UNIVERSITE D'ANTANANARIVO

ECOLE SUPERIEURE POLYTECHNIQUE D'ANTANANARIVO

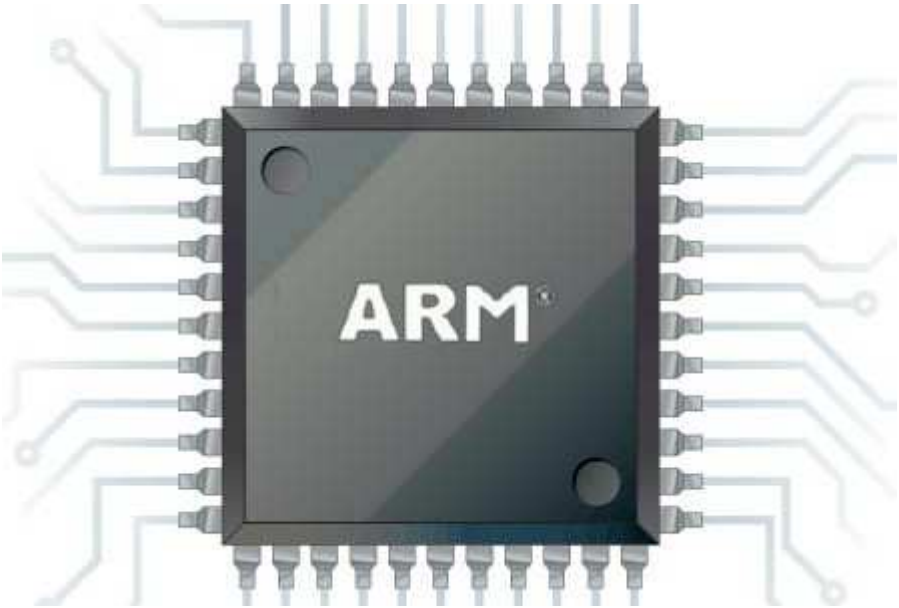
DEPARTEMENT ELECTRONIQUE

MEMOIRE EN VUE DE L'OBTENTION DU DIPLOME DE MASTER 2

Spécialité : ELECTRONIQUE

Option : Electronique Automatique

Développement sur architecture ARMv6-M / ARMv7-M



Présenté par : ANDRY-NANJA Jerry Stéphane

N° d'ordre :

Soutenu le : 1er juillet 2015

Année Universitaire : 2013 - 2014

UNIVERSITE D'ANTANANARIVO

ECOLE SUPERIEURE POLYTECHNIQUE D'ANTANANARIVO

DEPARTEMENT ELECTRONIQUE

MEMOIRE EN VUE DE L'OBTENTION DU DIPLOME DE MASTER 2

Spécialité : ELECTRONIQUE

Option : Electronique Automatique

Développement sur architecture ARMv6-M / ARMv7M

Présenté par :

ANDRY-NANJA Jerry Stéphane

Devant le Jury composé de :

Mme. RABEHERIMANANA Lyliane Irène : Président du jury
Mr. RATSIMBAZAFY Guy Prédon Claude : Examinateur
Mr. RANDRIAMAROSON Rivo Mahandrisoa : Examinateur
Mr. HERINANTENAINA Edmond Fils : Examinateur

Rapporteur :

Mr. ANDRIAMANANTSOA Guy Danielson

Soutenu le : 1er juillet 2015

REMERCIEMENT

Je remercie Dieu Tout puissant de m'avoir donné la force et la santé tout au long de l'accomplissement de ce mémoire.

Je tiens également à exprimer ma reconnaissance à tous ceux qui on apporté leur contribution dans la réalisation de ce mémoire, en particulier :

Monsieur RATSIMBA Mamy Nirina, Chef du Département Electronique de l'ESPA, qui a consacré ses efforts pour le bon déroulement de ma formation durant toutes ces années.

Mon encadreur Monsieur ANDRIAMANANTSOA Guy Danielson, qui m'a dirigé tout au long du travail.

Madame RABEHERIMANANA Lyliane Irène, qui a voulu présider la soutenance de ce mémoire malgré ses occupations.

Aux membres du jury

- Monsieur RATSIMBAZAFY Guy Prédon Claude
- Monsieur RANDRIAMAROSON Rivo Mahandrisoa
- Monsieur HERINANTENAINA Edmond Fils

qui ont consacré leur temps pour l'amélioration de ce mémoire :

Aux enseignants du Département Electronique pour les connaissances qu'ils m'ont transmis durant ces cinq dernières années.

A ma famille qui m'a soutenu moralement et financièrement durant la réalisation de ce mémoire et durant toutes ces années d'études.

