVE*, *IDLE*, *STANDBY*, *SHUT-DOWN*
Unité de Debug	oui	oui	oui	oui	oui	oui

TABLE 7.2 – Comparaison des caractéristiques du processeur de réveil avec certains microcontrôleurs ultra basse consommation de l'état de l'art

Conclusion générale et perspectives

Conclusion

La consommation des circuits intégrés est aujourd'hui déterminante lors de leur conception surtout pour leurs applications dans l'*Internet of Things*. Ces applications sont très diverses et impliquent l'utilisation de ces circuits intégrés dans des environnements fluctuants pour des activités de fonctionnement excessivement variables. Il y a donc une énorme demande en flexibilité de ces systèmes que cela soit au niveau technologique et gestion de la puissance pour être à l'efficacité énergétique maximale dans n'importe quelles conditions d'utilisation ou que cela soit au niveau de l'architecture même de ces systèmes afin d'avoir toujours plus de services tout en consommant le moins possible. Dans le contexte des microcontrôleurs, ultra basse consommation, cette demande en flexibilité peut-être atteinte par des choix architecturaux du cœur de processeur et du système, du choix du type de conception synchrone ou asynchrone, de la technologie et des mémoires utilisées et par la gestion de la puissance interne. La plupart de ces systèmes fonctionnent sur rapport cyclique et se réveillent souvent pour effectuer des tâches très simples mais nécessitent parfois de grosses ressources en calcul pour des tâches irrégulières. C'est pourquoi un seul processeur généraliste pour effectuer toutes ces tâches peut se montrer beaucoup moins efficace en énergie qu'un système multi-processeurs dont l'un très efficace énergétiquement s'occupe des tâches courantes et l'autre, avec beaucoup plus de ressources en calcul, gère les tâches irrégulières et, par conséquent, consomme beaucoup plus.

Cette thèse s'est focalisée sur l'étude de la réduction de la consommation du microcontrôleur du nœud de capteurs communicant. Cela a consisté à effectuer des analyses des gains en consommation entre une architecture classiques de microcontrôleur et une architecture partitionnée entre un processeur principal et un processeur de réveil nommé Wake Up Controller (WUC) dans des scénarios pour nœuds de capteurs communicants. Cette étude a contribué à l'implémentation d'un processeur de réveil ultra basse consommation conçue en logique asynchrone pour les tâches régulières que doit exécuter un microcontrôleur. Son architecture est partagée en deux parties : une partie *Always Responsive* contenant le WUC et les périphériques de réveil ultra basse consommation et une partie *On Demand* avec le processeur principal pour les tâches irrégulières, calculatoires et de maintenance du nœud.

La modélisation des consommations d'un nœud de capteurs sans fil et du microcontrôleur a été faite. La modélisation de la consommation du microcontrôleur est basée sur les chiffres de la technologie FDSOI 28nm. Des simulations ont pu être faites dans les différentes phases applicatives du microcontrôleurs pour nœuds de capteurs. Ensuite, des simulations de scénarios complets ont été réalisées pour différentes activités. Ainsi, pour des scénarios à très faible activité (1 évènement/min) le WUC permettrait de réduire la consommation moyenne du microcontrôleur de 14,5% alors que pour une activité moyenne (10 évènements/s) le WUC permettrait de réduire la consommation moyenne du microcontrôleur de 76%.

Le processeur de réveil a été spécifié pour n'exécuter que des routines d'interruption et

pour effectuer des transferts de données de taille 8, 16 et 32 bits provenant des capteurs ou de la radio de réveil. Il est capable aussi d'effectuer quelques calculs sur les données brutes récupérées et d'évaluer des conditions dans le cadre d'exécutions de machines d'états dû à son modèle de programmation. Ce processeur possède un jeu d'instructions 16-bit contenant des instructions *load-store*, arithmétiques et logiques, de branchement et de support aux fonctions. Il a été implémenté en logique asynchrone QDI double rail quatre phases en technologie FDSOI 28 nm. La logique asynchrone permet d'avoir une large gamme de tension de fonctionnement, une consommation lissée, un réveil de la logique presque instantané, et est très adaptée au fonctionnement sur évènements non déterministes puisque celle-ci est intrinsèquement en attente d'évènements appelés aussi jetons et se réveille instantanément à l'arrivée de ceux-ci.

La plateforme de réveil est constituée dans cette première version du WUC, de son contrôleur d'interruption (huit interruptions possibles), du bus de communication, de sa mémoire de programme et données et de l'unité de programmation et debug. Le cœur de processeur est débogable, en d'autres termes il est possible de stopper l'exécution du cœur, de poser des points d'arrêt dans le code, et de faire du pas à pas sur l'exécution du code. La plateforme a été fabriquée en technologie UTBB FDSOI 28 nm LVT. Le cœur de processeur possède une empreinte matérielle de 26,8K cellules et un total de 32,1K cellules pour la macro asynchrone et une surface réelle de $0,081mm^2$. Les simulations *post-backend* à 0,6V montrent que le WUC est capable d'effectuer le premier chargement en mémoire en 55 ns et de charger la première instruction en 80 ns. Il a une vitesse de 11,9 MIPS et est capable de se remettre en veille en 70 ns. Sur les programmes de tests et grâce au modèle physique du circuit, la consommation de la macro asynchrone est estimée à $125\mu W$ en mode actif et $6,29\mu W$ en mode veille sans utiliser de technique de power gating ni de polarisation arrière. La consommation dynamique peut être traduite en énergie par instruction, elle est de $10,5pJ/inst$ pour le WUC à 0,6V. En se concentrant sur le cœur de processeur, celui-ci consomme $3,58\mu W$ en phase de veille et $93,5\mu W$ en phase active.

Ainsi pour conclure sur les contributions, cette thèse a permis de :

- montrer les gains potentiels en consommation moyenne d'une architecture partitionnée de microcontrôleur pour les scénarios de nœuds de capteurs communicants ;
- créer les spécifications du processeur de réveil et de la plateforme de réveil ;
- spécifier l'architecture et implémenter une partie de la plateforme de réveil en logique asynchrone ;
- mettre en place une méthodologie de conception mixte asynchrone-synchrone ;
- montrer que les résultats sont corrects au regard de l'état de l'art mais que certaines améliorations restent à faire pour avoir des gains conséquents en énergie par rapport au Cortex M0+ pour une même technologie.

Amélioration et perspectives

Nous allons voir maintenant toutes les améliorations possibles et les perspectives d'évolution de la plateforme en passant par le jeu d'instructions du WUC, l'architecture, la microarchitecture, la gestion de la puissance du système, la partie logiciel/compilation et enfin l'architecture de la plateforme finale visée.

Jeu d'instructions

Quelques instructions sont manquantes afin de réduire l'énergie consommée par une tâche. Comme il a pu être vu, certaines instructions qui n'existent pas dans le jeu sont

émulées par des fonctions avec des instructions du jeu et utilisées par le compilateur quand cela est nécessaire. Cela engendre un coût en énergie non négligeable si celles-ci sont utilisées à de nombreuses reprises. Les instructions à ajouter sont les décalages arithmétiques à droite (*Arithmetic Shift Right*) avec base registre et immédiat et les décalages à gauche et à droite à base registre.

Une autre optimisation à explorer est la pertinence d'avoir 16 registres dans le banc. Les différents programmes qui ont été compilés utilisent que très peu les registres `R8` à `R14`. Cela est peut-être dû à une mauvaise gestion des registres par le compilateur ou au fait que les applications qui s'exécutent sur le WUC n'ont pas besoin de ces 16 registres. Si les 8 registres hauts `R8` à `R14` peuvent être supprimés alors les registres peuvent être encodés sur trois bits dans l'instruction. Cela permettrait d'avoir plus de place dans l'*opcode* de l'instruction pour ajouter des instructions et cela permettrait de diminuer la consommation statique (25% actuellement) et dynamique (16% actuellement) du banc de registres ainsi que sa surface.

Il faudrait évaluer aussi le coût de l'ajout de l'instruction de multiplication. Si cela ne représente pas trop de matériel alors il serait intéressant de l'ajouter pour diminuer l'énergie d'une tâche.

Architecture

Plusieurs améliorations architecturales sont à implémenter pour atteindre une meilleure consommation en énergie que le Cortex M0+ mais aussi obtenir de meilleures performances. Effectivement nous avons pu voir que pour de très faibles modifications du processeur il suffisait d'atteindre 18 à 20 MIPS de performance à 0,6V pour être en dessous de l'énergie consommée par instruction du Cortex M0+. Dans un second temps, pour obtenir des performances accrues et sûrement une meilleure énergie par instruction, il serait envisageable de séparer la mémoire de programme et de données et de les mettre au plus près du processeur pour pouvoir en même temps charger les instructions et travailler sur les données. Cette modification impactera fortement l'architecture du WUC et son fonctionnement. De plus il faut pouvoir garder les possibilités de debug et donc avoir accès aux mémoires et au cœur à tout moment. Il serait donc envisageable d'avoir des mémoires à double port : un port pour le bus de communication principal (debug et programmation) et un port pour la communication directe avec le cœur.

Afin de réduire le nombre d'accès à la mémoire pour la lecture des instructions, une ligne de cache de 16 bits peut être ajoutée, étant donné que les instructions sont sur 16 bits et qu'une ligne de mémoire est sur 32 bits.

La plateforme de réveil devra aussi implémenter des périphériques tel que des *timers*, une unité de gestion de la puissance et des modules spéciaux lorsque la plateforme *On Demand* sera couplée à la plateforme de réveil. En effet lorsque le WUC sera couplé à un processeur principal, l'accès à la mémoire partagée pour les données devra être arbitré par un sémaphore binaire pour avoir une cohérence des données et pour avoir une synchronisation inter-processeur.

Microarchitecture

Au niveau de la microarchitecture la première amélioration à effectuer se concentre sur la mémoire. Il faudra avoir des mémoires capables de fonctionner de 0,3V à 1V et de manière asynchrone obligatoirement. C'est à dire que la mémoire doit elle même générer les acquittements de lecture et d'écriture pour éviter d'avoir ces chaînes de délais et pour être affranchi du respect des temps de maintien, de *setup* et d'accès à celles-ci. Si les mémoires

sont au plus prêt du cœur, il faut aussi qu'elles possèdent deux ports d'accès (un port pour le WUC et un port pour le bus de communication principal pour la programmation et le debug). De plus une mémoire avec un mode très basse consommation serait un plus pour obtenir des consommations statiques très faibles lors des longues phases de veille (<80nW dans l'état de l'art [120]) avec aussi la possibilité d'utiliser la polarisation arrière de la technologie FDSOI 28nm.

Afin de réduire la consommation dynamique du processeur et du bus de communication, le chemin de données devrait être implémenté en multi-rail 4 [40].

Pour optimiser encore plus la consommation il serait bien de faire passer les données provenant des unités d'exécution les plus utilisées par le moins de logique possible, au niveau du contrôle du chemin de données.

De plus, la surface de l'ALU pourrait être optimisée en la customisant davantage. La surface de la *Function Unit* pourrait être aussi largement réduite en revoyant le fonctionnement et les possibilités des instructions `PUSH/POP`.

Gestion de la puissance et des horloges

Au niveau du *power management* il y a énormément de perspectives car durant cette thèse nous n'avons pas pu mettre en place une unité de contrôle de la puissance avec différents modes de puissance et les possibilités de *power gating* et de variation de la tension automatique. La figure 7.14 présente un des objectifs en terme de gestion de la puissance et d'horloge à atteindre sur les prochains circuits. Un domaine *Always On* (PD_SYSTEM) contiendrait le gestionnaire de puissance, de fréquence et de reset. Il contiendrait la gestion des modes de puissance et viendrait effectuer du *clock gating* et du *power gating* sur les différents domaines de puissance. Il pourrait être capable aussi d'effectuer de la DVS [3] (et non DVFS [4] puisque l'implémentation est faite en asynchrone) sur les domaines de puissance du WUC et de ses mémoires en fonction du rapport cyclique de l'application, pour être toujours au point d'énergie minimum. En effet, Intel [139] dans VLSI 2016 a bien montré que le point d'énergie minimal se déplaçait en fonction du rapport cyclique. Il suffirait alors de programmer un rapport cyclique et le module DVS enverrait les commandes nécessaires au régulateur de tension pour être toujours au point d'énergie minimal. Le régulateur viendrait faire varier les tensions d'alimentation et la tension de FBB pour suivre le MEP. Le régulateur de tension numérique (*Digital Voltage Regulator*) devra contenir différents types de régulateur pour être à l'efficacité de conversion maximale en fonction du point de tension demandé. La partie *On Demand* est quant à elle la plus part du temps éteinte et est dans son propre domaine d'alimentation *VD_OnDemand*. Le module de gestion de la puissance et des horloges est aussi capable de distribuer les horloges provenant des oscillateurs aux périphériques qui en ont besoin comme les *timers* et le SPI. L'implémentation de la partie *Always Responsive* étant en logique asynchrone, il n'y a pas besoins de PLL/FLL pour cadencer la logique. Ceci simplifie beaucoup le module de gestion de la puissance et des horloges.

Le circuit aurait alors une multitude de mode de puissance et de possibilité de *clock-gating* et *power-gating*. Une option qui pourrait être ajoutée à l'instruction `ENDIT` est sa possibilité de programmer automatiquement le mode de puissance du circuit une fois l'instruction exécutée. Cela permettrait d'accélérer la mise en veille automatique des différentes parties du circuit dès qu'il n'y a plus rien a exécuter.

3. Dynamic Voltage Scaling
4. Dynamic Voltage and Frequency Scaling

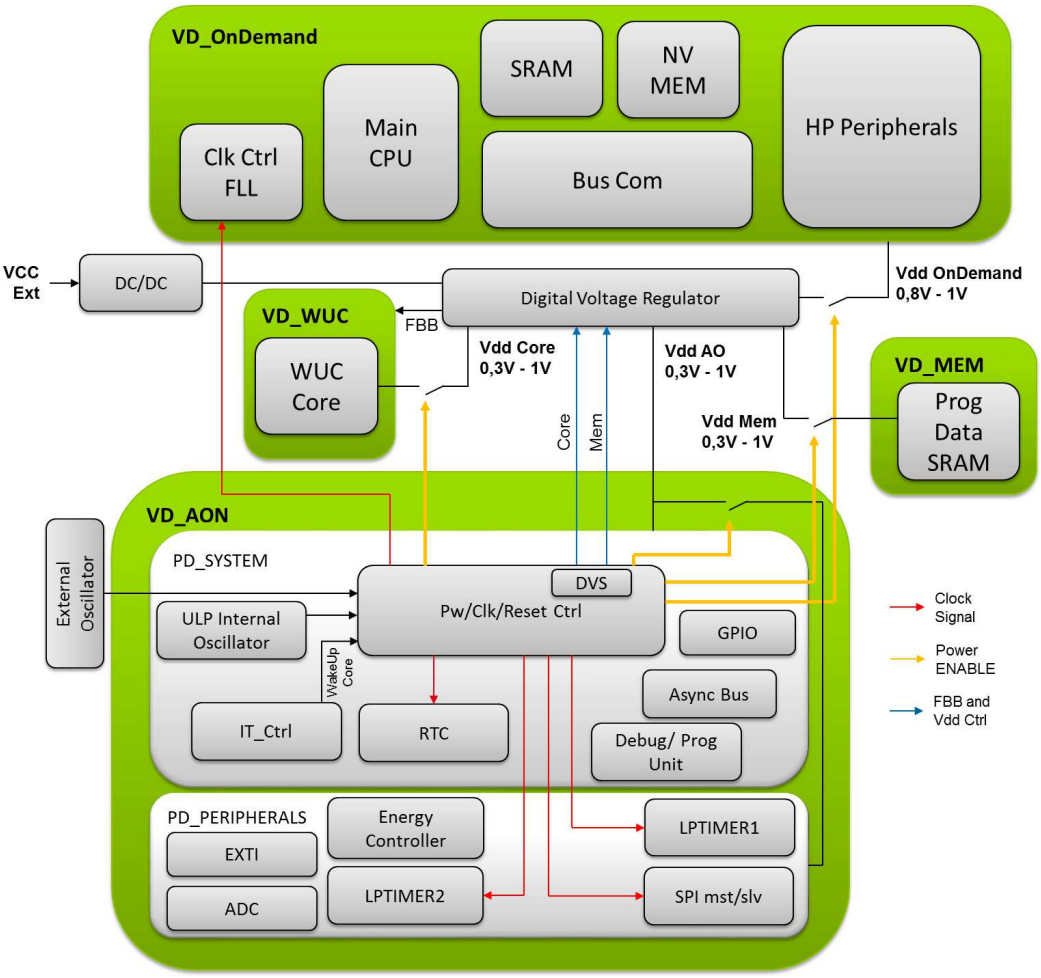


FIGURE 7.14 – Architecture de partitionnement des domaines de puissance/fréquence et de sa gestion

Software/compilation/outils de développement/simulations

Toutes ces modifications architecturales et de spécifications vont avoir un impact sur le compilateur, qu'il faudra modifier en étroite collaboration avec les personnes travaillant dessus. Le système devenant de plus en plus compliqué et mélangeant partie analogique/-numérique et potentiellement de la RF avec la *WUR* il faudra sûrement mettre en place un modèle SystemC de la plateforme pour la simuler entièrement afin de détecter en amont les problèmes et de pouvoir tester différentes applications et fonctionnalités. Il sera alors possible de tester du code s'exécutant sur le WUC gérant le protocole de communication de la WUR par exemple tout en ayant modélisé le *Digital Base Band* de celle-ci et la réception des données à travers elle. De plus lorsque le processeur principal sera choisi, et une fois les deux plateformes couplées, il serait intéressant d'intégrer un RTOS dans le processeur principal pour faciliter le développement des tâches dans le microcontrôleur et sa maintenance.

Perspective d'évolution de la plateforme

Ayant pris conscience des problèmes de la plateforme actuelle, nous pouvons présenter l'objectif de la prochaine architecture du système intégrant toutes les modifications et

améliorations citées plus haut. Cette architecture est présentée figure 7.15. On peut y voir les modifications architecturales du cœur avec les ports pour les instructions et les données, la réduction du nombre de registre dans le banc, l'ajout des périphériques *WUR*, *Timer*, *ADC*, *Semaphore* et les mémoires doubles ports. La partie *On Demand* serait maître sur le bus de communication de la partie *Always Responsive*. Les requêtes viendraient soit de la *Debug Unit*, soit du port de données du processeur principal de la partie *On Demand*.