Et enfin, à toutes les personnes qui ont participé de prés ou de loin à l'élaboration de ce mémoire.

JERRY

RESUME

Aujourd'hui, l'utilisation de microprocesseur est devenue très importante dans les appareils technologiques. Ce mémoire est consacré sur le développement des microprocesseurs ARM Cortex-M3/M4 dans les systèmes embarqués. Les généralités sur ces types de processeurs et l'étude concernant la technologie interne du microprocesseur ARM Cortex-M3/M4 sont mises en valeur dans cet ouvrage. De plus, les outils de programmations et la manière de programmer sur ces microprocesseurs, basés essentiellement autour du langage C, y sont développés. Grace à ces études, on a pu réaliser un logiciel embarqué sur ce microprocesseur, pour l'acquisition et le traitement de données venant des capteurs et l'acheminement de ces données via internet en temps réel.

TABLE DES MATIERES

LISTE DES ABREVIATIONS

ADC: Analog to Digital Converter
AHB: Advanced High-performance Bus
AHB-AP: AHB Access Port
AHP: Alternate Half-Precision
ALU: Arithmetic Logic Unit
AMBA: Advanced Microcontroller Bus Architecture
APB: Advanced Peripheral Bus
API: Application Programming Interface
APSR: Application PSR
ARM Ltd: Advanced RISC Machines Limited
ATB: Advanced Trace Bus
AXI: Advanced eXtensible Interface
BBC: British Broadcasting Corporation
CAN: Controller Area Network
CLB: Configurable Logic Block
CMOS: Complementary Metal Oxide Semi-conductor
CMSIS: Cortex Microcontroller Software Interface Standard
CPACR: Co-Processor Access Control Register
CPU: Central Processor Unit
CRC: Cyclic Redundancy Check
DAC: Digital to Analog Converter
DAP: Debug Access Port
DCT: Discrete Cosine Transform
DMA: Direct Memory Access
DMIPS: Dhrystone Millions Instructions Per Second
DN: Default NaN (Not A Number)
DSC: Digital Signal Controller
DSP: Digital Signal Processing/Digital Signal Processor
DWT: Data Watchpoint and Trace
DZC: Division by Zero Cumulative
E/S: Entrée/ Sortie
ECC: Error Correcting Code
EEPROM: Electrically Erasable Programmable ROM
EPSR: Execution PSR
ETM: Embedded Trace Macrocell
FFT: Fast Fourier Transform

FIR: Finite Impulse Response
FPB: Flash Patch and Breakpoint
FPCA: Floating Point Context Active
FPGA: Field Programmable Gate Array
FPSCR: Floating Point Status Control Register
FPU: Floating Point Unit
FZ: Flush-to-Zero
GNU: GNU's Not Unix
GPIO: General Purpose Input Output
GPU: Graphics Processor Unit
HAL: Hardware Abstraction Layer
I/O: Input/ Output
I2C: Inter-Integrated Circuit
I2S: Inter-IC Sound
IC: Input Capture
IC: Integrated Circuit
ICE: In-Circuit Emulator
ICI: Interrupt-Continuable Instruction
ID: IDentifier
IDC: Input Denormal Cumulative
IDE: Integrated Development Environment
IEEE: Institute of Electrical and Electronics Engineers
IIR: Infinite Impulse Response
IOC: Invalid Operation Cumulative
IP: Internet Protocol
IP: Intellectual Property
IPSR: Interrupt PSR
IRQ: Interrupt ReQuest
ISR: Interrupt Service Routine
ITM: Instrumentation Trace Macrocell
JTAG: Joint Test Action Group
LCD: Liquid-Crystal Display
LIFO: Last In First Out
LR: Link Register
LSB: Least Significant Bit
LWIP: LightWeight Internet Protocol
MAC: Media Access Control
MAC: Multiply-ACcumulate

MCI: Multimedia Card Interface
MCU: Microcontroller Unit
MDK-ARM: Keil Microcontroller Development Kit ARM
MemManage: Memory Management
MII: Media Independent-Interface
MMU: Memory Management Unit
MPU: Memory Protection Unit
MSP: Main Stack Pointer
MSPS: Millions of Samples Per Second
MTB: Micro Trace Buffer
NMI: NonMaskable Interrupt
NVIC: Nested Vectored Interrupt Controller
OC: Output Capture
OFC: OverFlow Cumulative
OS: Operating System
OSI: Open Systems Interconnection
OTG: On –The-Go
PC: Program Counter
PendSV: Pendable request for System serVice
PHY: PHYsique
PID: Proportional Integral Derivative
PLL: Phase Locked Loop
PPB: Private Peripheral Bus
PSP: Process Stack Pointer
PSR: Program Status Register
PSRAM: PseudoStatic RAM
PWM: Pulse Width Modulation
RAM: Random Access Memory
RC: Resistance Capacité
RISC: Reduced Instructions Set Computer
RMII: Reduced MII
RMODE: Rounding Mode
ROM: Read-Only Memory
RTC: Real-Time Calendar/Clock
RTL: Register Transfer Level
RTOS: Real-Time Operating System
SCB: System Control Block
SCS: System Control Space