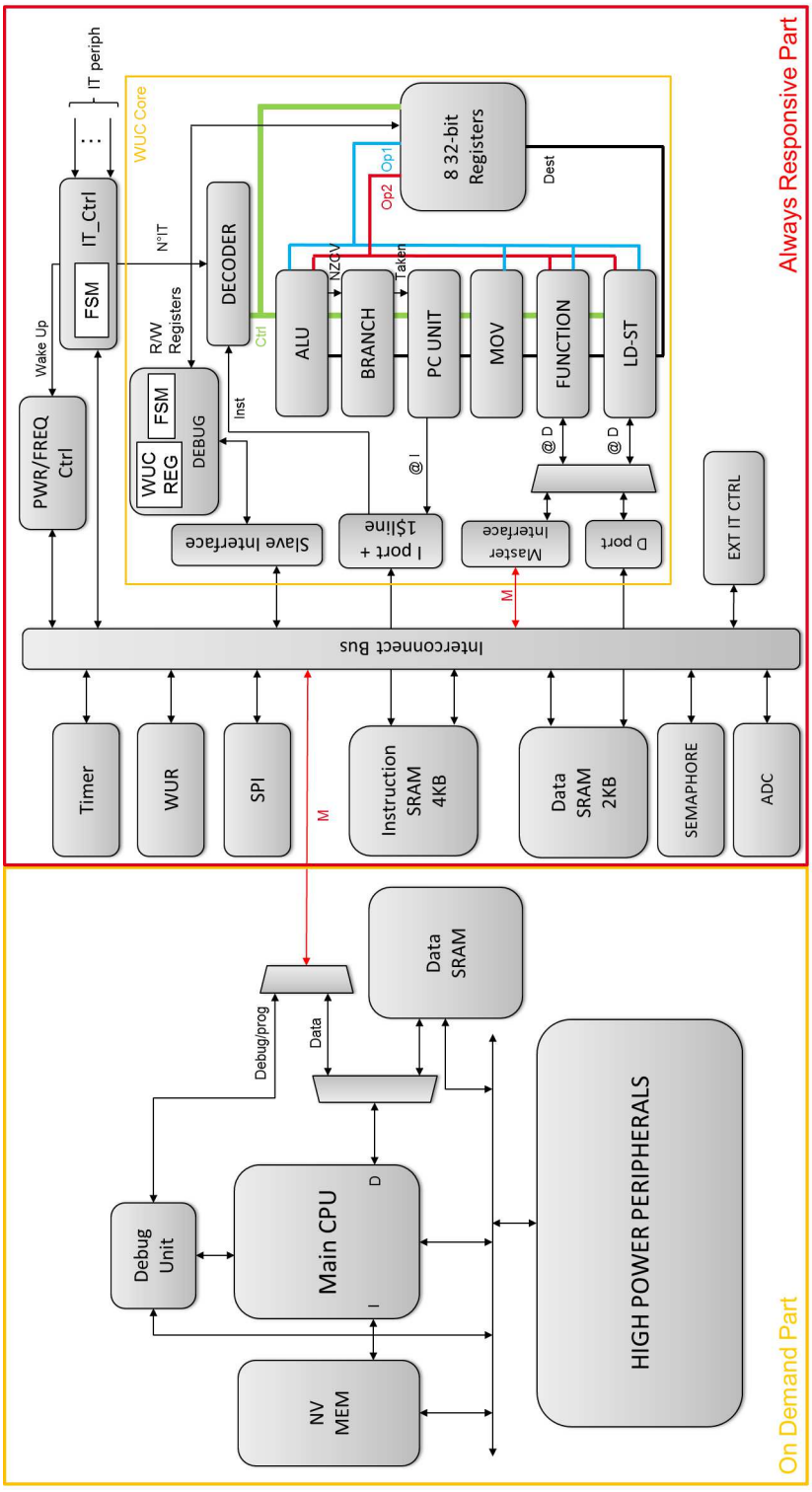


FIGURE 7.15 – Architecture possible de la future plateforme avec les améliorations

Références Bibliographiques

[1] Amazon prime air. http://www.amazon.com/b?node=8037720011. Accessed : 2016-18-04.

[2] Business insider uk. http://uk.businessinsider.com/internet-of-everything-2015-bi-2014-12?r=US&IR=T.

[3] Business insider uk. http://uk.businessinsider.com/how-the-internet-of-things-market-will-grow-2014-10?r=US&IR=T.

[4] Cityzen sciences. http://www.cityzensciences.fr/. Accessed : 2016-20-04.

[5] Compteur linky erdf. http://www.erdf.fr/linky-le-compteur-communicant-derdf. Accessed : 2016-20-04.

[6] Delta drone. http://www.deltadrone.com/fr/.

[7] Destinhaus. http://destinhaus.com/. Accessed : 2016-18-04.

[8] Diagramme de ragone. http://dspace.mit.edu/bitstream/handle/1721.1/73637/10-391j-spring-2005/contents/lecture-notes/0303se05toolbox5.pdf. Accessed : 2016-10-05.

[9] Drones parrot. http://www.parrot.com/fr/drones/. Accessed : 2016-18-04.

[10] Ehang 184. http://www.ehang.com/ehang184.

[11] Google glass. https://developers.google.com/glass/. Accessed : 2016-20-04.

[12] Heliatek the future is light. http://www.heliatek.com/en/heliafilm/technology. Accessed : 2016-09-05.

[13] Ibm. http://www.research.ibm.com/articles/brain-chip.shtml.

[14] Itead studio soil moisture. http://wiki.iteadstudio.com/Moisture_Sensor. Accessed : 2016-26-04.

[15] Lora alliance. https://www.lora-alliance.org/The-Alliance/About-the-Alliance. Accessed : 2016-22-04.

[16] Panasonic. http://news.panasonic.com/global/press/data/2014/04/en140410-4/en140410-4.html. Accessed : 2016-09-05.

[17] Principe de fonctionnement cellule photovoltaïque. http://www.connaissancedesenergies.org/fiche-pedagogique/solaire-photovoltaique. Accessed : 2016-09-05.

[18] Project pole. `http://projectpole.com/`. Accessed : 2016-20-04.

[19] Pulse sensor. `http://pulsesensor.com/`. Accessed : 2016-26-04.

[20] Recon technology. `http://www.reconinstruments.com/products/jet/`. Accessed : 2016-20-04.

[21] Samsung. `http://www.samsung.com/fr/galaxy/gear-s2/features/`. Accessed : 2016-20-04.

[22] Sigfox. `http://www.sigfox.com/`.

[23] Sony. `http://www.sonymobile.com/fr/products/smartwear/smartwatch-3-swr50/`. Accessed : 2016-20-04.

[24] Tiempo secure. `http://www.tiempo-secure.com/product/secure-ip-platform/`.

[25] Z-wave products. `http://www.z-wave.com/products`.

[26] (2010) 802.11p IEEE Standard for Information technology. Specific requirements–part 11 : Wireless lan medium access control (mac) and physical layer (phy) specifications amendment 6 : Wireless access in vehicular environments. In *Local and metropolitan area networks*, 2010.

[27] (2011) 802.11s IEEE Standard for Information Technology. Local and metropolitan area networks–specific requirements part 11 : Wireless lan medium access control (mac) and physical layer (phy) specifications amendment 10 : Mesh networking. In *Telecommunications and information exchange between systems*, 2011.

[28] Advancer Technologies. *Muscle Sensor v3*, 02 2013.

[29] Allegro Microsystems. *ACS712 Fully Integrated, Hall Effect-Based Linear Current Sensor with 2.1 kVRMS Voltage Isolation and a Low-Resistance Current Conductor*, 1 2007. Rev. 7.

[30] ARM. *ARMv6-M Architecture Reference Manual*, 09 2010.

[31] ARM. *ARMv7-M Architecture Reference Manual*, 02 2010.

[32] ARM. *ARM Cortex-M0+ Technical Reference Manual*, 12 2012. Rev. r0p1.

[33] ARM. *ARM Cortex-M4 Processor Technical Reference Manual*, 02 2015. Rev. r0p1.

[34] ATMEL. *AT42QT1010 Single-key QTouch Touch Sensor IC Datasheet*, 05 2013.

[35] ATMEL. *SAM D20 Datasheet Complete*, 2013.

[36] Atmel. *Datasheet AT86RF233*, 7 2014.

[37] ATMEL. *SAM D21E / SAM D21G / SAM D21J Datasheet Complete*, 03 2016.

[38] B. Atwood, B. Warneke, and K. S. J. Pister. Preliminary circuits for smart dust. In *Mixed-Signal Design, 2000. SSMSD. 2000 Southwest Symposium on*, pages 87–92, 2000.

[39] C. Bachmann, G. J. van Schaik, B. Busze, M. Konijnenburg, Y. Zhang, J. Stuyt, M. Ashouei, G. Dolmans, T. Gemmeke, and H. de Groot. 10.6 a 0.74v 200 uw multi-standard transceiver digital baseband in 40nm lp-cmos for 2.4ghz bluetooth smart / zigbee / ieee 802.15.6 personal area networks. In *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, pages 186–187, Feb 2014.

[40] W. J. Bainbridge and S. B. Furber. Delay insensitive system-on-chip interconnect using 1-of-4 data encoding. In *Asynchronus Circuits and Systems, 2001. ASYNC 2001. Seventh International Symposium on*, pages 118–126, 2001.

[41] E. Beigne, J. F. Christmann, A. Valentian, O. Billoint, E. Amat, and D. Morche. Utbb fdsoi technology flexibility for ultra low power internet-of-things applications. In *2015 45th European Solid State Device Research Conference (ESSDERC)*, pages 164–167, Sept 2015.

[42] E. Beigne, A. Valentian, B. Giraud, O. Thomas, T. Benoist, Y. Thonnart, S. Bernard, G. Moritz, O. Billoint, Y. Maneglia, P. Flatresse, J.P. Noel, F. Abouzeid, B. Pelloux-Prayer, A. Grover, S. Clerc, P. Roche, J. Le Coz, S. Engels, and R. Wilson. Ultra-wide voltage range designs in fully-depleted silicon-on-insulator fets. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2013*, pages 613–618, March 2013.

[43] O. Berder and O. Sentieys. Powwow : Power optimized hardware/software framework for wireless motes. In *Architecture of Computing Systems (ARCS), 2010 23rd International Conference on*, pages 1–5, Feb 2010.

[44] C. Bernier, F. Hameau, G. Billiot, E. de Foucauld, S. Robinet, D. Lattard, J. Durupt, F. Dehmas, L. Ouvry, and P. Vincent. An ultra low power soc for 2.4ghz ieee802.15.4 wireless communications. In *Solid-State Circuits Conference, 2008. ESSCIRC 2008. 34th European*, pages 426–429, Sept 2008.

[45] F. Berthier, E. Beigne, F. Heitzmann, O. Debicki, and O. Sentieys. Coeur de processeur asynchrone et microcontrôleur de noeud de capteur communicant comportant un tel coeur de processeur. Brevet, 2016.

[46] F. Berthier, E. Beigne, P. Vivet, and O. Sentieys. Fresh ideas : Asynchronous wake up controller for wsn's microcontroller : Power simulation and specifications. In *IEEE 21th International Asynchronous Circuits and Systems (ASYNC)*, May 2015.

[47] F. Berthier, E. Beigne, P. Vivet, and O. Sentieys. Power gain estimation of an event-driven wake-up controller dedicated to wsn's microcontroller. In *IEEE 13th International New Circuits and Systems Conference (NEWCAS)*, pages 1–4, June 2015.

[48] Florent Berthier, Edith Beigne, Frédéric Heitzmann, Olivier Debicki, Jean-Frédéric Christmann, Alexandre Valentian, Olivier Billoint, Esteve Amat, Dominique Morche,

Soundous Chairat, and Olivier Sentieys. UTBB FDSOI suitability for iot applications : Investigations at device, design and architectural levels. *Solid-State Electronics*, pages –, 2016.

[49] J. P. Bodanese, G. M. de Araújo, G. V. Raffo, and L. B. Becker. Rbesp : Reliable and best effort stack protocol for uav collaboration with wsn. In *2014 12th IEEE International Conference on Industrial Informatics (INDIN)*, pages 382–387, July 2014.

[50] D. Bol, J. De Vos, C. Hocquet, F. Botman, F. Durvaux, S. Boyd, D. Flandre, and J. D. Legat. Sleepwalker : A 25-mhz 0.4-v sub-mm2 7-uw/mhz microcontroller in 65-nm lp/gp cmos for low-carbon wireless sensor nodes. *IEEE Journal of Solid-State Circuits*, 48(1) :20–32, Jan 2013.

[51] Bosch. *BMP180 Digital Pressure Sensor*, 04 2013. Rev. 2.5.

[52] Bosch. *BMP280 Digital Pressure Sensor*, 10 2015. Rev. 1.

[53] H. Bottner, J. Nurnus, A. Gavrikov, G. Kuhner, M. Jagle, C. Kunzel, D. Eberhard, G. Plescher, A. Schubert, and K. H. Schlereth. New thermoelectric components using microsystem technologies. *Journal of Microelectromechanical Systems*, 13(3) :414–420, June 2004.

[54] M. R. Brust and B. M. Strimbu. A networked swarm model for uav deployment in the assessment of forest environments. In *Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP), 2015 IEEE Tenth International Conference on*, pages 1–6, April 2015.

[55] CdS Photoconductive cells. *GL5528*.

[56] J. H. Chang, S. Diao, R. M. Kumarasamy, and M. Je. 32khz mems-based oscillator for implantable medical devices. In *2011 International Symposium on Integrated Circuits*, pages 246–249, Dec 2011.

[57] K. L. Chang, J. S. Chang, B. H. Gwee, and K. S. Chong. Synchronous-logic and asynchronous-logic 8051 microcontroller cores for realizing the internet of things : A comparative study on dynamic voltage scaling and variation effects. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 3(1) :23–34, March 2013.

[58] Y. Chee, A. Niknejad, and J. Rabaey. A 46 In *2006 Symposium on VLSI Circuits, 2006. Digest of Technical Papers.*, pages 43–44, 2006.

[59] Yuen-Hui Chee, M. Koplow, M. Mark, N. Pletcher, M. Seeman, F. Burghardt, D. Steingart, J. Rabaey, P. Wright, and S. Sanders. Picocube : A 1cm3 sensor node powered by harvested energy. In *Design Automation Conference, 2008. DAC 2008. 45th ACM/IEEE*, pages 114–119, June 2008.

[60] G. Chen, H. Ghaed, R. u. Haque, M. Wieckowski, Y. Kim, G. Kim, D. Fick, D. Kim, M. Seok, K. Wise, D. Blaauw, and D. Sylvester. A cubic-millimeter energy-autonomous wireless intraocular pressure monitor. In *2011 IEEE International Solid-State Circuits Conference*, pages 310–312, Feb 2011.

[61] J. F. Christmann, E. Beigné, C. Condemine, N. Leblond, P. Vivet, G. Waltisperger, and J. Willemin. Bringing robustness and power efficiency to autonomous energy harvesting microsystems. In *Asynchronous Circuits and Systems (ASYNC), 2010 IEEE Symposium on*, pages 62–71, May 2010.

[62] J. F. Christmann, E. Beigné, C. Condemine, and C. Piguet. Event-driven asynchronous voltage monitoring in energy harvesting platforms. In *New Circuits and Systems Conference (NEWCAS), 2012 IEEE 10th International*, pages 457–460, June 2012.

[63] J. F. Christmann, E. Beigné, C. Condemine, and J. Willemin. An innovative and efficient energy harvesting platform architecture for autonomous microsystems. In *NEWCAS Conference (NEWCAS), 2010 8th IEEE International*, pages 173–176, June 2010.

[64] Wesley A. Clark. Macromodular computer systems. In *Proceedings of the April 18-20, 1967, Spring Joint Computer Conference*, AFIPS '67 (Spring), pages 335–336, New York, NY, USA, 1967. ACM.

[65] COBHAM. *GRLIB IP Core User's Manual*, 01 2016.

[66] B. W. Cook, A. D. Berny, A. Molnar, S. Lanzisera, and K. S. J. Pister. An ultra-low power 2.4ghz rf transceiver for wireless sensor networks in 0.13/spl mu/m cmos with 400mv supply and an integrated passive rx front-end. In *2006 IEEE International Solid State Circuits Conference - Digest of Technical Papers*, pages 1460–1469, Feb 2006.

[67] J. Q. Cui, S. K. Phang, K. Z. Y. Ang, F. Wang, X. Dong, Y. Ke, S. Lai, K. Li, X. Li, F. Lin, J. Lin, P. Liu, T. Pang, B. Wang, K. Wang, Z. Yang, and B. M. Chen. Drones for cooperative search and rescue in post-disaster situation. In *2015 IEEE 7th International Conference on Cybernetics and Intelligent Systems (CIS) and IEEE Conference on Robotics, Automation and Mechatronics (RAM)*, pages 167–174, July 2015.

[68] G. de Streel, J. De Vos, D. Flandre, and D. Bol. A 65nm 1v to 0.5v linear regulator with ultra low quiescent current for mixed-signal ulv socs. In *Faible Tension Faible Consommation (FTFC), 2014 IEEE*, pages 1–4, May 2014.

[69] M. Defosseux, M. Allain, P. Ivaldi, E. Defay, and S. Basrour. Highly efficient piezoelectric micro harvester for low level of acceleration fabricated with a cmos compatible process. In *2011 16th International Solid-State Sensors, Actuators and Microsystems Conference*, pages 1859–1862, June 2011.

[70] A. Dementyev, S. Hodges, S. Taylor, and J. Smith. Power consumption analysis of bluetooth low energy, zigbee and ant sensor nodes in a cyclic sleep scenario. In *Wireless Symposium (IWS), 2013 IEEE International*, pages 1–4, April 2013.

[71] G. Despesse, T.Jager, J.-J Chaillout, J.-M Leger, and S. Basrour. Design and fabrication of a new system for vibration energy harvesting. In *Research in Microelectronics and Electronics, 2005 PhD*, volume 1, pages 225–228 vol.1, July 2005.

[72] Virantha Ekanayake, Clinton Kelly, IV, and Rajit Manohar. An ultra low-power processor for sensor networks. *SIGARCH Comput. Archit. News*, 32(5) :27–36, October 2004.

[73] Elec Freaks. *Ultrasonic Ranging Module HC - SR04*, 04 2016.

[74] B. Fleming. Microcontroller units in automobiles. In *IEEE Veh. Technol. Mag.*, volume 6, pages 4–8, Sep 2011.

[75] W. J. Fleming. Overview of automotive sensors. *IEEE Sensors Journal*, 1(4) :296–308, Dec 2001.

[76] 802.11-2012 IEEE Standard for Information technology. Specific requirements part 11 : Wireless lan medium access control (mac) and physical layer (phy) specifications. In *Telecommunications and information exchange between systems Local and metropolitan area networks*, 2012.

[77] 802.15.1-2005 IEEE Standard for Information technology. Part 15.1 : Wireless medium access control (mac) and physical layer (phy) specifications for wireless personal area networks (wpans). In *Telecommunications and information exchange between systems - Local and metropolitan area networks - Specific requirements.*, 2005.

[78] 802.15.4-2011 IEEE Standard for Local and metropolitan area networks. Part 15.4 : Low-rate wireless personal area networks (lr-wpans). 2011.

[79] 802.15.6-2012 IEEE Standard for Local and metropolitan area networks. Part 15.6 : Wireless body area networks. 2012.

[80] S. B. Furber, D. A. Edwards, and J. D. Garside. Amulet3 : a 100 mips asynchronous embedded processor. In *Computer Design, 2000. Proceedings. 2000 International Conference on*, pages 329–334, 2000.