SDIO: Secure Digital Input Output
SIMD: Single Instruction, Multiple Data
SoC: System-on-Chip
SOIC: Small Outline Integrated Circuit
SOP: Sum Of Product
SOP: Small Outline Package
SPI: Serial Peripheral Interface
SPSEL: Stack Pointer SELection
SRAM: Static RAM
SVC: SuperVisor Call
SVD: System View Description
SW: Serial Wire
SWD: Serial Wire Debug
SW-DP: SW-Debug Port
SWJ-DP: SW JTAG – Debug Port
Systick: System tick
TCM: Tightly-Coupled Memory
TCP: Transmission Control Protocol
TPIU: Trace Port Interface Unit
TSOP: Thin SOP
UART: Universal Asynchronous Receiver Transmitter
UDP: User Datagram Protocol
UFC: UnderFlow Cumulative
USART: Universal Synchronous Asynchronous Receiver Transmitter
USB: Universal Serial Bus
VFP: Vector Floating Point
VLSI: Very Large Scale Integration
VTOR: Vector Table Offset Register
WIC: Wakeup Interrupt Controller
XML: eXtensive Markup Language
XN: eXecute Never
xPSR: IPSR+EPSR+APSR

LISTE DES FIGURES

LISTE DES TABLEAUX

INTRODUCTION

Aujourd'hui la technologie joue un rôle très important dans la vie quotidienne des gens. Quand l'humanité est entrée dans l'ère du numérique, les technologies ne cessent d'évoluer. Une des découvertes dans le domaine du numérique durant ces dernières décennies était le microprocesseur. En effet, il est l'une des principales technologies numériques responsables de cet essor. L'évolution des microprocesseurs a aujourd'hui donné naissance à plusieurs types de microprocesseurs dont les microprocesseurs ARM. Que ce soit dans l'iphone ou l'ipad d'Apple ou dans les terminaux concurrents de Samsung, HTC ou dans presque tous les fabricants de Smartphones, ou les appareils ménagers et les appareils industriels, une puce ARM peut y être intégrée et leur servir de cerveau.

Contrairement aux processeurs utilisés dans les ordinateurs personnels, les microprocesseurs ARM sont surtout utilisés dans les systèmes qui ont besoin de faible encombrement ou, pour les initiés, dans les systèmes embarqués. En effet, à cause de sa forte densité d'intégration et de sa taille minuscule, il est vraiment idéal pour des appareils de petite taille tout en gardant une puissance et une performance comparable aux processeurs présents dans certains ordinateurs personnels. Jusqu'à maintenant le microprocesseur ARM est le leader dans son domaine. La raison de ce succès n'est pas seulement due à sa performance ni à sa puissance mais surtout à la rentabilité économique que son exploitation offre.

L'objet de notre travail consiste à programmer et à développer sur l'architecture ARM. Plus précisément comme le titre l'indique : « Développement sur architecture ARM Cortex-M3/ M4 », l'étude se focalise sur l'architecture ARM Cortex-M3/M4 qui est l'une des architectures ARM destinée aux microcontrôleurs. L'étendue du travail ira jusqu'à l'exploitation de l'ARM Cortex-M4 dans le cadre de l'acquisition et du traitement de données en temps réel.

Le contenu du travail sera divisé en quatre chapitres. Le premier chapitre intitulé : « Les processeurs ARM Cortex » traite la généralité de ce type de microprocesseur. Puis, le second chapitre intitulé : « La technologie ARM Cortex-M3/M4 » donne une description détaillée du microcontrôleur ARM Cortex-M3/M4. Après, il y aura le troisième chapitre intitulé : « Programmation en C sur un microcontrôleur ARM Cortex-M3/M4 ». Enfin, le dernier chapitre intitulé : « Mise en place d'un RTOS pour les traitements de données en temps réel » présente une implémentation et une application concrète du microcontrôleur ARM Cortex-M4.