[81] T. Galchev, E. E. Aktakka, H. Kim, and K. Najafi. A piezoelectric frequency-increased power generator for scavenging low-frequency ambient vibration. In *Micro Electro Mechanical Systems (MEMS), 2010 IEEE 23rd International Conference on*, pages 1203–1206, Jan 2010.

[82] X. Guo, Y. Wang, and X. Wei. Design of wsn-based environment monitoring system on repair of gas leakage. In *The 27th Chinese Control and Decision Conference (2015 CCDC)*, pages 3340–3344, May 2015.

[83] Lenka Halámková, Jan Halámek, Vera Bocharova, Alon Szczupak, Lital Alfonta, and Evgeny Katz. Implanted biofuel cell operating in a living snail. *Journal of the American Chemical Society*, 134(11) :5040–5043, 2012. PMID : 22401501.

[84] Adnan Harb. Energy harvesting : State-of-the-art. pages 2641–2654, October 2011.

[85] M. Hempstead, D. Brooks, and G. Y. Wei. An accelerator-based wireless sensor network processor in 130 nm cmos. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 1(2) :193–202, June 2011.

[86] M. Hempstead, N. Tripathi, P. Mauro, Gu-Yeon Wei, and D. Brooks. An ultra low power system architecture for sensor network applications. In *32nd International Symposium on Computer Architecture (ISCA'05)*, pages 208–219, June 2005.

[87] D. T. Ho and S. Shimamoto. Highly reliable communication protocol for wsn-uav system employing tdma and pfs scheme. In *2011 IEEE GLOBECOM Workshops (GC Wkshps)*, pages 1320–1324, Dec 2011.

[88] Nicholas S. Hudak and Glenn G. Amatucci. Small-scale energy harvesting through thermoelectric, vibration, and radiofrequency power conversion. *Journal of Applied Physics*, 103(10), 2008.

[89] Intel. *Intel 64 and IA-32 Architectures Software Developer's Manual Combined Volumes : 1, 2A, 2B, 2C, 3A, 3B, 3C and 3D*, 04 2016.

[90] InvenSense. *MPU-9250 Product Specification*, 01 2014. Rev. 1.

[91] N. B. M. Isa, T. C. Wei, and A. H. M. Yatim. Smart grid technology : Communications, power electronics and control system. In *Sustainable Energy Engineering and Application (ICSEEA), 2015 International Conference on*, pages 10–14, Oct 2015.

[92] V. R. Jain, R. Bagree, A. Kumar, and P. Ranjan. wildcense : Gps based animal tracking system. In *Intelligent Sensors, Sensor Networks and Information Processing, 2008. ISSNIP 2008. International Conference on*, pages 617–622, Dec 2008.

[93] Ning Jin, Renzhi Ma, Yunfeng Lv, Xizhong Lou, and Qingjian Wei. A novel design of water environment monitoring system based on wsn. In *Computer Design and Applications (ICCDA), 2010 International Conference on*, volume 2, pages V2–593–V2–597, June 2010.

[94] S. E. Jo, M. K. Kim, M. S. Kim, and Y. J. Kim. Flexible thermoelectric generator for human body heat energy harvesting. *Electronics Letters*, 48(16) :1013–1015, August 2012.

[95] C. Kamble, N. Chide, S. Mhatre, and S. Sukhdeve. Thin film organic solar cell as an emerging pv technique. In *Green Computing, Communication and Conservation of Energy (ICGCE), 2013 International Conference on*, pages 649–653, Dec 2013.

[96] H. Karvonen, J. Petäjäjärvi, J. Iinatti, M. Hämäläinen, and C. Pomalaza-Ráez. A generic wake-up radio based mac protocol for energy efficient short range communication. In *2014 IEEE 25th Annual International Symposium on Personal, Indoor, and Mobile Radio Communication (PIMRC)*, pages 2173–2177, Sept 2014.

[97] E. Katz. Implantable biofuel cells operating in vivo : Providing sustainable power for bioelectronic devices : From biofuel cells to cyborgs. In *Advances in Sensors and Interfaces (IWASI), 2015 6th IEEE International Workshop on*, pages 2–13, June 2015.

[98] G. Kim, Y. Lee, Zhiyoong Foo, P. Pannuto, Ye-Sheng Kuo, B. Kempke, M. H. Ghaed, Suyoung Bang, Inhee Lee, Yejoong Kim, Seokhyeon Jeong, P. Dutta, D. Sylvester, and D. Blaauw. A millimeter-scale wireless imaging system with continuous motion

detection and energy harvesting. In *2014 Symposium on VLSI Circuits Digest of Technical Papers*, pages 1–2, June 2014.

[99] Y. Kim and H. bang. Ad-hoc network for inter-vehicle communication of multiple uavs. In *Control, Automation and Systems (ICCAS), 2014 14th International Conference on*, pages 840–843, Oct 2014.

[100] T. N. Le, A. Pegatoquet, and M. Magno. Asynchronous on demand mac protocol using wake-up radio in wireless body area network. In *Advances in Sensors and Interfaces (IWASI), 2015 6th IEEE International Workshop on*, pages 228–233, June 2015.

[101] Y. Lee, S. Bang, I. Lee, Y. Kim, G. Kim, M. H. Ghaed, P. Pannuto, P. Dutta, D. Sylvester, and D. Blaauw. A modular 1 mm3 die-stacked sensing platform with low power i2c inter-die communication and multi-modal energy harvesting. *IEEE Journal of Solid-State Circuits*, 48(1) :229–243, Jan 2013.

[102] Y. Lee, G. Kim, S. Bang, Y. Kim, I. Lee, P. Dutta, D. Sylvester, and D. Blaauw. A modular 1mm3 die-stacked sensing platform with optical communication and multi-modal energy harvesting. In *2012 IEEE International Solid-State Circuits Conference*, pages 402–404, Feb 2012.

[103] Yoonmyung Lee, Yejoong Kim, Dongmin Yoon, D. Blaauw, and D. Sylvester. Circuit and system design guidelines for ultra-low power sensor nodes. In *Design Automation Conference (DAC), 2012 49th ACM/EDAC/IEEE*, pages 1037–1042, June 2012.

[104] Z. Lei and J. Lu. Distributed coverage of forest fire border based on wsn. In *Industrial and Information Systems (IIS), 2010 2nd International Conference on*, volume 1, pages 341–344, July 2010.

[105] Zheng Liang, J. Plosila, Lu Yan, and K. Sere. On-chip debug for an asynchronous java accelerator. In *Sixth International Conference on Parallel and Distributed Computing Applications and Technologies (PDCAT'05)*, pages 312–315, Dec 2005.

[106] Libelium. *Waspmote Datasheet*, 11 2015.

[107] Y. S. Lin, D. M. Sylvester, and D. T. Blaauw. A 150pw program-and-hold timer for ultra-low-power sensor platforms. In *2009 IEEE International Solid-State Circuits Conference - Digest of Technical Papers*, pages 326–327,327a, Feb 2009.

[108] Linear Technology. *LTC3331 Nanopower Buck-Boost DC/DC with Energy Harvesting Battery Charger*, 2015.

[109] Y. H. Liu, X. Huang, M. Vidojkovic, A. Ba, P. Harpe, G. Dolmans, and H. d. Groot. A 1.9nj/b 2.4ghz multistandard (bluetooth low energy/zigbee/ieee802.15.6) transceiver for personal/body-area networks. In *2013 IEEE International Solid-State Circuits Conference Digest of Technical Papers*, pages 446–447, Feb 2013.

[110] LND inc. *712 End Window-Alpha-Beta-Gamma Detector*.

[111] G. Luhn, D. Habich, K. Bartl, J. Postel, T. Stevens, and M. Zinner. Real-time information base as key enabler for manufacturing intelligence and industrie 4.0 ;. In *2015*

26th Annual SEMI Advanced Semiconductor Manufacturing Conference (ASMC), pages 216–222, May 2015.

[112] M. Magno and L. Benini. An ultra low power high sensitivity wake-up radio receiver with addressing capability. In *2014 IEEE 10th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*, pages 92–99, Oct 2014.

[113] M. Magno, V. Jelicic, B. Srbinovski, V. Bilas, E. Popovici, and L. Benini. Design, implementation, and performance evaluation of a flexible low-latency nanowatt wake-up radio receiver. *IEEE Transactions on Industrial Informatics*, 12(2) :633–644, April 2016.

[114] M. S. Makowski and D. Maksimovic. Performance limits of switched-capacitor dc-dc converters. In *Power Electronics Specialists Conference, 1995. PESC '95 Record., 26th Annual IEEE*, volume 2, pages 1215–1221 vol.2, Jun 1995.

[115] A. J. Martin, A. Lines, R. Manohar, M. Nystrom, P. Penzes, R. Southworth, U. Cummings, and Tak Kwan Lee. The design of an asynchronous mips r3000 microprocessor. In *Advanced Research in VLSI, 1997. Proceedings., Seventeenth Conference on*, pages 164–181, Sep 1997.

[116] A.J Martin. In *Synthesis of asynchronous VLSI circuits. Technical report, California Institute of Technology*, January 1991.

[117] M. Marzencki, M. Defosseux, and S. Basrour. Mems vibration energy harvesting devices with passive resonance frequency adaptation capability. *Journal of Microelectromechanical Systems*, 18(6) :1444–1453, Dec 2009.

[118] MaxDetect Technology. *RHT03*.

[119] Micropelt Power Generator. *MPG-D655 Thin Film Thermogenerator Preliminary Datasheet*, 09 2015. Ver. 3.0.

[120] J. Myers, A. Savanth, D. Howard, R. Gaddh, P. Prabhat, and D. Flynn. An 80nw retention 11.7pj/cycle active subthreshold arm cortex-m0+ subsystem in 65nm cmos for wsn applications. In *2015 IEEE International Solid-State Circuits Conference - (ISSCC) Digest of Technical Papers*, pages 1–3, Feb 2015.

[121] Jinwoo Nam, Seong-Mun Kim, and Sung-Gi Min. Extended wireless mesh network for vanet with geographical routing protocol. In *11th International Conference on Wireless Communications, Networking and Mobile Computing (WiCOM 2015)*, pages 1–6, Sept 2015.

[122] S. V. Nandury and B. A. Begum. Smart wsn-based ubiquitous architecture for smart cities. In *Advances in Computing, Communications and Informatics (ICACCI), 2015 International Conference on*, pages 2366–2373, Aug 2015.

[123] Sarfraz Nawaz, Xiaomin Xu, David Rodenas-Herr'aiz, Paul Fidler, Kenichi Soga, and Cecilia Mascolo. Monitoring a large construction site using wireless sensor networks.

In *Proceedings of the 6th ACM Workshop on Real World Wireless Sensor Networks*, RealWSN '15, pages 27–30, New York, NY, USA, 2015. ACM.

[124] M. Nicolae, D. Popescu, and R. Dobrescu. Uav-wsn communication algorithm with increased energy autonomy. In *2015 9th International Symposium on Advanced Topics in Electrical Engineering (ATEE)*, pages 939–944, May 2015.

[125] Nordic Semiconductor. *Datasheet Single-chip ANT ultra-low power wireless network solution*, 6 2010.

[126] C. T. O. Otero, J. Tse, R. Karmazin, B. Hill, and R. Manohar. Ulsnap : An ultra-low power event-driven microcontroller for sensor network nodes. In *Fifteenth International Symposium on Quality Electronic Design*, pages 667–674, March 2014.

[127] M. Pande, N. K. Choudhari, S. Pathak, and D. Mukhopadhyay. H2e2 : A hybrid, hexagonal amp ; energy efficient wsn green platform for precision agriculture. In *Hybrid Intelligent Systems (HIS), 2012 12th International Conference on*, pages 155–160, Dec 2012.

[128] G. Papotto, F. Carrara, A. Finocchiaro, and G. Palmisano. A 90-nm cmos 5-mbps crystal-less rf-powered transceiver for wireless sensor network nodes. *IEEE Journal of Solid-State Circuits*, 49(2) :335–346, Feb 2014.

[129] N. Pinckney, D. Blaauw, and D. Sylvester. Low-power near-threshold design : Techniques to improve energy efficiency energy-efficient near-threshold design has been proposed to increase energy efficiency across a wid. *IEEE Solid-State Circuits Magazine*, 7(2) :49–57, Spring 2015.

[130] PIR Sensor Module. *SE-10*. Rev. 1.

[131] N. E. Roberts and D. D. Wentzloff. A 98nw wake-up radio for wireless body area networks. In *2012 IEEE Radio Frequency Integrated Circuits Symposium*, pages 373–376, June 2012.

[132] Y. s. Kuo, P. Pannuto, G. Kim, Z. Foo, I. Lee, B. Kempke, P. Dutta, D. Blaauw, and Y. Lee. Mbus : A 17.5 pj/bit/chip portable interconnect bus for millimeter-scale sensor systems with 8 nw standby power. In *Proceedings of the IEEE 2014 Custom Integrated Circuits Conference*, pages 1–4, Sept 2014.

[133] A. Sample and J. R. Smith. Experimental results with two wireless power transfer systems. In *2009 IEEE Radio and Wireless Symposium*, pages 16–18, Jan 2009.

[134] Semtech. *Datasheet SX1272*, 3 2015.

[135] Semtech. *LoRa Modulation Basics*, 5 2015.

[136] Mingoo Seok, Scott Hanson, Yu-Shiang Lin, Zhiyoong Foo, Daeyeon Kim, Yoonmyung Lee, Nurrachman Liu, D. Sylvester, and D. Blaauw. The phoenix processor : A 30pw platform for sensor applications. In *2008 IEEE Symposium on VLSI Circuits*, pages 188–189, June 2008.

[137] Sigma Design. *Datasheet ZM5202*, 2015.

[138] A. M. Sodagar, K. D. Wise, and K. Najafi. A wireless implantable microsystem for multichannel neural recording. *IEEE Transactions on Microwave Theory and Techniques*, 57(10) :2565–2573, Oct 2009.

[139] P. Somnath, H. Vinayak, K. Ryan, M. Turbo, A. Paolo, G. Vaughn, S. Robert, M. Debendra, J. Sandeep, V. Sriram, T. James, and D. Vivek. An energy harvesting wireless sensor node for iot systems featuring a near-threshold voltage ia-32 microcontroller in 14nm tri-gate cmos. In *(VLSI), 2016*, pages 78–79, June 2016.

[140] Spectra Symbol. *Flex Sensor Datasheet*. Rev. A.

[141] G. Stamatescu, D. Popescu, and R. Dobrescu. Cognitive radio as solution for ground-aerial surveillance through wsn and uav infrastructure. In *Electronics, Computers and Artificial Intelligence (ECAI), 2014 6th International Conference on*, pages 51–56, Oct 2014.

[142] STMicroelectronics. *VS6663 1.3 megapixel camera module*, 07 2015. Rev. 3.

[143] STMicroelectronics. *MP45DT02 MEMS audio sensor omnidirectional digital microphone*, 1 2016.

[144] STMicroelectronics. *Reference manual Ultra-low-power STM32L0x3 advanced ARM-based 32-bit MCUs*, 02 2016. Rev. 4.

[145] STMicroelectronics. *STM32L053C6 Ultra-low-power 32-bit MCU ARM-based Cortex-M0+, up to 64KB Flash, 8KB SRAM, 2KB EEPROM, LCD, USB, ADC, DAC*, 03 2016. Rev. 6.

[146] STMicroelectronics. *Ultra-low-power 32-bit MCU ARM-based Cortex-M0+, up to 192KB Flash, 20KB SRAM, 6KB EEPROM, LCD, USB, ADC, DACs, AES*, 03 2016. Rev. 3.

[147] X. Sun and X. M. Li. Study of the feasibility of vanet and its routing protocols. In *2008 4th International Conference on Wireless Communications, Networking and Mobile Computing*, pages 1–4, Oct 2008.

[148] B. Supriyo, S. S. Hidayat, A. Suharjono, M. Anif, and S. Koesuma. Design of real-time gas monitoring system based-on wireless sensor networks for merapi volcano. In *Information Technology, Computer and Electrical Engineering (ICITACEE), 2014 1st International Conference on*, pages 30–34, Nov 2014.

[149] I. E. Sutherland. Micropipelines. *Commun. ACM*, 32(6) :720–738, June 1989.

[150] Texas Instrument. *2.4 GHz IEEE 802.15.4 / ZigBee-ready RF Transceiver*, 11 2003.

[151] Texas Instrument. *Datasheet Multi standard fully integrated 13.56-MHz RFID analog front end and data -framing reader system TRF7960*, 8 2010.

[152] Texas Instrument. *MSP430F261x MSP430F241x Mixed Signal Microcontroller*, 11 2012.

[153] Texas Instrument. *Datasheet WL1805MOD*, 7 2013.

[154] Texas Instrument. *MSP430x2xx Family User's Guide*, 07 2013.

[155] Texas Instrument. *bq25570 Nano Power Boost Charger and Buck Converter for Energy Harvester Powered Applications*, 03 2015.

[156] Texas Instrument. *CC2650 SimpleLink Multistandard Wireless MCU*, 10 2015. Rev. 2.

[157] Texas Instrument. *Datasheet CC1120 High-Performance RF Transceiver for Narrowband Systems*, 7 2015.

[158] Texas Instrument. *Datasheet CC2640*, 10 2015.

[159] Texas Instrument. *Humidity and Temperature Sensor Node for Star Networks Enabling 10+ Year Coin Cell Battery Life*, 03 2016.

[160] U-Blox. *Datasheet TOBY-LA series 4G lTE modules*, 3 2016.

[161] ublox. *MAX-7 u-blox 7 GNSS modules Data Sheet*, 11 2014. Rev. 5.

[162] H. van Gageldonk, K. van Berkel, A. Peeters, D. Baumann, D. Gloor, and G. Stegmann. An asynchronous low-power 80c51 microcontroller. In *Advanced Research in Asynchronous Circuits and Systems, 1998. Proceedings. 1998 Fourth International Symposium on*, pages 96–107, Mar 1998.

[163] Versa Point Technology. *Force Sensing Resistor Integration Guide and Evaluation Parts Catalog.* Rev. D.

[164] Pascal Vivet. *Thèse : une méthodologie de conception de circuits intégrés quasi-insensible aux délais : appication à l'étude et à la réalisation d'un processeur RISC 16-bit asynchrone.* France Telecom, 2001.

[165] M. T. Vo, V. S. Tran, T. D. Nguyen, and H. T. Huynh. Wireless sensor network for multi-storey building : Design and implementation. In *Computing, Management and Telecommunications (ComManTel), 2013 International Conference on*, pages 175–180, Jan 2013.

[166] J. De Vos, D. Flandre, and D. Bol. A dual-mode dc/dc converter for ultra-low-voltage microcontrollers. In *Subthreshold Microelectronics Conference (SubVT), 2012 IEEE*, pages 1–3, Oct 2012.

[167] R.J.M. Vullers, R. van Schaijk, I. Doms, C. Van Hoof, and R. Mertens. Micropower energy harvesting. pages 684–693, July 2009.

[168] T. Y. Wang, Y. C. Lin, C. Y. Tai, C. C. Fei, M. Y. Tseng, and C. W. Lan. Recovery of silicon from kerf loss slurry waste for photovoltaic applications. *Progress in Photovoltaics : Research and Applications*, 17(3) :155–163, 2009.

[169] B. Warneke, M. Last, B. Liebowitz, and K. S. J. Pister. Smart dust : communicating with a cubic-millimeter computer. *Computer*, 34(1) :44–51, Jan 2001.

[170] B. A. Warneke and K. S. J. Pister. An ultra-low energy microcontroller for smart dust wireless sensor networks. In *Solid-State Circuits Conference, 2004. Digest of Technical Papers. ISSCC. 2004 IEEE International*, pages 316–317 Vol.1, Feb 2004.

[171] Winsen Electronics. *Toxic gas sensor Model MQ-7*, 01 2014.

[172] Joseph Yiu. *The Definitive Guide to ARM Cortex-M3 and Cortex-M4 Processors.* ARM, 2013. Third Edition.

[173] D. Yoon, D. Sylvester, and D. Blaauw. A 5.58nw 32.768khz dll-assisted xo for real-time clocks in wireless sensing applications. In *2012 IEEE International Solid-State Circuits Conference*, pages 366–368, Feb 2012.

[174] P. Yu, X. Yong, and P. Xi-yuan. Gems : A wsn-based greenhouse environment monitoring system. In *Instrumentation and Measurement Technology Conference (I2MTC), 2011 IEEE*, pages 1–6, May 2011.

[175] Zensys. *Software Design Specification, Z-Wave protocol Overview*, 4 2006.

[176] Jianhua Zhao, Aihua Wang, Martin A. Green, and Francesca Ferrazza. 19.8mono-crystalline silicon solar cells. *Applied Physics Letters*, 73(14) :1991–1993, 1998.

Annexes

A État de l'art

A.1 Réseaux et radios existantes pour les nœuds de capteurs

Aujourd'hui il existe une multitude de radios et de standards de communication pour les nœuds de capteurs. La meilleure solution doit être choisie en fonction de l'application visée et du cahier des charges. Pour cela les radios peuvent être séparées par gamme de portée comme montré dans le tableau 8.1. Quatre types de portées existent aujourd'hui, les solutions très courte portée avec les radios de type RFID/NFC (Near Field Communication) [151], les solutions courte portée avec les radios ANT+ [125], Bluetooth Low Energy [158], Z-Wave [137], Wi-Fi [153] et ZigBee [36], les solutions récentes longue portée basse consommation avec les radios LoRa [134] et les radios compatibles avec l'opérateur Sigfox [157], et enfin les solutions longue portée haute consommation avec les modems compatibles pour le réseau cellulaire 4G ou 3G [160]. Le NFC permet d'effectuer des communications à une distance de quelques centimètres à l'ordre du mètre suivant le terminal employé. Elle est donc très utilisée aujourd'hui pour les paiements sans contact, l'appairage avec des appareils électroniques, ou encore les tickets de transports. Dans les solutions courte portée, trois radios ressortent pour le domaine des nœuds de capteurs communicants en raison de leur faible consommation, ce sont les radios ANT+, Bluetooth LE et ZigBee. Le Wi-Fi lui peut atteindre de très hautes consommations (238 mA en transmission) pour des portées équivalentes au ZigBee (13,8 mA en transmission). C'est pourquoi il est très peu utilisé dans le domaine des nœuds de capteurs communicants. Le Z-Wave [175] lui reste uniquement sur le domaine de la domotique (contrôle et surveillance d'une maison ou d'un bâtiment) et possède uniquement deux fournisseurs de puce radio. Toutes ces radios courte portée fonctionnent dans le domaine des fréquences ISM (Bandes Industrielle, Scientifique et Médicale) et diffèrent par leur modulation et leur couche MAC [1]. Une comparaison de la consommation des radios Bluetooth LE, ZigBee et ANT a été faite pour des scénarios de sommeil cyclique [70]. Il en ressort que le Bluetooth LE réussit à avoir la plus basse consommation d'énergie suivit du ZigBee et de l'ANT pour les mêmes scénarios. Ce qui l'en ressort aussi c'est que la majorité de la consommation passe dans les temps de reconnections entre le coordinateur et l'esclave après un cycle de sommeil et non dans la consommation active et de sommeil de la radio de l'esclave. Le choix entre les différents standards se fera donc en fonction des portées voulues, du débit voulu, de la topologie du réseau visé et de l'environnement dans lequel il sera (interférence électromagnétique...). Mais souvent les chiffres en consommation statique (en transmission, réception et sommeil) et le débit de la radio ne suffisent pas à choisir une radio par rapport à une autre pour avoir la consommation la plus faible. Il faut tenir compte aussi du protocole utilisé par la radio et du rapport cyclique de l'application pour avoir le meilleur compromis entre consommation et performance. De nouvelles solutions voient le jour actuellement pour les nœuds de capteurs communicants grâce aux radios longue portée bas débit telles que celles proposées par Semtech [134] [135] compatibles LoRaWAN ou celle de Texas Instrument [157] compatible avec l'opérateur Sigfox [22] (l'opérateur uniquement dédié aux objets connectés). Sigfox a déployé des stations de base dans la France entière ($\sim$ 1500 antennes) et dans plein d'autres pays pour une couverture presque totale de ceux-ci ($\sim$ 91% en France). Les radios compatibles sigfox fonctionnent sur le principe de bande ultra étroite avec une modulation DBPSK (Differential Phase Shift Keying). Le message est envoyé sur une largeur de bande de 100Hz pour un canal de fréquence de largeur de 200KHz. De plus Sigfox n'autorise qu'au maximum 12 octets par message et 140 messages

1. Media Access Control

par jours par objet. Contrairement aux radios à bande ultra étroite pour Sigfox, la radio LoRa est très différente, car elle fonctionne sur le principe de la modulation par étalement de spectre [135]. Elle utilise 8 canaux de fréquence de 125 KHz de large. De plus LoRa permet de faire son propre réseau local sans passer par une station de base qui appartient à un opérateur (Bouygues et Orange pour l'instant). Il n'y a pas non plus de limitation de longueur de message ni de nombre de message par objet et par jour mais doit quand même respecter le taux d'occupation de la bande 868 MHz de 1% à 10%. Ainsi ce sont deux solutions très différentes et qui présentes chacune des avantages et des inconvénients. Il pourrait même être envisagé d'utiliser les deux radios pour des parties différentes d'une application. Ce qui est très intéressant dans ces nouvelles solutions ce sont les très grandes portées possibles que ce soit dans un milieu urbain (5 à 10 km) ou rural (15 km à 50 km) pour des consommations jusqu'à 1000 fois plus faible qu'un modem 4G LTE [160].

Nous pouvons voir que l'écosystème de radios pour nœuds de capteurs communicants est énorme et très concurrentiel. Plein de solutions radio fréquence existent, il faut donc bien déterminer les besoins de l'application en termes de portée, consommation et autonomie pour sélectionner la bonne radio et protocole à utiliser.

A.2 Applications pour nœuds de capteurs communicants

A.2.1 Domotique

L'un des premiers secteurs touché par les nœuds de capteurs est la domotique. Le but est de centraliser le contrôle des différents systèmes électriques, électroniques, mécaniques d'une maison ou d'un bâtiment pour améliorer le confort des personnes, la sécurité du bâtiment et la communication homme machine. Un exemple de déploiement de nœuds de capteurs dans un bâtiment est donné dans [165]. Les auteurs ont installé sur plusieurs étages d'un bâtiment des nœuds de capteurs et actionneurs suivants :

— capteur de température et d'humidité,
— capteur de détection infrarouge passif,
— capteur de statut de porte et fenêtre,
— nœud de commande,
— nœud de contrôle de la lumière,
— nœud de surveillance de la puissance, consommée par une prise électrique

Ce réseau de capteurs est basé sur les radios ZigBee où les nœuds de capteurs se réveillent toutes les 120 secondes et communiquent avec le nœud coordinateur (figure A.1) pendant 1 seconde. Avec cette technique ils estiment une durée de vie du nœud de 1,7 ans. Cela reste assez limité et peut être vraiment amélioré. Dans le domaine du commerce il existe une multitude de système pour faire de la domotique tels que les produits compatibles Z-Wave [25] pouvant être directement connectés et utilisés dans un réseau Z-Wave et ayant des durées de vie de 2 ans sur pile. Ainsi, la réduction de la consommation du microcontrôleur a toute son importance pour pouvoir augmenter la durée de vie globale du nœud pour la domotique.

A.2.2 Transports

Le secteur du transport va complètement changer dans les vingts prochaines années. Il va être bouleversé par les voitures autonomes, le transport par drone, et des transports en commun plus intelligents avec un service largement amélioré.

- Automobile

Radios	Solution très courte portée	Solution courte portée					Solution longue portée LP		Solution longue portée HP
	NFC	**ANT+**	**Bluetooth LE**	**Z-Wave**	**Wi-Fi**	**ZigBee**	**LoRa**	**Compatible SIGFOX**	**4G**
Standard	NFCIP-1	ANT	802.15.1	Z-Wave alliance	802.11.n	802.15.4	LoRaWAN	Bande étroite	LTE Cat 3
Fréquence	13,56MHz	2,4GHz	2,4GHz	868MHz (EU)	2,4GHz / 5GHz	868MHz (EU) /2,4GHz	868MHz (EU)	868MHz (EU)	700 - 2600MHz
Portée intérieur(m)	0,2	10	10	45	50	30	2000 - 5000 (ville)	3000 - 10000 (ville)	> 100 (ville)
Portée extérieur(m)	0,2	30	50	150	125	1500	15000 (rural)	30000 - 50000 (rural)	30000 (rural)
Courant max	10 mA	17 mA	5,9 (RX) - 9,1 mA (TX)	41 mA	65 (RX) - 238 mA (TX)	11,8 (RX) - 13,8 mA (TX)	11,2 (RX) - 18->125 mA (TX)	17 (RX) - 45 mA (TX)	< 800 mA
Courant veille	$120\mu A$	$2\mu A$	$1\mu A$	$1\mu A$	$160\mu A$	$0,2\mu A$	$0,1\mu A$	$0,12\mu A$	-
Débit max	848 kb/s	20 kb/s	1 Mb/s	100 kb/s	100 Mb/s	2 Mb/s	300 kb/s	50 kb/s	100 Mb/s
Débit typique	26,48 kb/s	20 kb/s	-	40 kb/s	80 Mb/s	250 kb/s	-	-	-
Topologie	point à point	point à point étoile arbre maillé diffusion	étoile multi-étoile	étoile arbre maillé	point à point étoile	étoile arbre maillé	étoile multi-étoile	comme réseau cellulaire	réseau cellulaire
Applications	Ticket de transport Paiment sécurisé Traçabilité de produit	Dispositifs santé Suivi de sport	Santé, Médicale Industrie Domotique Sport	Domotique	PC WLAN Multime-dia	Domotique Industrie Environnement	Domotique Industrie Sécurité Longue portée	Alarme Sécurité Domotique Industrie	Téléphone Mobile Clé 4G

TABLE 8.1 – Tableau comparatif des radios pour l'internet des objets

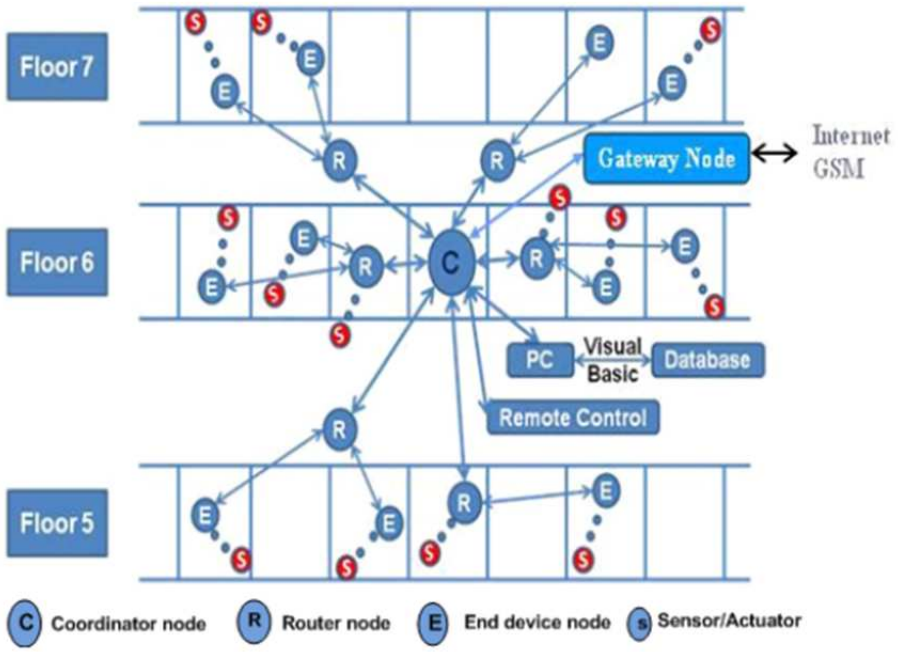


FIGURE A.1 – Implémentation d'un système automatique dans un batiment à plusieurs étage [165]

Une voiture peut embarquer jusqu'à 50 microcontrôleurs aujourd'hui [74] avec près de 100 capteurs [75]. Ce nombre va fortement augmenter dans les années à venir pour rendre la voiture entièrement autonome avec de multiples capteurs de détection, des radars et caméras. De plus, une énorme recherche est en cours pour mettre en place une communication véhicule à véhicule et véhicule à infrastructure grâce au réseau VANET [121] [147] (*vehicular ad-hoc network*). Ce réseau permettra de faire communiquer les véhicules les uns avec les autres et avec les routes pour s'échanger des informations (figure A.2) sur leur position, les dangers détectés, etc. Cela permettra une meilleure gestion du trafic routier, et pourra, à terme, réduire le nombre d'accidents. Ces systèmes fonctionneront dans la bande 5,9 GHz sur le standard IEEE 802.11p [26] pour le standard de communication de véhicule à véhicule/infrastructure (WAVE [2]). L'une des approches prometteuse est de créer des réseaux sans fil en maille (WMNs [3]) pour avoir un réseau sans fil à très haute bande passante grâce au standard IEEE 802.11s [27].

- Drones

Il existe quatres domaines d'application des drones : le domaine du transport de personnes [10], du loisir [9], du transport de marchandise [1] et le domaine pour usage professionnel (agriculture, mines et carrières, analyse de terrains, etc.)[6]. Les drones peuvent être vus comme des nœuds de capteurs communicants car ils ont au moins un microcontrôleur, une radio et une multitude de capteurs tels que des accéléromètres, gyroscopes, compas, baromètre, gps, caméras, ultrasons pour pouvoir s'asservir et se diriger en autonomie dans un environnement très compliqué (vent, obstacles, etc.). Dans un futur proche, il pourrait y avoir des milliers de drones volant au dessus d'une ville. Il faudra alors que tous ces drones communiquent les uns avec les autres [99] [54] et avec le sol pour éviter les collisions. Il y aura alors des réseaux de capteurs fixes au sol qui pourront communiquer avec des résaux de drones mobiles en vol [87] [141]. Des réseaux de nœuds de capteurs communicants

2. Wireless Access in Vehicular Environment
3. Wireless Mesh Networks

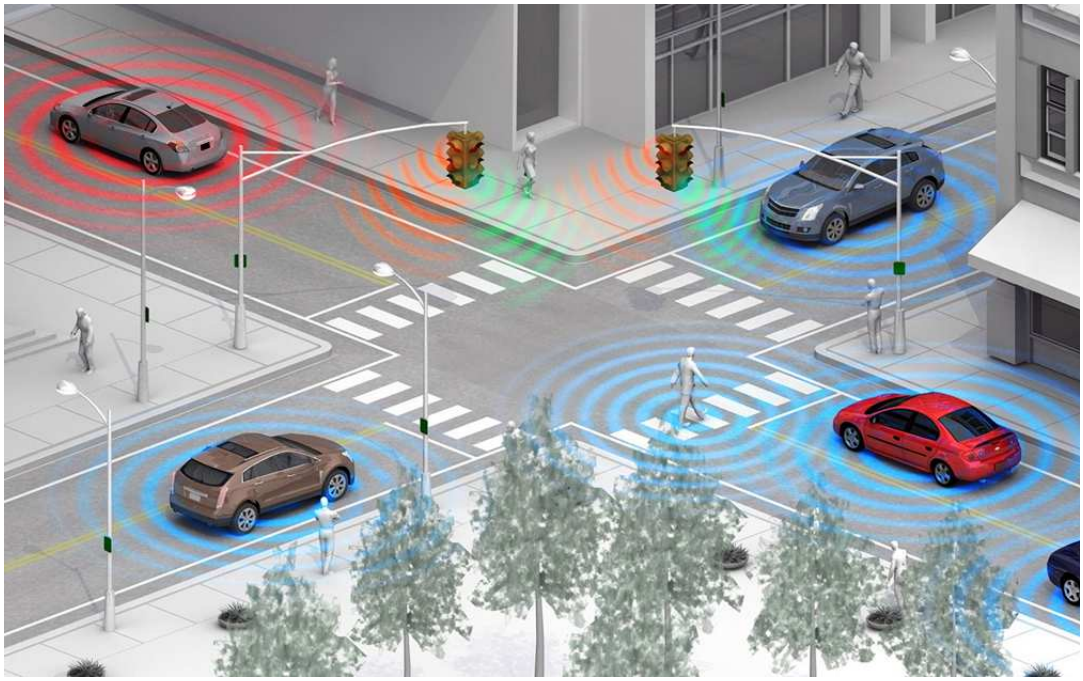


FIGURE A.2 – Exemple de véhicules interconnectés entre eux et avec les infrastructures [7]

pourront aussi effectuer des mesures et les envoyer aux drones [124] [49].

Dans le cadre des véhicules et des drones, les capteurs et microcontrôleurs sont branchés directement sur une batterie à forte capacité. Ainsi la consommation du nœud importe moins par rapport à l'énergie dépensée par les moteurs des véhicules. Néanmoins, dans le cadre des nœuds de capteurs déployés dans l'environnement ou dans une ville pour communiquer avec les drones ou véhicules, ceux-ci devront avoir une très grande autonomie pour pouvoir communiquer le plus longtemps possible. De plus des récupérateurs d'énergie pourront être ajoutés pour rendre ces nœuds presque illimités en énergie. Enfin, l'un des points important dans ce domaine sera la fiabilité des communications et la sécurité des liaisons radios entre les différents moyens de transport afin d'éviter tout incident et piratage.

A.2.3 Santé et dispositifs portatifs sur le corps

Les nœuds de capteurs sont aujourd'hui très utilisés dans le domaine du biomédical et des accessoires connectés sur le corps. Plusieurs exemples d'applications vont être donnés ici.