CHAPITRE I : LES PROCESSEURS ARM CORTEX

I.1. GENERALITE SUR LE PROCESSEUR ARM

Le processeur ARM (*Advanced RISC - Reduced Instruction Set Computer - Machines*) est un CPU (*Central Processing Unit*) « Unité centrale de traitement » qui possède comme cœur l'architecture ARM conçue par la société *ARM Ltd* ex-filiale de la société anglaise *Acorn Computers Ltd.*

La société *ARM Ltd* est aujourd'hui le leader mondial dans le développement d'architecture 32 bits basée sur le principe du RISC qui a pour but la limitation du nombre d'instructions du processeur et l'exécution simultanée de ces instructions. Ce principe permet une meilleure vitesse de traitement, une faible consommation d'électricité et une faible génération de chaleur [1]. La société *ARM Ltd* conçoit et fournit juste l'architecture du cœur du processeur mais elle ne produit pas directement les semi-conducteurs. Ce sont les grandes firmes comme *Texas Instruments, Apple, Qualcomm, Nvidia, STMicroelectronics, Freescale*…qui achètent les licences et produisent des processeurs à base ARM.

Les cœurs de processeurs ARM sont surtout utilisés dans les microprocesseurs pour les applications générales, dans les microcontrôleurs pour les systèmes embarqués. Ils sont principalement intégrés au sein de systèmes complets embarqués sur une seule puce SoC (*System-On Chip*). Les conceptions SoC sont généralement utilisées avec des applications nécessitant de hautes performances et une polyvalence dans un faible encombrement [1]. Ils peuvent travailler dans les SoC avec des coprocesseurs comme les:

- DSP (*Digital Signal Processing*) qui sont des processeurs utilisés pour traiter des signaux numériques. Ils sont optimisés pour effectuer des opérations mathématiques et arithmétiques nécessaires fréquemment et rapidement comme le SOP (Sum *of Product*), le calcul en virgule flottante, le calcul spécialisé dans le chiffrement et l'encodage. Ils sont conçus pour traiter de manière fiable, une quantité spécifique de données par unité de temps (capacité en temps réel) [1]. Les DSP peuvent être désormais utilisés dans les SoC.
- GPU (*Graphics Processing Unit*) qui sont des processeurs dédiés pour le calcul et le traitement des performances graphiques comme le rendu 2D/ 3D, l'éclairage, les *shaders*, les textures, les ombres…etc.
- FPGA (*Field Programmable Gate Array)* qui sont des composants VLSI (*Very-Large-Scale Integration*) entièrement reconfigurables. Ce sont des circuits logiques programmables basés autour d'une matrice de blocs logiques programmables appelés *Configurable Logic Blocks* (CLB). Ces derniers sont entourés de blocs d'entrée/sortie programmables. L'ensemble est relié par un réseau d'interconnexions programmables. Le progrès de ces technologies donne la possibilité de faire des

composants toujours plus rapides et à plus haute densité d'intégration, ce qui permet d'avoir plusieurs circuits logiques et de programmer des applications importantes sur une même puce [2].

I.2. HISTORIQUE

À la fin de 1978, Hermann Hauser et Chris Curry fondaient *Acorn Computers Ltd*. à Cambridge, au Royaume-Uni. Travaillant d'abord comme consultant, Hauser et Curry obtenaient un contrat d'*Ace Coin Equipment* pour développer une machine de fruits à base de microprocesseur. Après une recherche initiale et la première phase de développement, Hauser et Curry ont choisi le processeur 6502 de *MOS Technology* pour la conception de leur machine. Le 6502 a été produit en 1974, il était l'un des processeurs les plus fiables disponibles en ce temps, ce qui allait bientôt révolutionner l'industrie informatique. Le 6502 est un microprocesseur 8 bits qui était de loin le moins cher de l'époque, tout en restant comparable aux conceptions des concurrences. Il était facile à programmer et sa vitesse globale était acceptable. Sa conception était simpliste; il avait "seulement" 3510 transistors (*Intel 8085* en avait 6500, le *Zilog Z80* en avait 8500 et le *Motorola 6800* en avait 4100). Pour eux, c'était le processeur idéal pour les systèmes embarqués de l'époque.

Acorn Computers remportait plusieurs contrats et continuait à utiliser le 6502 pour ses projets, en misant sur sa maitrise de ce microprocesseur. Cette excellente connaissance de la 6502 lui permettait de pousser le processeur à ses limites et parfois même au-delà. L'expérience d'*Acorn* avec le 6502 était si légendaire qu'en 1983, il a même pu produire des ordinateurs personnels qui utilisaient le processeur 6502.