- Biomédical

Dans le domaine du biomédical, l'objectif est de recueillir des signaux analogiques qui correspondent aux activités physiologiques d'une personne ou des actions de son corps. Le but est de pouvoir faire un diagnostic en temps réel d'une personne. Dans certains cas il faut que ces dispositifs soient capables de communiquer avec les services distants, d'appeler les urgences dans le cas d'un danger immédiat ou encore d'administrer sur commande des médicaments. Ces nœuds de capteurs peuvent être disposés autour, sur ou dans le corps. Ils doivent donc être d'une grande autonomie et avoir une très faible consommation pour éviter de remplacer les dispositifs trop souvent. De plus, la plupart de ces systèmes électroniques doivent être de petite taille pour que le patient ne se rende pas compte de

sa présence. Ces réseaux de capteurs de corps sont appelés des BAN [4] ou WBAN [5] et sont définis par le standard IEEE 802.15.6 [79]. Un exemple est donné figure A.3 . Les capteurs biomédicaux peuvent être par exemple : des accéléromètres/gyroscopes pour surveiller la posture du corps, des capteurs de mesure de la glycémie pour mesurer la concentration de glucose dans le sang, des capteurs de pression artérielle pour la pression sanguine, des capteurs SpO2 pour la concentration d'oxygène dans le sang, un électrocardiogramme (ECG) pour enregistrer l'activité électrique du cœur, un électroencéphalogramme (EEG) pour enregistrer l'activité électrique du cerveau, un électromyogramme (EMG) pour mesurer les signaux électriques produits par les muscles et des capteurs d'humidité/température pour mesurer la température du corps et l'humidité ambiante. Certains actionneurs peuvent être installés comme les pompes à insuline pour injecter de l'insuline automatiquement en fonction de mesures en temps réel.

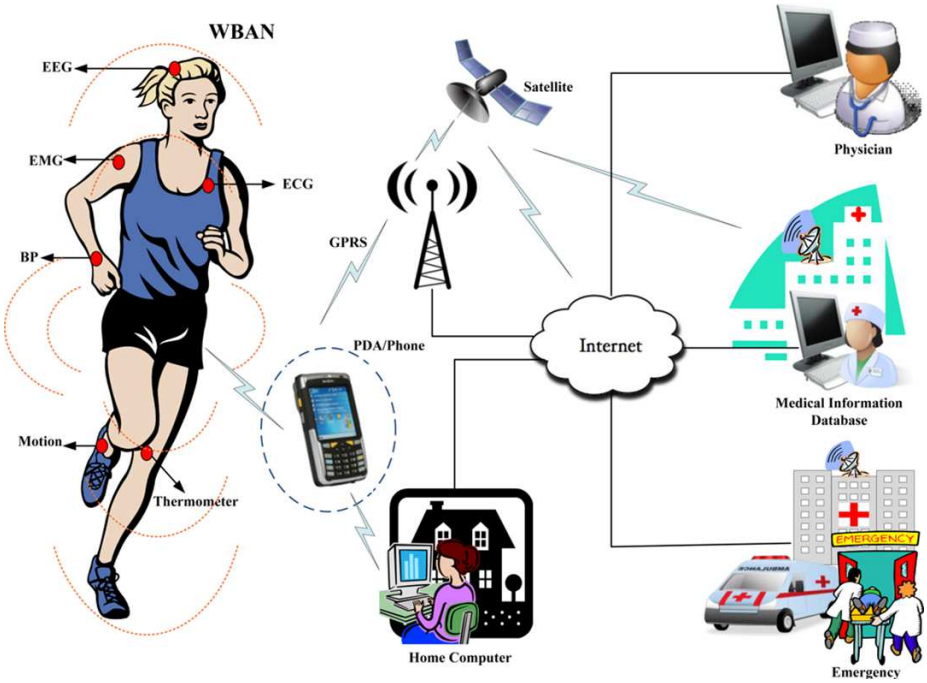


FIGURE A.3 – Exemple de BAN

On voit bien que dans cette application la consommation de chacune des entités du nœud de capteurs a une importance fondamentale pour que ces dispositifs aient l'autonomie la plus longue possible afin d'être le moins contraignant pour le patient.

- Vêtements et accessoires connectés

Ce domaine est en pleine expansion que ce soit pour les particuliers avec les montres [23] [21] et lunettes connectées [20] [11], ou bien encore toute sorte de vêtements connectés [4] [18] pour le sport ou le quotidien afin de surveiller son activité et avoir des conseils pour améliorer la santé et le bien être des personnes (figure A.4). Dans le domaine militaire ou de la sécurité, de nombreuses recherches sont faites pour rendre les soldats ou pompiers ultra connectés afin d'améliorer la communication entre eux sur le terrain, les assister au combat ou dans les feux et surveiller leur état. Ici encore, la réduction de la consommation de ces vêtements ou accessoires connectés est primordiale afin d'éviter de les recharger trop souvent.

4. Body Area Network
5. Wireless Body Area Network

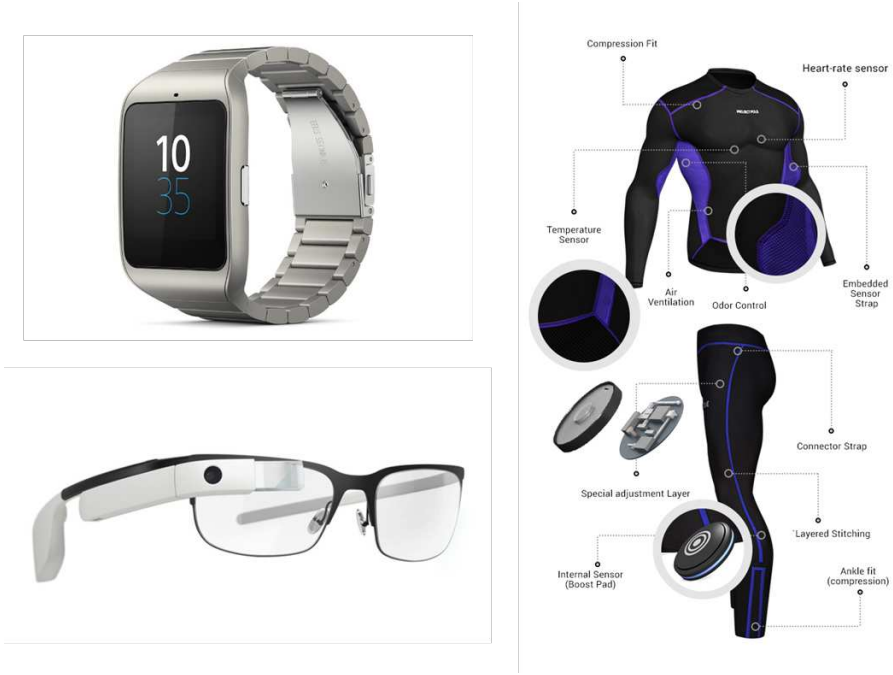


FIGURE A.4 – Exemple de vêtements et d'accessoire connectés, google glass, montre sony, vêtements

A.2.4 Constructions, infrastructures, villes et environnement

Le déploiement de nœuds de capteurs dans les villes va permettre une meilleure gestion de celles-ci par l'envoi de données en temps réel aux autorités concernées ou aux citoyens directement. Plusieurs applications sont envisagées dans les villes pour les rendre plus intelligentes, plus facile à gérer pour les collectivités et améliorer la qualité de vie des citoyens [122]. Ces applications peuvent aller de la surveillance du niveau de pollution dans chaque rue de la ville, la détection de fuite d'eau, la détection de pollution sonore, une meilleure gestion des déchets et du tri, une meilleure gestion des places de parking disponibles les plus proches, ainsi qu'une gestion dynamique de l'éclairage de la ville et des transports en commun beaucoup plus au service des citoyens. Dans le domaine de la construction de bâtiments, les réseaux de nœuds de capteurs communicants peuvent être utilisés pour surveiller les contraintes en temps réel sur les infrastructures [123]. La surveillance de paramètres environnementaux tels que la qualité de l'eau [93] ou les fuites de gaz [82] sont des applications dédiées aux réseaux de capteurs communicants, et sont aujourd'hui beaucoup utilisées pour toute la maintenance des pipelines et des réseaux d'eau potable. Cela permet de réduire considérablement les coûts de maintenance. Lorsqu'un problème survient, les autorités concernées sont renseignés immédiatement de l'endroit et peuvent agir le plus vite possible.

A.2.5 Gestion de l'énergie ou *Smart Grid*

Le *Smart Grid* désigne un réseau électrique dit *intelligent* afin d'optimiser la production, la distribution et la consommation. La production d'électricité locale se démocratise de plus en plus avec les panneaux solaires installés sur les toits des bâtiments, les éoliennes installées dans les champs et jardins de particuliers afin d'avoir des bâtiments dits à énergie positive. Le but du réseau intelligent est de pouvoir gérer les consommations va-

riables en adaptant la consommation par rapport à la production instantanée grâce à des compteurs communicants installés chez les particuliers [5], d'optimiser la production en fonction des sites de production irrégulier (solaire, éolien) et des sites de production régulier (hydrolique, nucléaire), et enfin d'optimiser la distribution de l'électricité en consommant l'électricité au plus près de là où elle est produite (production décentralisée) contrairement à la production centralisée actuelle [91]. Il faudra donc des millions de nœuds de capteurs communicants capables de faire des mesures au niveaux de chaque consommateur, au niveau des réseaux de distributions et au niveau des réseaux de production afin que le réseau optimise en temps réel l'offre et la demande. Les contraintes en termes de consommation sur ces nœuds de capteurs sont faibles étant donné qu'ils sont reliés au réseau électrique. La consommation doit cependant rester faible dans l'optique que ce réseau intelligent doit lui aussi être peu consommateur d'énergie.

A.2.6 Sécurité, Gestion des désastres et catastrophes naturelles

La sécurité et la gestion des catastrophes naturelles sont aussi des applications phare des réseaux de capteurs communicants. Cela peut par exemple aller de l'étude des volcans pour anticiper les éruptions volcaniques afin d'évacuer les populations alentours [148], à la détection et la gestion des feux de forêt [104], ou encore à la gestion juste après des désastres grâce à l'utilisation de drones [67] et de tous les objets connectés présents pour détecter des personnes en danger et les secourir.

A.2.7 Industrie 4.0

Cette quatrième révolution industrielle va s'appuyer sur différentes briques technologiques telles que le *Big Data*, la robotique, la cobotique, les outils de simulation, les capteurs communicants, la cyber-sécurité, la fabrication additive et la réalité augmentée. Cette industrie se basera sur la communication de tous les équipements de la chaine de production et d'approvisionnement [111]. Les capteurs communicants permettront d'avoir un diagnostic en temps réel de toute la chaine et de la contrôler à distance. Le but sera d'avoir une flexibilité et une personnalisation de la production ainsi qu'une usine économe en énergie et en matière première grâce à l'interconnexion de tous ces systèmes.

A.2.8 Agriculture

L'agriculture va pouvoir bénéficier des réseaux de capteurs communicants afin d'optimiser le rendement des cultures [127], d'économiser l'eau, de connaitre les paramètres environnementaux pour agir sur la croissance des plantes [174] en mesurant la température et l'humidité du sol ou de l'air, l'intensité de la lumière ou encore la concentration en CO2. Les nœuds de capteurs peuvent aussi être utilisés pour traquer les animaux sauvages [92] ou bien un élevage afin de comprendre leur comportement dans la nature ou les retrouver s'ils sont perdus. Dans ces applications, les contraintes en énergie sont fortes car il faut éviter que l'agriculteur vienne changer les piles des capteurs régulièrement, c'est pourquoi ces nœuds devront avoir une très faible consommation d'énergie ainsi que des récupérateurs d'énergie pour allonger considérablement la durée de vie du réseau.

A.3 Capteurs

Dans le domaine acoustique, des microphones [143] peuvent être utilisés pour détecter et enregistrer de la voix de même que des capteurs à ultrason [73] pour effectuer des mesures de distances. Pour des mesures chimiques, il existe par exemple des capteurs de

Type	**Capteurs**	**Références**	**Interfaces Microcontrôleur**	**Courant mesure**	**Courant veille**
Acoustique	Microphone	MP45DT02 [143]	I2S / I2C / ADC	0,65 mA	$20\mu A$
	Ultrason	HC-SR04 [73]	1 entrée d'interruption Timer	15 mA	-
Chimique	CO2, CO, CH4	MQ-7 [171]	ADC	$958\mu A$	$530\mu A$
Courant électrique	Effet Hall	ACS712 [29]	ADC	10 mA	-
Environnement	humidité du sol	Itead Studio Soil Moisture [14]	ADC	5 mA	-
	Humidité / Température Air	RHT03 [118]	1 entrée / sortie	1,5 mA	$50\mu A$
	Pression	BMP280 [52]	I2C	$600\mu A$	$0,1\mu A$
Physiologique	Fréquence cardiaque	PULSE SENSOR [19]	ADC	4 mA	-
	Activité musculaire	MUSCLE SENSOR [28]	ADC	-	-
Radiation	Tube Geiger	712 End Window Alpha Beta [110]	1 entrée d'interruption Timer	20 mA	-
Déplacement	Accéléromètre / Gyroscope / Compas	MPU9250 [90]	I2C / SPI	$8,4\mu A$ - 3,7 mA	$8\mu A$
	PIR detecteur	SE-10 [130]	1 entrée d'interruption	1,6 mA	-
Position	GPS	Ublox MAX 7C [161]	UART	4,5 mA - 16,5 mA	$300\mu A$
Optique	Photorésistance	GL5528 [55]	ADC	$33\mu A - 500\mu A$	-
	Imageur	VS6663 [142]	MIPI CSI	58 mA	$4\mu A$
Mécanique	Force	Force Sensor[163]	ADC	$4,95\mu A - 450\mu A$	-
	Flexion	Flex Sensor [140]	ADC	$160\mu A - 250\mu A$	-
	Position, Fin de course	Interrupteur	Entrée d'interruption	-	-
	Tactile	AT42QT1010 [34]	Entrée d'interruption	$34\mu A - 378\mu A$	-

TABLE 8.2 – Exemples de capteurs par domaine d'application ainsi que leurs consommations en mesure et en veille

dioxyde de carbone (CO2), de monoxyde de carbone (CO) [171], de méthane (CH4) ou d'autre types de capteurs de gaz. Pour la mesure de courants, des capteurs à effet hall sont utilisés [29] afin de connaître la consommation d'un appareil électrique ou d'une infrastructure. Dans le domaine environnemental, des capteurs d'humidité pour le sol [14] sont utilisés pour arroser automatiquement des plantes. Des capteurs d'humidité/température de l'air [118], ou encore des capteurs de pression [52] sont utilisés pour avoir les tendances météorologiques ainsi que pour en déduire l'altitude du capteur. Dans le cadre de la mesure de radiation des tubes geiger [110] sont employés pour avoir les taux de radiation alpha beta gamma dans l'environnement. Dans le domaine des mesures physiologiques, l'activité musculaire [28] et cardiaque [19] peuvent être utilisées pour surveiller l'activité du corps dans le cadre des BAN. Les déplacements, vitesses et angles peuvent être mesurés grâce à des centrales inertielles (l'ensemble des accéléromètres, gyroscopes et compas) [90]. Des personnes peuvent être détectées dans une pièce grâce à des capteurs infrarouges passifs (PIR detector) [130]. La position géographique d'une personne ou d'un objet peut être acquise grâce à l'utilisation d'un GPS [161]. Dans le domaine de l'optique, des capteurs de luminosité [55] peuvent être employés pour automatiser certaines tâches d'une application. Dans le domaine de l'optique, d'énormes recherches ont été faites sur les imageurs [142]

pour les applications de surveillance de même que pour les appareils photo de téléphones portables. Enfin, dans le domaine de la mécanique beaucoup de capteurs existent pour mesurer la force exercée sur une surface [163], l'angle de flexion d'une surface [140], la fin de course de structures mécaniques, ou encore des capteurs pour rendre tactile des surfaces [34]. Certains de ces capteurs sont illustrés figure A.5 .

FIGURE A.5 – Exemples de capteurs utilisés dans les nœuds de capteurs sans fil. (De gauche à droite) capteur de flexion [140], capteur de pression [52], capteur d'humidité [118], capteur de pulsation [19], centrale inertielle [90], capteur d'humidité du sol [14], imageur [142], capteur de monoxyde de carbone (CO) [171]

A.4 Récupérateurs d'énergie

A.4.1 Énergie vibratoire, mécanique, mouvement

La récupération d'énergie vibratoire et mécanique peuvent être réalisées grâce à des récupérateurs piézoélectriques, électromagnétiques ou encore électrostatiques.

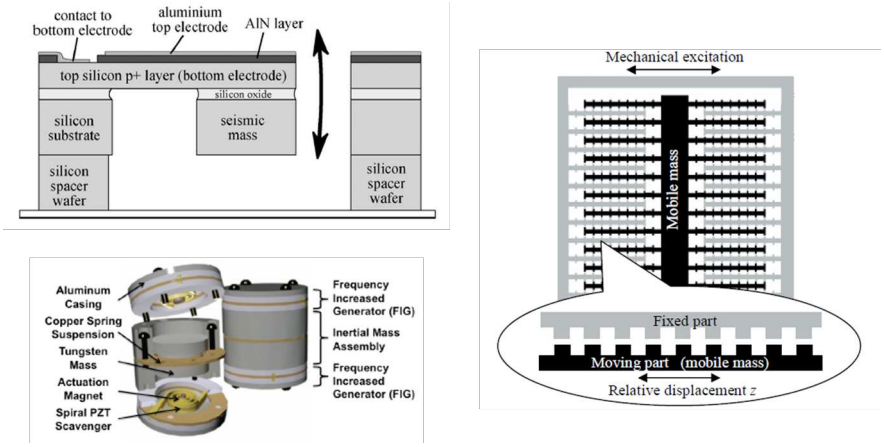


FIGURE A.6 – Récupérateur piézoélectrique [117], Récupérateur électromagnétique [81] et récupérateur électrostatique [71]

Les récupérateurs piézoélectriques fonctionnent grâce à la capacité de certains matériaux à se polariser électriquement lorsque ceux-ci sont soumis à des déformations. L'effet piézoélectrique inverse existe aussi en appliquant une différence de potentiel à ces matériaux, ils ont la capacité de se déformer. L'effet piézoélectrique est utilisé pour les récupérateurs d'énergie afin de récupérer de la puissance électrique générée à partir d'un matériau soumis à des vibrations. Beaucoup de travaux ont été fait sur les récupérateurs piézoélectriques comme [117] [69]. En 2011 [69] arrive à récupérer $0{,}62\mu W$ à 214Hz pour des accélérations de 0,17g.

Les récupérateurs électromagnétiques utilisent le principe de l'induction électromagnétique avec le déplacement relatif de bobines par rapport à des aimants. Dans [81] leur récupérateur électromagnétique arrive à récupérer $3{,}25\mu W$ pour une fréquence de 10Hz.

Les récupérateurs électrostatiques fonctionnent grâce à une capacité dont une des électrodes est mobile. Dans [71] le récupérateur électrostatique récupère $250\mu W$ pour une fréquence de 50Hz.