Cependant, ce microprocesseur devenait un vieux processeur alors que presque tous les projets d'*Acorn* avaient été faits sur un 6502. *Acorn* a été confronté à un problème majeur; en cherchant un nouveau processeur pour remplacer le vieux 6502, il a pu constater que les autres processeurs n'étaient pas tout simplement à la hauteur. De plus, les systèmes graphiques émergeaient, et il était clair que le 6502 ne pouvait pas suivre dans ce domaine Le Motorola 68000, un microprocesseur 16/32 bits qui a été utilisé dans de nombreux ordinateurs personnels et ordinateurs d'affaires ne suffisait même pas. En effet, son temps de réponse d'interruption étant lent, il ne pouvait pas suivre un protocole de communication que le 6502 aurait pu accomplir sans problème. Un par un, les processeurs sur le marché ont été étudiés, analysés et rejetés. Enfin, la liste était épuisée, et *Acorn* était laissé sans choix; s'il voulait faire les choses à sa manière, il aurait dû faire son propre processeur.

Le projet a été lancé en octobre 1983, avec Sophie Wilson comme concepteur du jeu d'instructions et Steve Furber concepteur matériel, avec l'aide de la firme *BBC Micros (British Broadcasting Corporation)* pour modéliser et développer la puce. Le 26 avril 1985, le premier processeur *Acorn RISC Machine* est né, l'ARM1. Il a également travaillé parfaitement la

première fois, ce qui était assez exceptionnel pour un processeur qui a été essentiellement conçu à la main.

Plus tard, *Apple Computers* a étudié le processeur ARM. *Apple* avait placé des exigences strictes pour son projet Newton qui nécessitait un processeur ayant les caractéristiques suivantes : consommation spécifique d'énergie, performance à moindre coût, habileté à s'arrêter complètement à tout moment par arrêt du système horloge. La conception d'*Acorn* se rapprochait le plus aux exigences d'*Apple* mais n'a pas tout à fait pu les remplir. Un certain nombre de changements étaient nécessaires, mais *Acorn* n'avaient pas les ressources nécessaires pour effectuer les changements. *Apple* a aidé *Acorn* à développer les exigences manquantes, et après une courte collaboration, il a été décidé que la meilleure solution serait de créer une nouvelle société, distincte pour le développement des architectures des processeurs. En Novembre 1990, avec un financement de *VLSI Technology*, *Acorn* et *Apple*, *ARM Ltd* (*Advanced RISC Machines Limited Company*) a été fondée [3].

I.3. LES FAMILLES DES PROCESSEURS ARM

a) Convention d'appellation des cœurs des processeurs ARM

Comme les noms des processeurs peuvent varier, tous les cœurs ARM partagent une convention d'appellation commune. Il y a eu deux principales conventions au cours de la durée de vie de l'architecture :

i. Les *ARM Classic cores*

Les premiers noyaux, appelés processeurs classiques ont le nom ARM {x} {labels}, mais plus tard l'adoption d'un nom de la forme ARM {x} {y} {z} {labels} était nécessaire à cause de l'apparition de plus de variantes. Le premier nombre (x) correspond à la version de base. Les deuxième et troisième nombres (y et z) correspondent respectivement aux informations sur le cache/MMU *(Memory Management Unit)/MPU (Memory Protection Unit)* et à la taille du cache. Le tableau I illustre la description de {x} {y} {z} et le tab II illustre les attributs de {labels} [3].

Pour certains ARM, le label *MPCore* signifie que le processeur est *multi-core* et peut varier jusqu'à 1-4 cœurs sur une même base RTL (*Register Transfer Level)* [4]. Les lettres H et Z signifient respectivement processeur sans horloge (*clockless processor*) et processeur ayant une extension de sécurité (*Trustzone*).

Dans l'annexe 1 se trouve les tableaux de plusieurs processeurs ARM avec leur architecture, leur famille et leur nom.

Tableau I **:** Numérotation des processeurs ARM [3]

X	Y	Z	DESCRIPTION	EXAMPLE
7			ARM7 core version	ARM7
9			ARM9 core version	ARM9
10			ARM10 core version	ARM10
11			ARM11 core version	ARM11
	1		Cache, write buffer and MMU	ARM710
	2		Cache, write buffer and MMU, Process ID support	ARM920
	3		Physically mapped cache and MMU	ARM1136
	4		Cache, write buffer and MPU	ARM940
	5		Cache, write buffer and MPU, error correcting memory	ARM1156
	6		No cache, write buffer	ARM966
	7		AXI bus, physically mapped cache and MMU	ARM1176
		0	Standard cache size	ARM920
		2	Reduced cache	ARM1022
		6	Tightly Coupled Memory	ARM1156
		8	As for ARM966	ARM968

Tableau II **:** Attribut des etiquettes ARM [3]

ATTRIBUTE	DESCRIPTION
D	Supports debugging via the JTAG interface. Automatic for ARMv5 and above.
E	Supports Enhanced DSP instructions. Automatic for ARMv6 and above.
F	Supports hardware floating point via the VFP coprocessor.
I	Supports hardware breakpoints and watchpoints. Automatic for ARMv5 and above.
J	Supports the Jazelle Java acceleration technology.
M	Supports long multiply instructions. Automatic for ARMv5 and above.
T	Supports Thumb instruction set. Automatic for ARMv5 and above.
-S	This processor uses a synthesizable hardware design.

ii. Les ARM *Cortex cores*

Depuis 2004, tous les cœurs ARM ont été vendus sous la marque de Cortex et ont des noms sous la forme Cortex-{x} {y}. La convention d'appellation est différente, et plus facile à suivre. Il y a trois profils de Cortex : Cortex-A, Cortex-R et Cortex-M [3]. Ces familles de

processeur sont les plus populaires des processeurs ARM aujourd'hui, c'est pour cela que ces familles seront plus détaillées dans la partie c) La famille des ARM cortex.
Il existe un ARM qui ne suit pas la convention d'appellation : les *SecurCore* qui sont des ARM spéciaux.

b) La famille ARM Cortex

i. Les ARM Cortex-A

L'ARM Cortex-A est la famille de la série des processeurs d'application. Ils sont reliés à une grande quantité de mémoire et fonctionnent à une vitesse d'horloge relativement élevée. Ils sont conçus comme des ordinateurs fonctionnels, capables d'exécuter directement les systèmes d'exploitation complexes [3]. Tous les processeurs Cortex-A offrent des performances 32 bits exceptionnelles pour l'informatique haut de gamme. Il en est de même pour les nouveaux processeurs Cortex-A72, Cortex-A57 et Cortex-A53 qui offre des performances combinées de 32 bits et 64 bits. Les processeurs Cortex-A sont disponibles dans les variétés *single-core* et *multi-core,* offrant jusqu'à quatre unités de traitement avec la possibilité d'intégrer des blocs de traitement multimédia NEON et des unités d'exécution avancée de virgule flottante [5].

Il est très utilisé dans les *Smartphones*, les *Netbooks*, les produits de réseaux et de serveurs comme illustrés sur la fig 1.1. Il est aussi présent dans les télévisions et écrans numériques et les tablettes électroniques …etc.

Figure 1.1 : Exemple d'applications de processeur ARM Cortex-A [6]

ii. Les ARM Cortex-R

L'ARM Cortex-R est la famille des processeurs à réaction rapide : la série des processeurs en temps réel. Ils sont souvent moins puissants que le Cortex-A, mais sont beaucoup plus réactif à des stimuli externes. Ils s'adaptent mieux à des situations exigeantes, ayant une faible latence d'interruption et une réponse en temps réel plus déterministe, et sont souvent utilisés dans les systèmes critiques où l'interprétation des données est essentielle [3]. Ces processeurs ont été développés pour les applications profondément embarquées en temps réel où le besoin de faible puissance et un bon comportement d'interruption sont équilibrés avec des performances exceptionnelles et une forte compatibilité avec les plates-formes existantes [5].

Il se trouve souvent, comme illustrés sur la fig 1.2, dans les cameras et les appareils photos numériques, les contrôleurs de bande de base pour les communications mobiles et les contrôleurs de périphériques de stockage de masse de bas niveau, tels que les contrôleurs de disque dur. Il est aussi présent dans les dispositifs médicaux, les systèmes de freinage des voitures, les systèmes aéronautiques (systèmes de propulsion), l'air bag…etc.

Figure 1.2 : Exemple d'applications de processeur ARM Cortex-R [6]

iii. Les ARM Cortex-M

L'ARM Cortex-M est la famille des processeurs à alimentation ultra faible : la série des processeurs pour microcontrôleur. Il fonctionne à une vitesse d'horloge plus lente généralement inférieure à la série A ou R (il peut bien fonctionner jusqu'à plus de 100 MHz) mais il nécessite moins d'énergie pour fonctionner. Il a beaucoup moins de mémoire, mais il est plus petit et coûte moins cher. Il est généralement intégré dans les microcontrôleurs avec plusieurs lignes d'entrées et de sorties et est conçu pour les systèmes de petits facteurs qui s'appuient sur des entrées et des sorties numériques lourdes [3].

Il est souvent utilisé pour contrôler les périphériques matériels ou pour être une interface entre le matériel et un autre processeur (la plupart des Clés USB et *Bluetooth* ont un processeur Cortex-M à l'intérieur). Il est aussi utilisé en tant que processeurs de support dans des dispositifs plus grands. Il est présent, comme sur la fig 1.3, dans les systèmes robotiques (commande des actionneurs comme les moteurs) et les appareils électroniques de consommation, dans les compteurs intelligents, les dispositifs d'interface humaine, les systèmes de contrôle automobiles et industriels, les appareils ménagers domestiques, l'instrumentation médicale ou industrielle...etc.