A.4.2 Énergie thermique

La récupération d'énergie thermique est basée sur l'effet Seebeck. C'est lorsqu'une différence de potentiel apparait entre deux jonctions de matériaux soumis à des températures différentes. L'équipe de Böttner présente un générateur thermoélectrique de $1{,}12mm^2$ contenant 12 jonctions seulement et générant $0{,}67\mu W$ [53]. Une des applications de ce genre de récupérateur d'énergie se retrouve dans les nœuds de capteurs sur corps dans le cadre des BAN [6]. Des générateurs thermoélectriques ont été créé pour le corps humain [94]. Celui-ci arrive à produire de $0{,}1\mu W$ à $2{,}1\mu W$ pour des températures de 5°C à 20°C de différence entre la température du corps et la température ambiante.

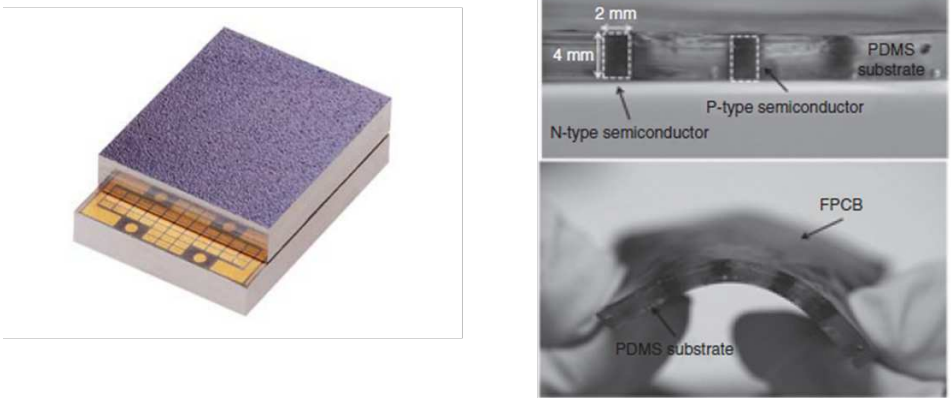


FIGURE A.7 – Générateur Thermoélectrique Micropelt [119] et pour corps humain [94]

A.4.3 Énergie électromagnétique

Aujourd'hui les ondes électromagnétiques provenant des hyperfréquences générées par les installations humaines sont omniprésentes. Il existe des récupérateurs d'énergie provenant de ces hyperfréquences comme les antennes pour les puces RFID pour des alimentations courtes portées ou les antennes pour des sources distantes mais très puissantes comme les antennes de télévision. Dans [133], Sample a créé un nœud de capteurs capable

6. Body Area Network

de récupérer jusqu'à $60\mu W$ à une distance de l'antenne de transmission des chaines de télévision de 4km. L'équipe de Papotto a créé un émetteur-récepteur capable d'être alimenté et de recevoir des données par la radio 915MHz et d'émettre sur la radio 2,4GHz [128]. La radio de réception est capable de récupérer des puissances de l'ordre du μW.

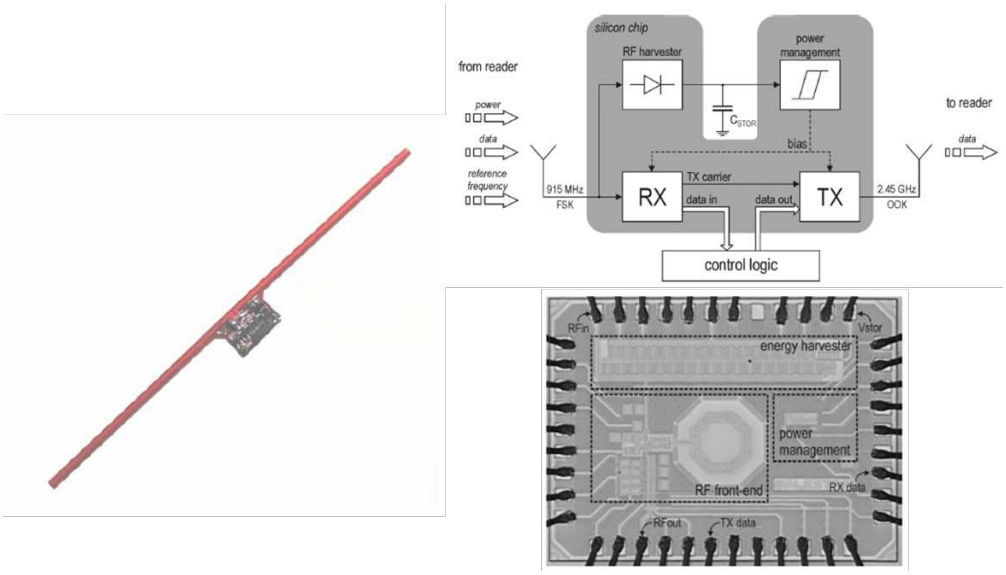


FIGURE A.8 – Récupérateur d'énergie par Radio Fréquence, WISPs [133] et RF récepteur-émetteur avec récupération d'énergie [128]

A.4.4 Énergie solaire

L'énergie solaire est l'une des plus grandes sources d'énergie disponibles dans le système solaire. Celle-ci est récupérée à l'aide de cellules photovoltaïques qui convertie l'énergie lumineuse en énergie électrique. Ces cellules photovoltaïques sont constituées d'une jonction PN. La couche supérieure de cristaux de silicium exposée au rayonnement est dopée avec du phosphore contenant plus d'électrons (dopage de type N) et la couche inférieure est dopée avec du bore contenant moins d'électrons (dopage de type P) comme montré figure A.9 . Lorsque les photons traversent la cellule, des électrons sont arrachés aux atomes de silicium et des paires électron-trou sont libérées au niveau de la jonction créant ainsi un courant continu dans la charge proportionnel à l'éclairement (figure A.9).

Chaque cellule photovoltaïque délivre une tension de 0,5V à 0,6V, c'est pourquoi un panneau solaire est constitué d'une multitude de cellules solaires pour atteindre la tension voulue. Il existe plusieurs types de cellules photovoltaïques (mono-cristallin, poly-cristallin, organique...) [176][16], [168], [95][12] ayant des rendements pouvant aller de 12% à 25%. Le rendement des cellules varie en fonction de la technologie utilisée, de l'irradiance mais également de la longueur d'onde de la source lumineuse.

A.4.5 Énergie biologique

L'énergie provenant de réaction chimique in vivo peut être récupérée avec l'utilisation de biopile. Dans [97], l'auteur fait une présentation des différentes avancées en terme de biopiles implantables in vivo. Il présente entre autre le papier de Halámková [83] dans lequel ils ont implanté une biopile dans un escargot capable de générer de l'électricité

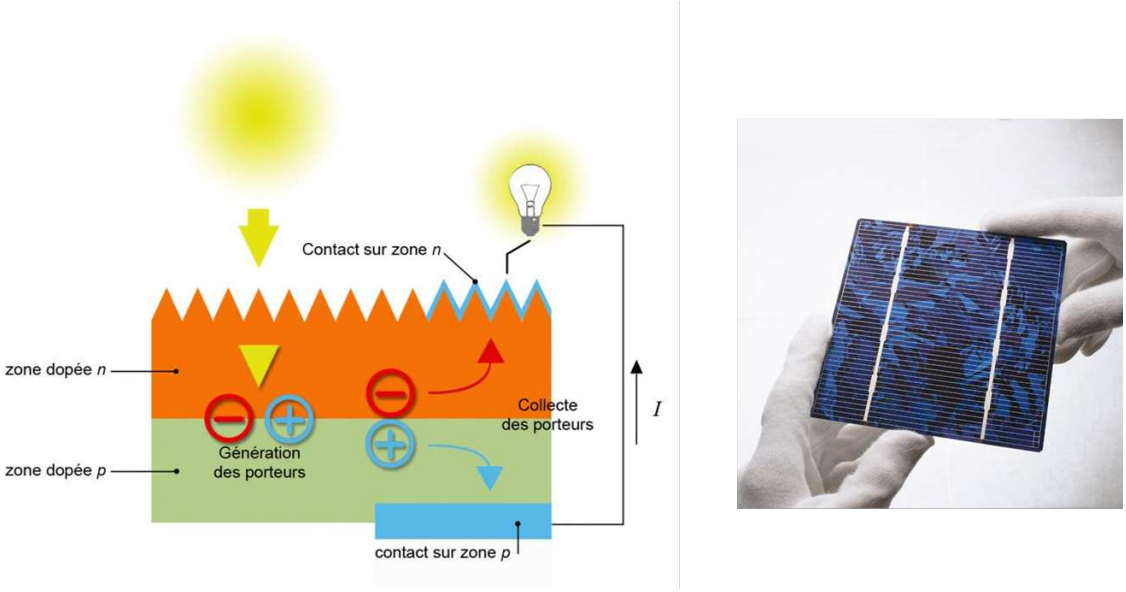


FIGURE A.9 – Principe de fonctionnement d'une cellule photovoltaïque [17] et cellule solaire polycristalline

grâce à la production interne de glucose. Ce récupérateur permet de récupérer $7,45\mu W$ pour une résistance de charge optimum de $20k\Omega$.

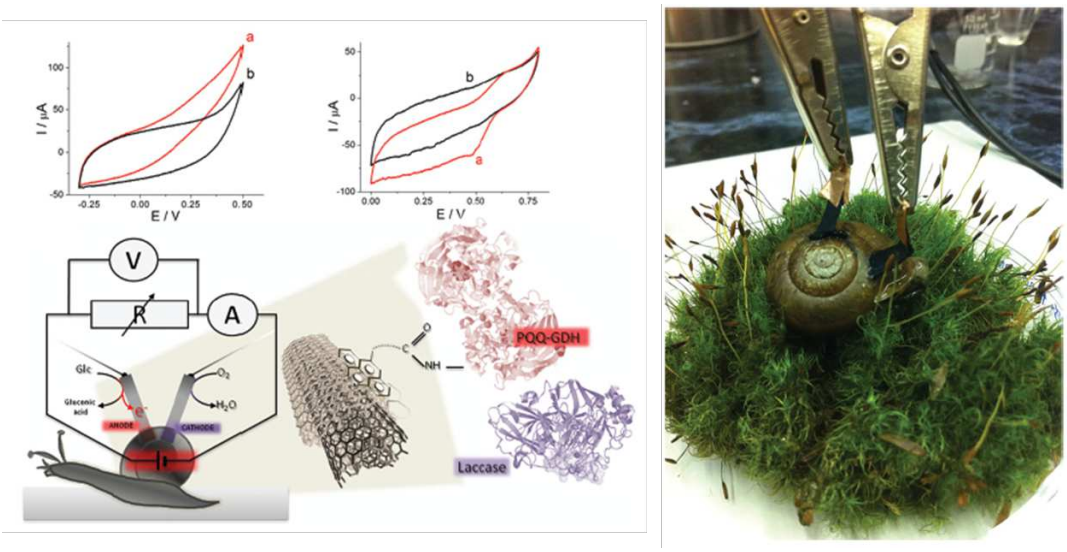


FIGURE A.10 – Bio-générateur implanté dans un escargot [83]

A.5 Gestion de l'énergie

Il existe deux principaux types de convertisseurs dans le domaine des nœuds de capteurs communicants :

- Les convertisseurs linéaires
- Les convertisseurs à capacités commutées

Les convertisseurs linéaires sont des convertisseurs abaisseurs de tension. La régulation de la tension de sortie se fait par une boucle de rétroaction qui vient piloter le transistor

Source d'énergie	Dimension	Densité de puissance	Excitation	Rendement	Ref
Vibratoire, mécanique					
piézoélectrique	$2.8mm^3$	$225\mu W.cm^{-3}$	2.7g, 214Hz	NC*	[69]
électromagnétique	$1.2mm^3$	$2.7\mu W.cm^{-3}$	1g, 10Hz	NC	[81]
électrostatique	$18cm^3$	$333\mu W.cm^{-3}$	104g, 50Hz	60%	[71]
Thermique					
générateurs thermoélectrique	$1.12mm^2$	$60\mu W.cm^{-2}$	$\Delta T = 5°C$	NC	[53]
générateurs thermoélectrique BAN	$50mm^2$	$0.2 - 4,2\mu W.cm^{-2}$	$\Delta T = 7 - 19°C$	NC	[94]
Micropelt MPG-D655	$8mm^2$	$2.5 - 72.5mW.cm^{-2}$	$\Delta T = 5 - 30°C$	NC	[119]
Électromagnétique					
RF-powered from VHF-UHF King-TV tower	-	$60\mu W$	960KW, 4.1Km, 674 - 680 MHz	NC	[133]
Solaire					
mono-cristallin	NC	$244W/m^2$	$1000W/m^2$	24.4%	[176]
poly-cristallin	$60cm^2$	$203W/m^2$	$1000W/m^2$	20.3%	[168]
organique	NC	$132W/m^2$	$1000W/m^2$	13.2%	[12]
Biologique					
Biopile	$0.25cm^2$	$30\mu W.cm^{-2}$	$20k\Omega$ load resistance	NC	[83]

*NC=Non Communiqué

TABLE 8.3 – Différents types de récupérateurs d'énergie et leurs performances

par lequel passe le courant d'entrée pour aller vers la sortie. Pour de faibles différences de tension entre l'entrée et la sortie on parle de LDO [7]. L'équation A.1 montre le rendement maximum qui peut être obtenu avec un régulateur linéaire.

$$\eta_{Lin} = \frac{P_{Out}}{P_{In}} = \frac{P_{In} - P_{Trans} - P_{Ctrl} - P_{Leak}}{P_{In}} \approx \frac{\overbrace{V_{In} - V_{Trans}}^{V_{Out}}}{V_{In}} - \frac{P_{Ctrl} + P_{Leak}}{P_{In}} < \frac{V_{Out}}{V_{In}} \tag{A.1}$$

Celui-ci est toujours inférieur à $Vout/Vin$ et la différence de tension entre l'entrée et la sortie est rejeté sous forme de chaleur dans le transistor de contrôle. Dans la littérature [68] mettent en œuvre un régulateur linéaire de 1V en entrée à 0,5V en sortie avec un très faible courant de repos (courant de contrôle et de fuite) de l'ordre de 280nA pour un courant maximum en sortie de 0,5mA.

Les convertisseurs à capacités commutées ou réseau de capacités commutées (SCN [8]) effectuent un transfert d'énergie d'une capacité à l'autre grâce à un transistor de contrôle entre chaque capacité. Ces convertisseurs sont basés sur l'architecture d'une pompe de charge de Dickson. Les transistors sont pilotés par des horloges différentes Φ_1 et Φ_2 qui ne se recouvre pas. Le rendement de tel convertisseur est défini par l'équation A.2 [114] où celui-ci est limité par une valeur maximale η_{max} dépendant de la topologie du réseau de capacités la tension d'entrée et la tension de sortie.

7. Low DropOut : faible chute en sortie
8. Switch Capacitor Network

$$\eta_{max} = \frac{1}{M} \cdot \frac{V_{Out}}{V_{In}} \tag{A.2}$$

Où M est le taux de conversion du réseau de capacité sans charge.

Dans [166] De Vos et al. ont implémenté un convertisseur DC-DC à base de capacités commutées capable de sortir des tensions de 0,3-0,4V en sortie pour des tensions d'entrée de 1-1,2V avec des rendements de 74% pour des charges de $100\mu W$ et 63% pour des charges de $100nW$.

Ce qui est très important pour des convertisseurs dans le domaine des nœuds de capteurs c'est d'avoir des rendements très importants pour une large gamme de puissance demandée par la charge. Ceci est due au rapport cyclique de fonctionnement du nœud passant d'un mode endormi consommant quelque nW à un mode actif consommant quelque μW au mW. Les convertisseurs DC-DC à capacités commutées remplissent très bien ce rôle et permettent d'avoir de très bon rendements. Ils ont toutefois de mauvaises performances en bruit à cause des commutations des transistors. Un convertisseur linéaire peut être utilisé en sorti d'un convertisseur à capacités commutées pour avoir une grande stabilité de la tension de sortie.

Il existe dans le commerce une multitude de module de gestion de l'énergie qui convertisse l'énergie et gère l'énergie des récupérateurs pour recharger la batterie et la distribuer à la charge [155] [108]. Le BQ25570 [155] dont l'architecture est présenté figure A.11 est capable de récupérer l'énergie d'un récupérateur de type cellule solaire, générateur thermoélectrique ou piézoélectrique, de recharger la batterie et de fournir en sortie des tensions de 1,3 à 5V pour un courant de sortie pic de 110mA. Ce gestionnaire utilise un convertisseur Boost pour recharger la batterie et un Buck pour convertir l'énergie provenant de la batterie rechargeable ou du récupérateur vers la sortie. Il possède aussi un traqueur de point maximale de puissance pour extraire du récupérateur la puissance maximale (MPPT [9]).

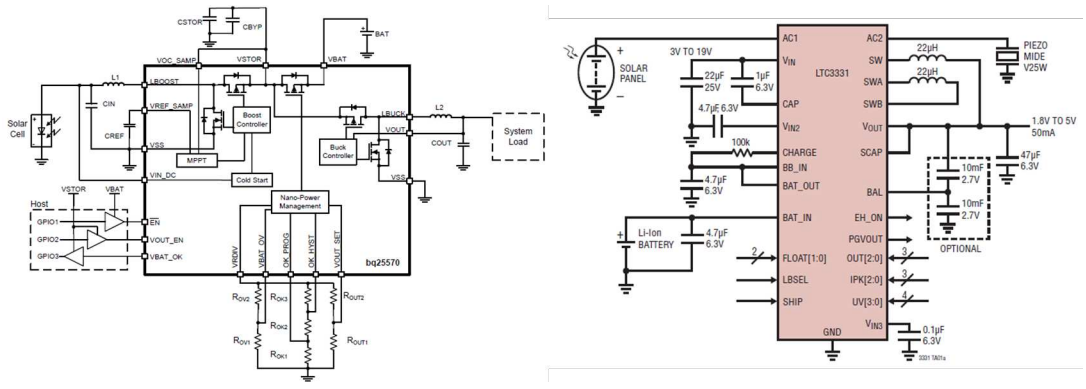


FIGURE A.11 – Gestionnaire d'énergie BQ25570 [155] et LTC3331 [108]

Le LTC3331 [108] dont un exemple d'application est montré figure A.11 est capable de récupérer de l'énergie de deux récupérateurs, de recharger une batterie et de fournir en sortie des tensions de 1,8-5V avec 50 mA de courant de charge maximum. Il emploie aussi en interne des convertisseurs Boost et Buck pour convertir l'énergie des récupérateurs vers la batterie ou la sortie et un Buck pour convertir l'énergie de la batterie ou des récupérateurs vers la sortie.