Figure 1.3 : Exemple d'applications de processeur ARM Cortex-M [6]

c) Comparaison entre les ARM Cortex 32 Bits

La figure 1.4 résume l'évolution et de la comparaison entre les ARM Cortex 32 bits selon la performance et la capacité.

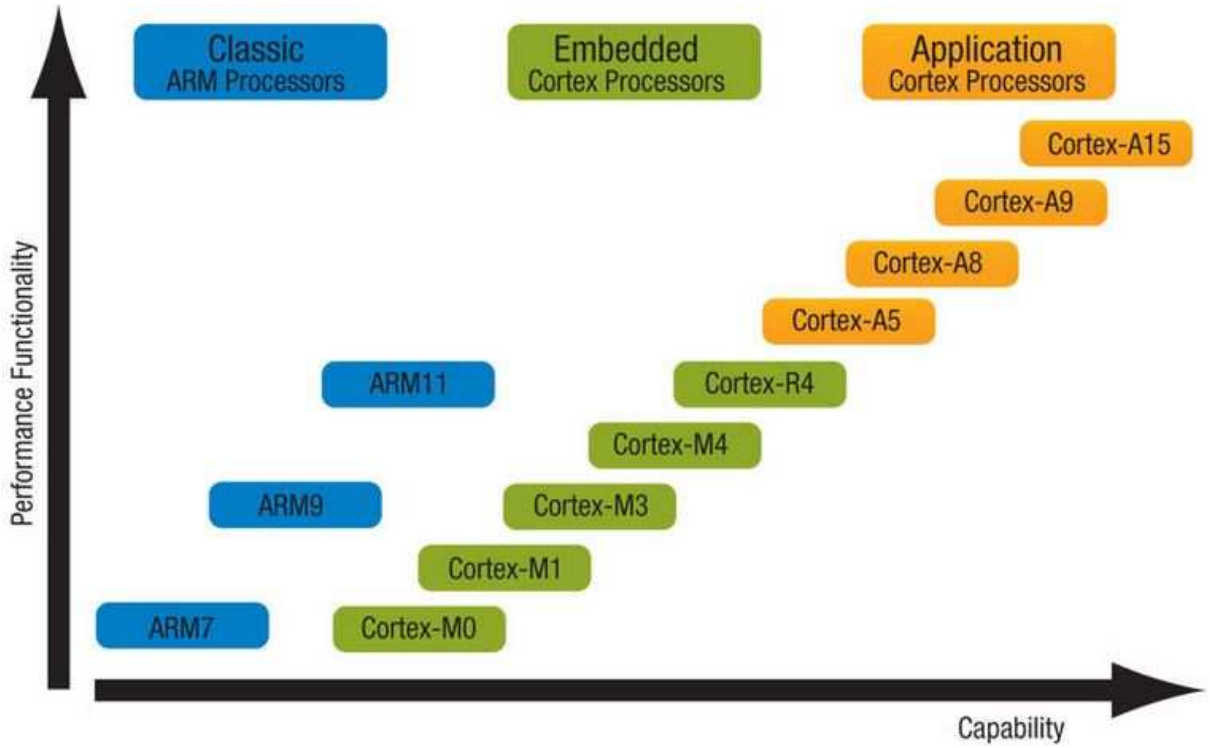


Figure 1.4 : Comparaison entre les ARM Cortex sur la capacité et la performance [7]

Comme on peut le constater les Cortex-A sont les plus performants et ont le plus de fonctions puisque ce sont des processors d'application. Ils tiennent la première place, suivi des Cortex-R. Les Cortex-M sont plus ou moins égaux aux processeurs classiques du point de vue performance mais ils ont plus de fonctions et consomment moins que ces derniers.

Aujourd'hui, avec l'évolution de la technologie, on peut maintenant rencontrer des processeurs de 64 bits pour les processeurs d'application. Ils peuvent être *multi-core*, jusqu'à quatre cœurs (*quadcore*) et pouvant travailler jusqu'à la fréquence de 2.5 GHz. C'est le cas des ARM Cortex-A72/A57/A53. C'est pour cela qu'aujourd'hui les *Smartphones*, les tablettes et autres sont aussi performants que les ordinateurs personnels.

Pour notre travail nous allons nous focaliser sur les processeurs ARM cortex- M pour les microcontrôleurs, étant donné que c'est notre domaine d'étude. Mais on peut trouver dans l'annexe 1 un tableau de l'évolution des processeurs ARM depuis 1994.