9. Maximum Power Point Tracking

A.6 Présentation de la technologie CMOS et des sources de consommation dans les circuits CMOS

Les circuits numériques modernes ont pour base le transistor à effet de champs utilisant une structure métal-oxyde-semiconducteur (MOSFET [10]), représenté figure A.12 pour un MOSFET de type N. Il est composé de deux électrodes qui sont le drain et la source dont la conduction entre elles s'effectue à travers un canal piloté par l'application d'une tension entre la grille et le substrat. Les électrodes du NMOS sont enrichies avec des atomes donneurs d'électrons alors que les PMOS sont enrichies avec des atomes donneurs de trous. Le transistor est utilisé fonctionnellement comme un interrupteur (mais possède d'autres modes de conduction) :

- Si $V_{GS} < V_{TH}$ avec V_{TH} sa tension de seuil, alors le transistor est bloqué.
- Si $V_{GS} > V_{TH}$ et que $V_{DS} > V_{DSat}$ alors le transistor est en mode saturé et se comporte comme une source de courant idéale. Dans un NMOS le canal se créé dans le substrat P juste à l'interface entre l'oxyde et le semi-conducteur dans une zone dite d'inversion composé à ce moment-là de charge négative.

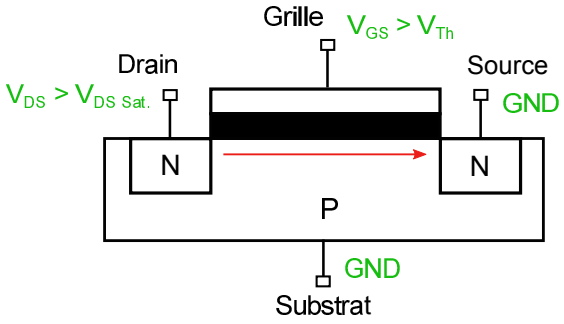


FIGURE A.12 – Transistor MOSFET type N

La technologie CMOS [11] permet de réaliser des traitements complexes en s'appuyant de la complémentarité des transistors NMOS et PMOS. Les transistors sont considérés comme des interrupteurs dont leur potentiel V_{DD} et GND sont assimilé à des niveaux hauts et bas. Il est alors possible de créer des opérateurs logiques de base en agençant en série et/ou parallèle ces transistors dont un exemple est donné figure A.13 pour un inverseur. Celui-ci convertie un niveau haut en niveau bas et inversement. Lorsqu'en entrée une tension plus élevé que la tension de seuil du NMOS (i.e V_{DD}) est appliqué alors le NMOS est passant et tire la sortie à GND, et si l'entrée est appliqué à GND alors le PMOS devient passant et tire la sortie à VDD.

L'avantage de la technologie CMOS est le pilotage de sa commutation en tension et non en courant comme pour les transistors bipolaire. Le canal est isolé de l'électrode de commande ce qui permet une faible consommation. Néanmoins il existe des sources de consommation dues au fonctionnement des cellules CMOS et aux imperfections des transistors. Ces sources de consommation de courant peuvent être séparées en une composante dynamique et une composante statique. En reprenant l'exemple de l'inverseur de la figure A.13 , il est possible d'identifier deux phases associées au changement d'état logique qui sont :

- Le courant de court-circuit I_{SC} due au basculement de la polarisation bloquée à la polarisation saturée et inversement qui n'est pas instantané (figure A.14 a)

10. Metal-Oxide-Semiconductor Field-Effect Transistor
11. Complementary MOS

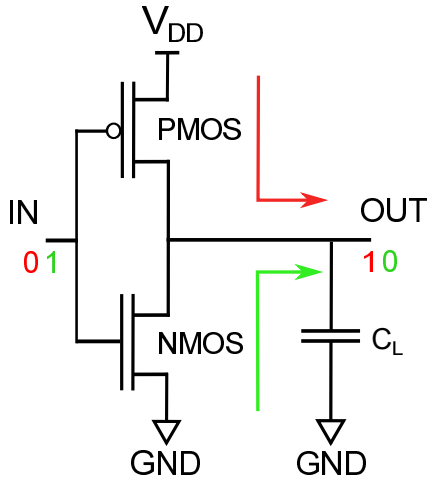
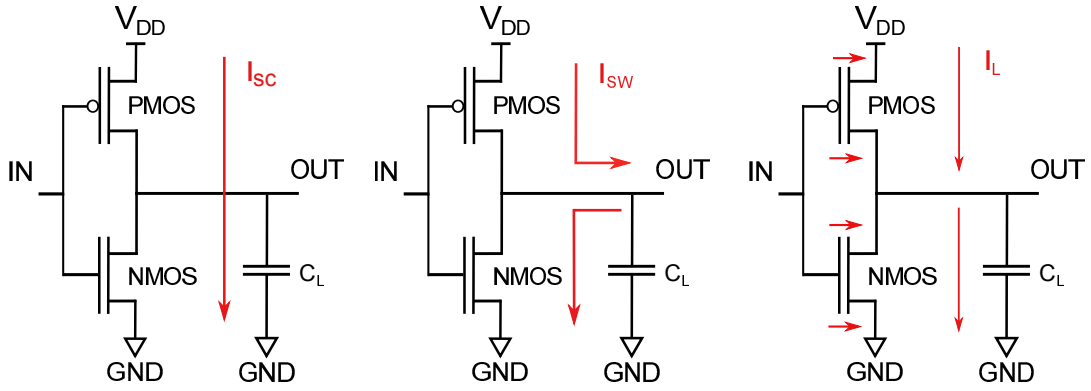


FIGURE A.13 – Inverseur CMOS

- Le courant de commutation I_{SW} lié à la charge et à la décharge des capacités de grille des transistors ramené à une valeur équivalente C_L (figure A.14 b)

I_{SC} dépend du dimensionnement des éléments actifs et du temps de commutation Δ_t alors que I_{SW} dépend du nœud technologique, de la structure du circuit et l'activité interne. La composante statique est due aux imperfections des transistors qui entrainent des courants de fuites multiples I_L (figure A.14 c). Ces courants de fuites proviennent du passage d'électrons à travers l'isolant sous la grille ou vers le substrat. Avec la réduction de la largeur des grilles et de l'épaisseur de l'isolant, les courants de fuites ne sont plus du tout négligeable dans les technologies avancées.



(a) Courant de court-circuit (b) Courant de commutation (c) Courant statique

FIGURE A.14 – Sources de consommation dans les circuits CMOS

Une puissance moyenne du circuit peut être définie en fonction de ces trois composantes et qui est donné par l'équation A.3 et A.4. V_{DD} est la tension d'alimentation, $I_{SC_{max}}$ le courant de court-circuit maximal, f la fréquence de fonctionnement du circuit et α le facteur d'activité qui modélise la probabilité de changement d'état du nœud de sortie.

$$P_{Average} = P_{Short-Circuit} + P_{Switch} + P_{Leakage} \tag{A.3}$$

$$P_{Average} = \alpha \frac{1}{2} \Delta_t I_{SC_{max}} V_{DD} f + \alpha C_L V_{DD}^2 f + V_{DD} I_L \tag{A.4}$$

De part cette équation, une réduction de la consommation des circuits CMOS passe par plusieurs facteurs :

- $\searrow V_{DD}$
- $\searrow \alpha$
- $\searrow f$
- $\searrow C_L$
- $\searrow I_{SC_{max}}$
- $\searrow I_L$

Cela se fait de manière évidente par la réduction de la tension d'alimentation V_{DD} qui est proportionnel à son carré pour la consommation de commutation et de manière linéaire pour la consommation statique. Cette consommation dynamique est fortement liée à la vitesse de fonctionnement des transistors et donc de la fréquence f de fonctionnement et de l'activité α. Il faut donc définir conjointement le point d'opération tension/fréquence pour minimiser le temps Δ_t de court-circuit et donc de diminuer les temps de montée et de descente des signaux. Au niveau technologique, la capacité C_L peut être diminuée en jouant sur la taille des grilles mais au détriment des courants de fuites I_L. $I_{SC_{max}}$ peut être diminué en optimisant le dimensionnement des transistors mais est contrainte par la capacité de la cellule à piloter les étages des transistors suivants.

Les courants de fuites I_L sont composés de trois composantes comme montré figure A.15 et nommé comme suit :

- les courants de conduction sous le seuil (subthreshold conduction)
- les courants de fuite grille-oxyde (gate-oxide leakage)
- les courants de fuite de jonction (junction leakage)

En réduisant la longueur du canal des transistors, la tension d'alimentation V_{DD} et la tension de seuil V_{TH} ont pu être diminuées. La diminution de la tension de seuil implique un très faible écart en dessous du seuil pour bloquer complètement le transistor. Ce faible écart engendre un courant de conduction sous le seuil et est de plus en plus présent au fur et à mesure que la longueur de canal du transistor diminue, et que V_{TH} diminue. Les courants de fuites grille-oxyde augmentent avec la diminution de l'épaisseur de l'isolant SiO2 ce qui implique des courants de fuites de la grille à travers l'oxyde vers le bulk, la source et le drain du transistor. Les courants de jonctions sont dues à la diode polarisée en inverse entre le drain-bulk et source-bulk.

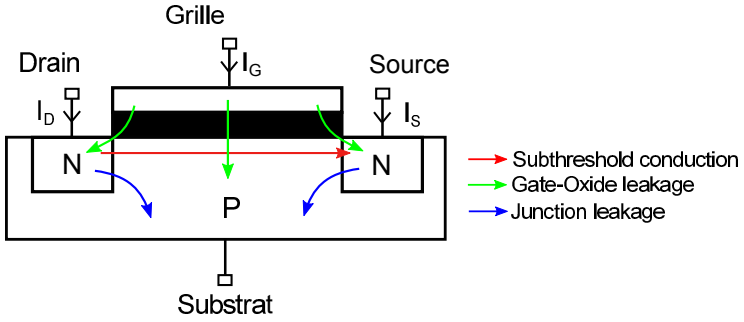


FIGURE A.15 – Courants de fuites dans un transistor NMOS

Finalement sur les aspects technologiques, la finesse de gravure permet d'abaisser l'énergie dissipée par le circuit en réduisant la surface des grilles et donc en diminuant la capacité effective C_L à charger ou décharger pendant un changement d'état du transistor. De plus les interconnexions sont de plus en plus courtes ce qui permet de réduire l'impédance et permet de diminuer les temps de montée et de descente des signaux liés à

I_{SC}. Malheureusement la taille de plus en plus petite des transistors implique des pertes par fuite de plus en plus grandes et une variabilité dans les procédés de fabrication de plus en plus importante. Une des techniques existante qui est utilisée pour réduire les pertes par fuites est d'utiliser un autre type de substrat qui est le silicium-sur-isolant [12], une technologie qui est utilisée dans cette thèse et expliquée section 3.3.

De plus, diminuer la tension de seuil V_{TH} permet d'augmenter les performances tandis que l'augmenter permet de réduire la consommation. Elle peut être modulée en jouant sur l'épaisseur d'oxyde et le dopage. Il est possible ainsi de créer plusieurs librairies de cellules logiques pour une même technologie avec des tensions de seuil différentes et qui vont être utilisées pour des applications différentes. Il existe trois types de librairies en fonction du V_{TH} :

- HVT [13], haut V_{TH} pour une faible consommation
- RVT [14], V_{TH} classique pour des besoins à la fois de performances en vitesse et consommation
- LVT [15], faible V_{TH} pour des performances en vitesse élevées

Ces librairies peuvent être mixées sur un même circuit en fonction du besoins applicatif du bloc [50].

12. Silicon On Insulator
13. High V_{TH}
14. Regular V_{TH}
15. Low V_{TH}

B Exemples de description de modules asynchrones en SystemVerilog

B.1 Machine d'états

Le WUC possède des machines d'états et voici un exemple de codage d'une machine d'états qui est utilisée dans son contrôleur d'interruptions. La variable d'état *state* est décrite en tant que *static* dans le processus. Cette machine possède cinq états ou chaque état est sensible à une combinaison d'évènements pour traiter quelque chose et passer à un autre état.

```
always
  begin: FSM_IT_ctrl
  static bit3 state = `BOOT;
  bit4 irq_num;
  bit it_ctrl_en;
  event_type it_ended;
  event_type new_cfg;

  // pragma translate off
  @(posedge resetn);
  // pragma translate on
  state = `BOOT;
  while(1)
  begin
    unique case (state)
      `BOOT: begin
        new_cfg_ev.BeginRead(new_cfg);
        power_state_c.Write(`WAIT_ACT_MODE);
        state = `SEND_NUM_IT; // for excuting boot code
        new_cfg_ev.EndRead();
      end
      `SLEEP: begin
        wait(new_irq_num_e.Ready || new_cfg_ev.Ready);
        unique case(1'b1)
          new_irq_num_e.Ready: begin
            power_state_c.Write(`WAIT_ACT_MODE);
            state = `SEND_NUM_IT;
            new_irq_num_e.EndRead();
          end
          new_cfg_ev.Ready: begin
            cfg_r.BeginRead(it_ctrl_en);
            if(it_ctrl_en == 1'b0) begin
              state = `IT_CTRL_DISABLE;
            end
            else begin
              state = `SLEEP;
            end
            cfg_r.EndRead();
            new_cfg_ev.EndRead();
          end
        endcase
      end
      `SEND_NUM_IT: begin
        irq_num_active_r.BeginRead(irq_num);
```

```
        if(irq_num[3] == 1'b1) begin //No interrupt to execute
          power_state_c.Write(`SLEEP_MODE);
          state = `SLEEP;
        end
        else begin
          fork
            Irq_num_act_o.Write(irq_num[2:0]);
            Irq_num_act_e.Write(SREVENT);
          join
          state = `WAIT_END_IT;
        end
        irq_num_active_r.EndRead();
      end
      `WAIT_END_IT: begin
        IT_ended_i.BeginRead(it_ended);
        state = `SEND_NUM_IT;
        IT_ended_i.EndRead();
      end
      `IT_CTRL_DISABLE: begin
        new_cfg_ev.BeginRead(new_cfg);
        state = `SLEEP;
        new_cfg_ev.EndRead();
      end
    endcase
  end
end
```

B.2 Split, Merge, Dup, Mutex

En asynchrone beaucoup de structures (*split*, *join*, *dup*, *merge*, etc.) existent pour manipuler les canaux de communication, rediriger les données, les séparer, dupliquer les canaux. Ici quelques exemples de structures beaucoup utilisées dans le WUC ainsi que leur code SystemVerilog associé sont présentés.

Le *Split* ou démultiplexeur permet de rediriger la donnée d'un canal vers plusieurs canaux mais est conditionné par un signal de contrôle qui spécifie sur quel canal de sortie l'envoyer. Le code montre un rendez-vous entre le signal de contrôle et la donnée d'entrée grâce au *fork-join*.

```
always
  begin: Split_Reg_Bus2
  bit32 op_bus2;
  bit2 split_cmd;

  fork
    Split_Bus2_Ctrl_i.BeginRead(split_cmd);
    Bus2_i.BeginRead(op_bus2);
  join
  unique case(split_cmd)
    `LD_ST_UNIT_BUS2: begin
      Ld_St_op2_o.Write(op_bus2);
    end
    `ALU_UNIT_BUS2: begin
      Alu_op2_o.Write(op_bus2);
    end
    `FUNC_UNIT_BUS2: begin
```

```
      Func_op2_o.Write(op_bus2);
    end
  endcase
  fork
    Split_Bus2_Ctrl_i.EndRead();
    Bus2_i.EndRead();
  join
end
```

La structure *Merge* est en attente de plusieurs canaux d'entrée afin de rediriger la donnée vers un seul canal de sortie. Pour que cette structure fonctionne il faut absolument que les canaux d'entrée soit mutuellement exclusifs.

```
always
  begin: merge_alu_op1
  bit8 imm8;
  bit32 data;

  wait(Imm8_alu_i.Ready || Reg_File_Bus1_i.Ready);
  unique case(1'b1)
    Imm8_alu_i.Ready: begin
      Imm8_alu_i.BeginRead(imm8);
      Alu_op1_o.Write({24'b0, imm8});
      Imm8_alu_i.EndRead();
    end
    Reg_File_Bus1_i.Ready: begin
      Reg_File_Bus1_i.BeginRead(data);
      Alu_op1_o.Write(data);
      Reg_File_Bus1_i.EndRead();
    end
  endcase
end
```

La structure *Duplicate* permet de dupliquer un canal en plusieurs autres canaux. Pour que le canal d'entrée du *Duplicate* soit acquitté il faut que tous les canaux sortant soient acquittés.

```
push_channel_bit4    alu_status_s (); //duplicate for decoder and branch
    unit
push_channel_bit4  status_bit_dup_c [1:0] ();
dup_push_bit4 dup_status_bit(alu_status_s, status_bit_dup_c);
```

Enfin il est quelquefois nécessaire d'être sûr que plusieurs canaux soient mutuellement exclusifs pour pouvoir les traiter. C'est le cas des interruptions par exemple, il est possible que plusieurs interruptions soient en attente en même temps. Or pour les lire, une structure comme le *Merge* est incapable de sélectionner un signal d'interruption parmi toutes celles qui sont en attente car il faut que toute les entrées soient mutuellement exclusives. C'est pourquoi une structure nommé *Mutex* a été développée pour pouvoir rendre ces signaux mutuellement exclusifs. Elles prennent en entrée un vecteur de canaux qui ne sont pas mutuellement exclusif et garantit de ne ressortir qu'une donnée à la fois parmi les canaux de sortie. Ici l'exemple de la gestion des ITs à l'entrée du contrôleur d'interruptions pour rendre les ITs mutuellement exclusives est montré.

```
module IT_ctrl_8_chan
(
  /********** Data in ***************/
  //All peripherals IT
```

```
  push_channel_bit.in IT_i [7:0],
  ...
  (*ACC_Reset*) input bit    resetn
);

...

push_channel_bit IT_mutex [7:0] ();
mutex_push_bit #(8) Mutex8 (IT_i, IT_mutex);

...

endmodule
```

B.3 Adder-Sub dans l'ALU

Pour finir voici un exemple de processus pour traiter l'opération d'addition et de soustraction dans l'ALU du WUC. On peut voir que les outils de Tiempo mettent à notre disposition des structures de bases comme le "+" qui est synthétisé automatiquement par ACC.

```
always
  begin: Arith_Unit
  bit33 src;
  bit33 dst;
  bit2  op_inst;
  bit n, z, c, v;
  bit33 res;
  bit src_inv;

  fork
    src_arith_unit.BeginRead(src);
    dst_arith_unit.BeginRead(dst);
    opcode_arith_unit.BeginRead(op_inst);
    src_inv_arith.BeginRead(src_inv);
  join
  //Execution
  res = dst + src + {32'h00000000, src_inv};
  //Generation of status bit
  fork
    if(res[31:0] == 0) begin
      z = 1'b1;
    end
    else begin
      z = 1'b0;
    end
    n = res[31];
    c = res[32];
    v = (src[31] & dst[31] & ~res[31]) | (~src[31] & ~dst[31] & res
   [31]);
  join
  //Assign Output
  fork
    out_arith_unit.Write(res[31:0]);
    status_arith_unit.Write({n, z, c, v});
```

```
  join

  fork
    src_arith_unit.EndRead();
    dst_arith_unit.EndRead();
    opcode_arith_unit.EndRead();
    src_inv_arith.EndRead();
  join
end
```

C Schématic carte WALPP

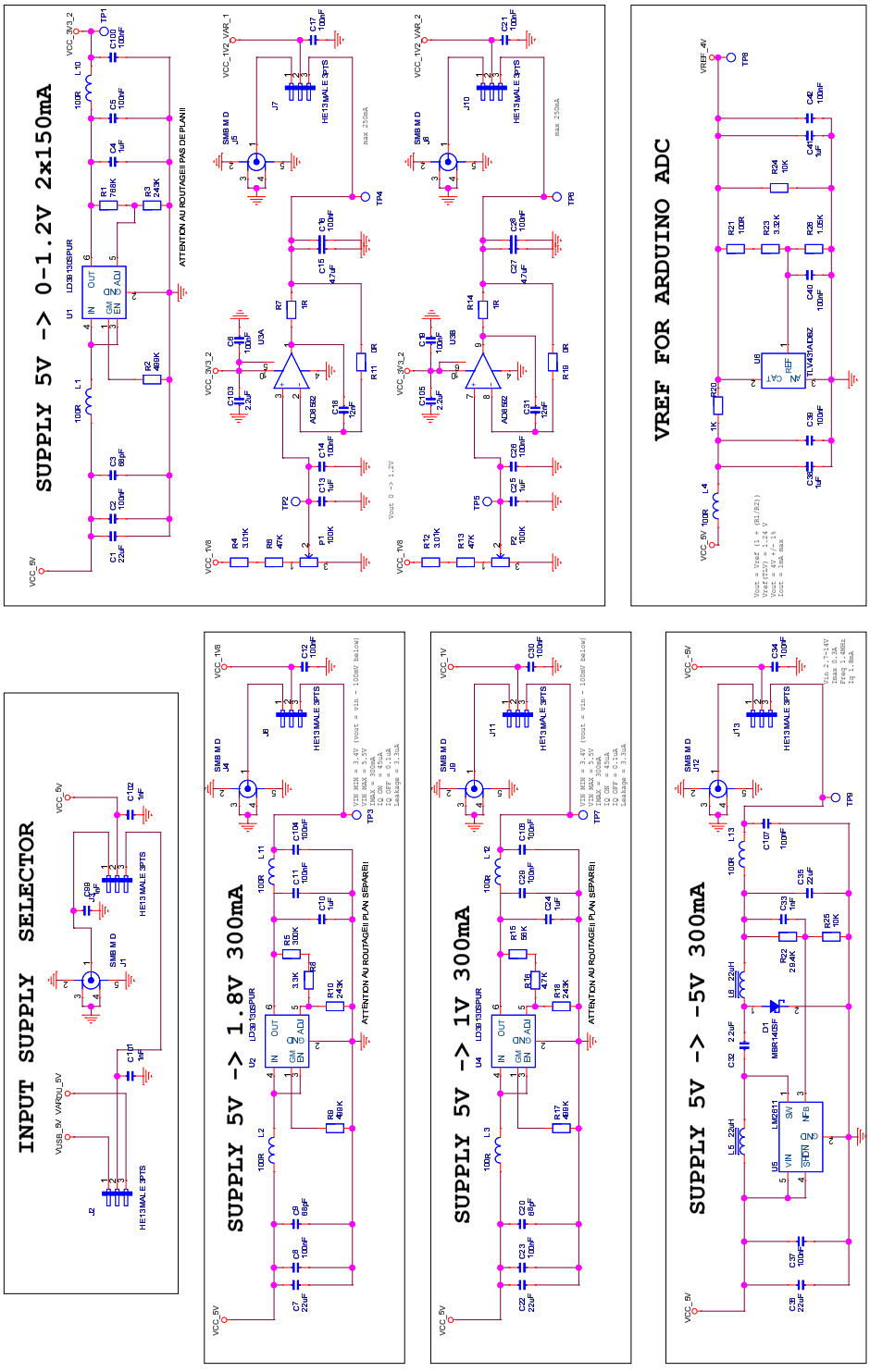


FIGURE C.16 – Schematic 1/3 de la carte WALPP

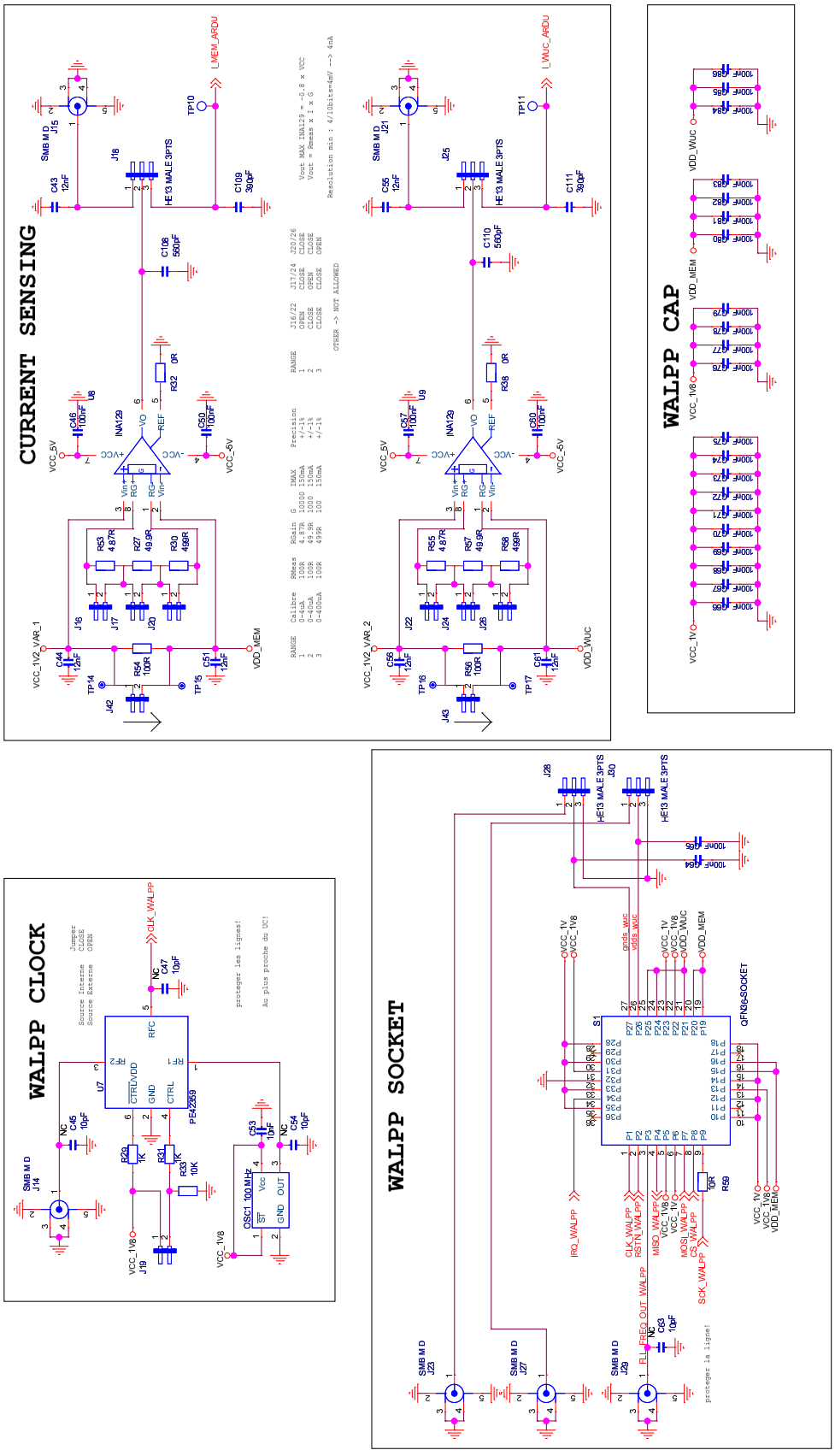


FIGURE C.17 – Schematic 2/3 de la carte WALPP

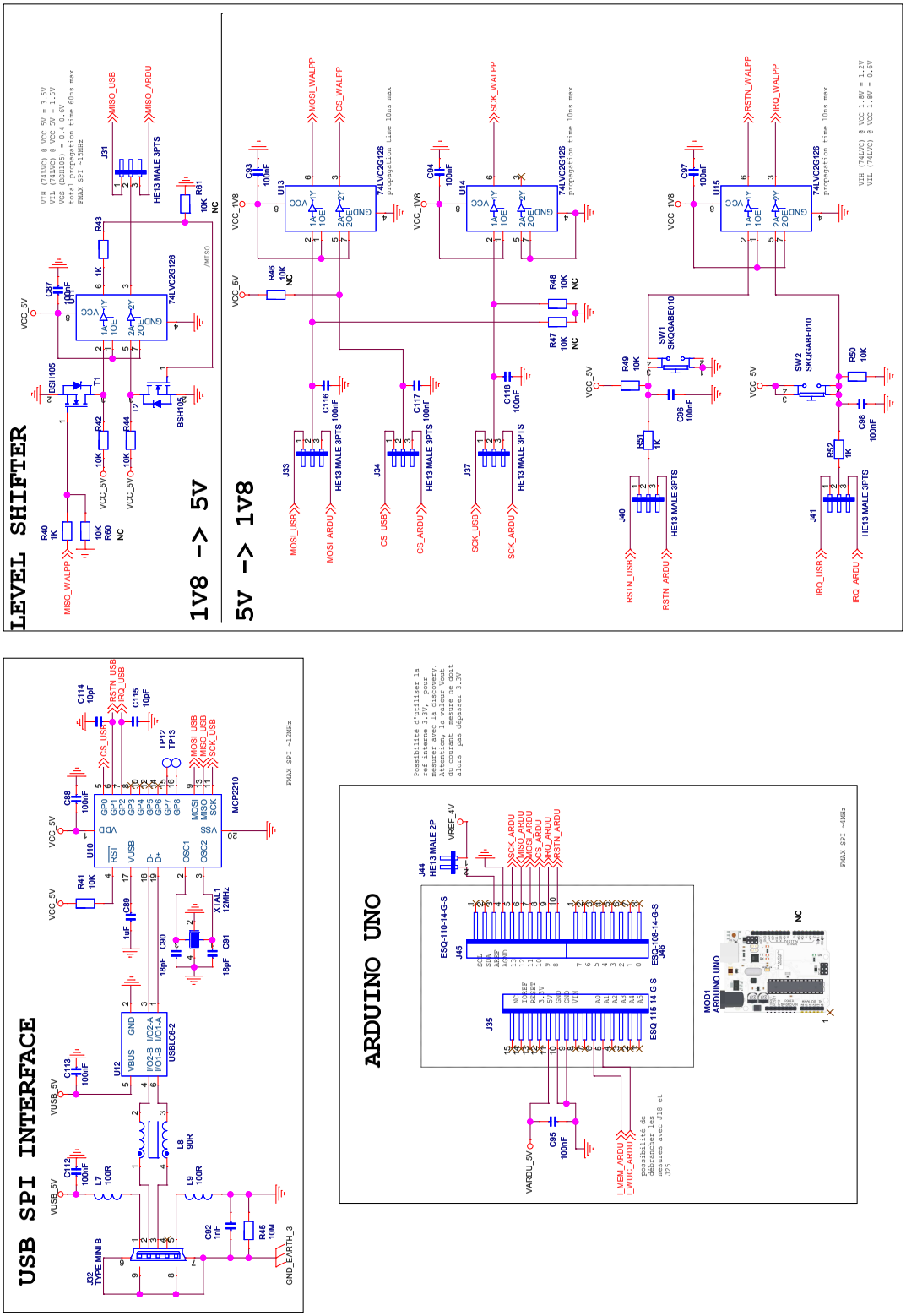


FIGURE C.18 – Schematic 3/3 de la carte WALPP

D Top de la carte de test WALPP

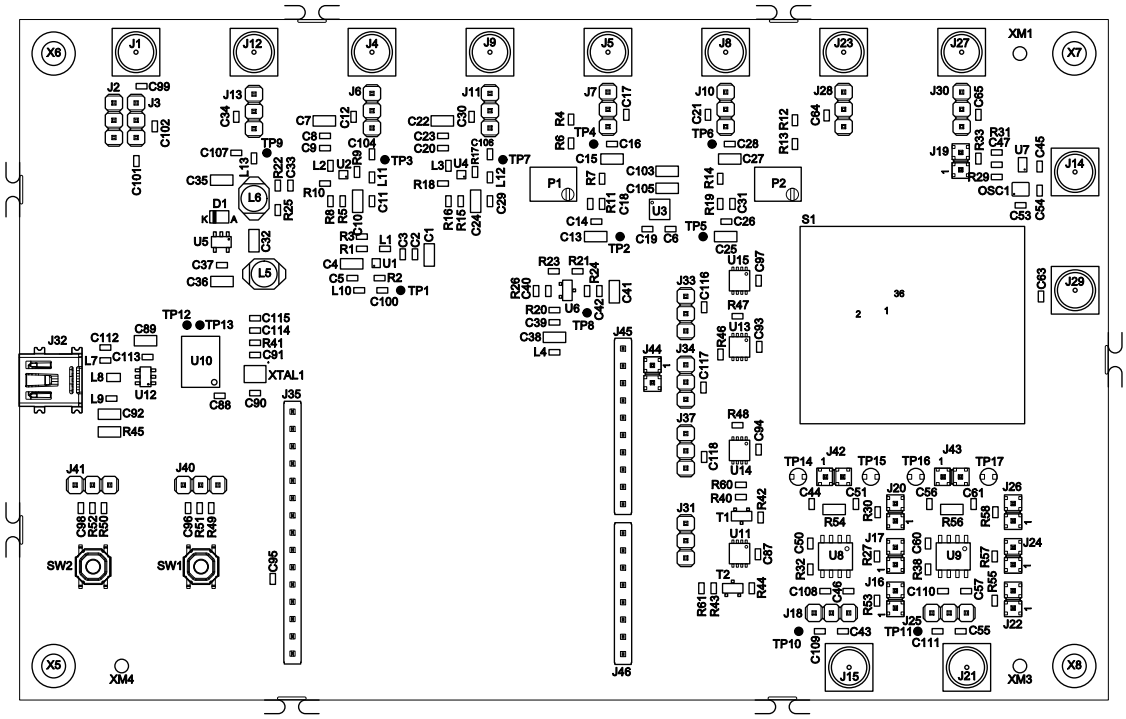


FIGURE D.19 – Top de la carte de test du circuit WALPP

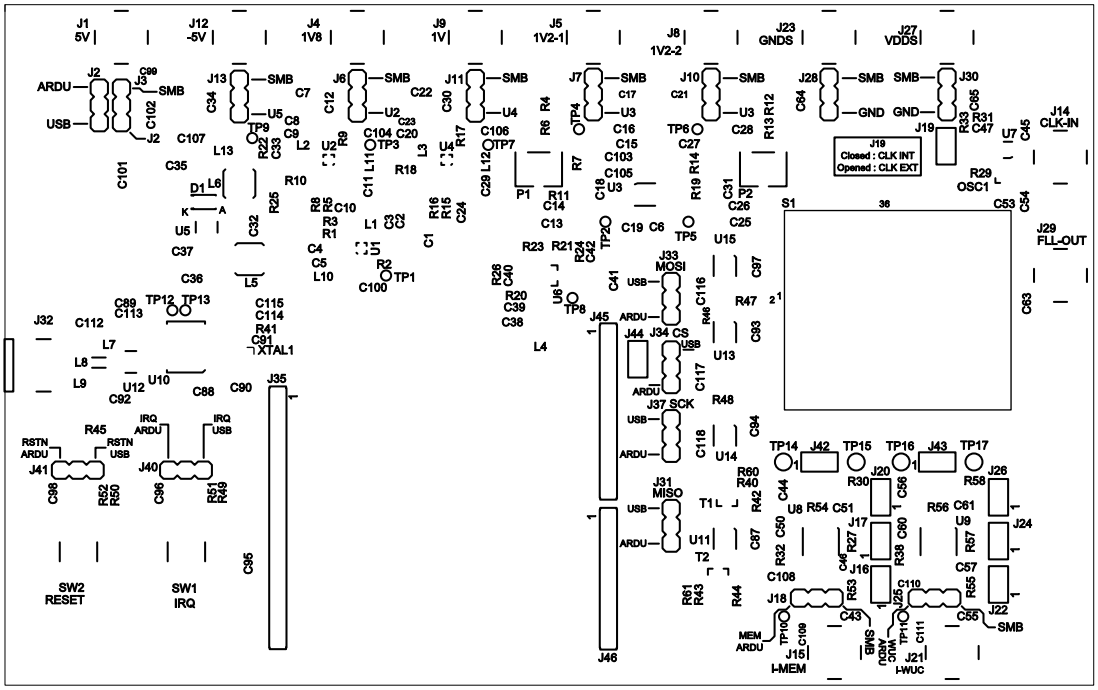


FIGURE D.20 – Sérigraphie de la carte WALPP

Conception d'un processeur ultra basse consommation pour les nœuds de capteurs communicants

Les travaux de cette thèse se concentrent sur la réduction de l'énergie consommée et l'amélioration des temps de réveil du microcontrôleur par des innovations au niveau de l'architecture, du circuit et de la gestion de l'énergie. Ces travaux proposent une architecture de microcontrôleur partitionnée entre un processeur de réveil programmable, appelé Wake Up Controller, s'occupant des tâches courantes du nœud de capteurs et un processeur principal gérant les tâches irrégulières.
Le Wake Up Controller proposé dans ces travaux de thèse est un processeur RISC 16-bit dont le jeu d'instructions a été adapté pour gérer les tâches régulières du nœud, et n'exécute que du code sur interruptions. Il est implémenté en logique mixte asynchrone/synchrone. Un circuit a été fabriqué en technologie UTBB FDSOI 28nm intégrant le Wake-Up Controller. Le cœur atteint une performance de 11,9 MIPS pour 125μW de consommation moyenne en phase active et un réveil depuis le mode de veille en 55ns pour huit sources de réveil possibles. La consommation statique est d'environ 4μW pour le cœur logique asynchrone à 0,6V sans utilisation de gestion d'alimentation (power gating) et d'environ 500nW avec.

Design of an ultra low power processor for wireless sensor nodes

This PhD work focuses on the reduction of energy consumption and wake up time reduction of a WSN node microcontroller through innovations at architectural, circuit and power management level. This work proposes a partitioned microcontroller architecture between a programmable wake up processor, named Wake Up Controller on which this work is focused, and a main processor. The first deals with the common tasks of a wireless sensor node while the second manages the irregular tasks.
The Wake Up Controller proposed in this work is a 16-bit RISC processor whose instruction set has been adapted to handle regular tasks of a sensor node. It only executes code on interruptions. It is implemented in asynchronous / synchronous mixed logic to improve wake up time and energy. A circuit was fabricated in a 28nm UTBB FDSOI technology integrating the Wake Up Controller. The core reaches 11,9 MIPS for 125 μW average power consumption in active phase and wakes up from sleep mode in 55ns from eight possible interruption sources. The static power consumption is around 4μW for the asynchronous logic core at 0.6V without power gating and 500nW when gated.