I.4. LES MICROPROCESSEURS ARM CORTEX-M

Il existe en réalité 6 types d'ARM Cortex- M jusqu'à maintenant:

- Les ARM Cortex-M3 qui sont basés sur l'architecture ARMv7-M
- Les ARM Cortex-M0 qui sont basés sur l'architecture ARMv6-M
- Les ARM Cortex-M0+ qui sont basés sur l'architecture ARMv6-M
- Les ARM Cortex-M1 qui sont basés sur l'architecture ARMv6-M
- Les ARM Cortex-M4/M4F qui sont basés sur l'architecture ARMv7E-M
- Les ARM Cortex-M7 qui sont basés sur l'architecture ARMv7E-M
-

Les ARM Cortex-M0/M0+/M3/M4/M7 sont les processeurs des microcontrôleurs. Il faut noter que les ARM Cortex-M1 font partie des processeurs ARM spéciaux qui sont adaptés pour les FPGA. Étant donné que notre domaine d'étude se situe dans les ARM Cortex pour les microcontrôleurs, les ARM Cortex- M1 seront omis.

a) Les architectures ARMv6-M et ARMv7-M

Tous les ARM Cortex-M sont en général à base d'architecture ARMv6-M et ARMv7-M. *ARM Ltd* développe de nouveaux processeurs, de nouvelles instructions. Des caractéristiques architecturales sont ajoutées de temps à autre comme le montre la fig 1.5. En conséquence, il existe des versions différentes de l'architecture. Par exemple, l'ARM7TDMI est basé sur la version de l'architecture ARMV4T (Le « T » signifie *« Thumb instruction support »*). Il est à noter que les numéros de version sont indépendants des noms de l'architecture de processeur.

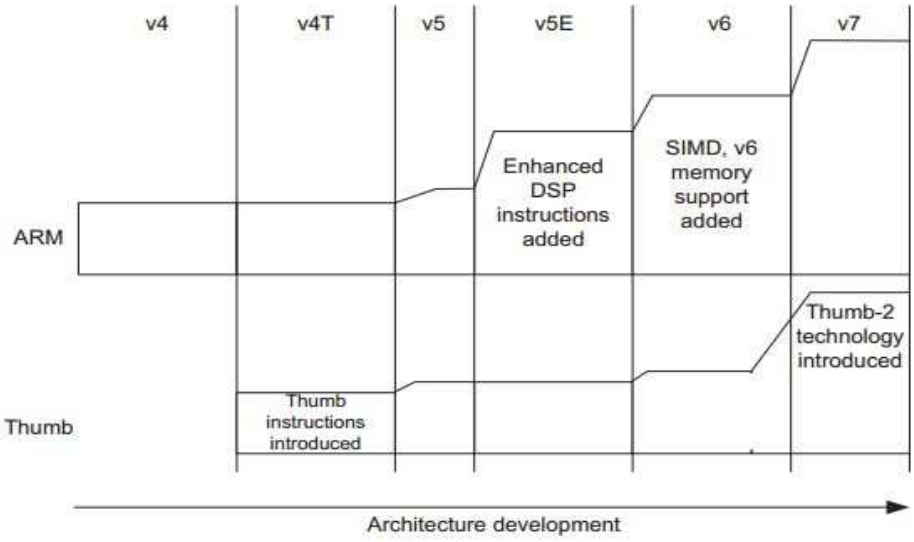


Figure 1.5 : Evolution des instructions sur le processeur [8]

L'architecture ARMv5TE a été introduite avec les familles de processeurs d'ARM9E, y compris les processeurs ARM926E-S et ARM946E-S. A cette architecture s'ajoutait des instructions de Traitement numérique du signal (DSP) améliorées pour les applications multimédias. Avec l'arrivée de la famille de processeur ARM11, l'architecture a été étendue à ARMv6. Les nouvelles fonctionnalités de cette architecture incluent les fonctionnalités du système de mémoire et les instructions *Single Instruction Multiples Data* (SIMD). Les processeurs classiques basés sur l'architecture ARMv6 sont l'ARM1136J(F)-S, l'ARM1156T2(F)-S et l'ARM1176JZ (F)-S.

Après le succès du processeur Cortex- M3, un profil d'architecture supplémentaire appelé ARMv6-M a également été créé, pour répondre aux besoins d'ultra-faible consommation d'énergies à faible coût. Il utilise le même modèle de programmation, les mêmes méthodes de gestion d'interruption et quelques instructions de la technologie *Thumb-2* de l'ARMv7-M. Il utilise principalement des instructions *Thumb* d'ARMv6 pour réduire la complexité de la conception ; on peut le constater sur la fig 1.6. L'architecture ARMv6-M est aussi similaire à ARMv7-M sur l'architecture de débogage. Cependant, ARMv6-M dispose d'un ensemble d'instructions très limitées [9].

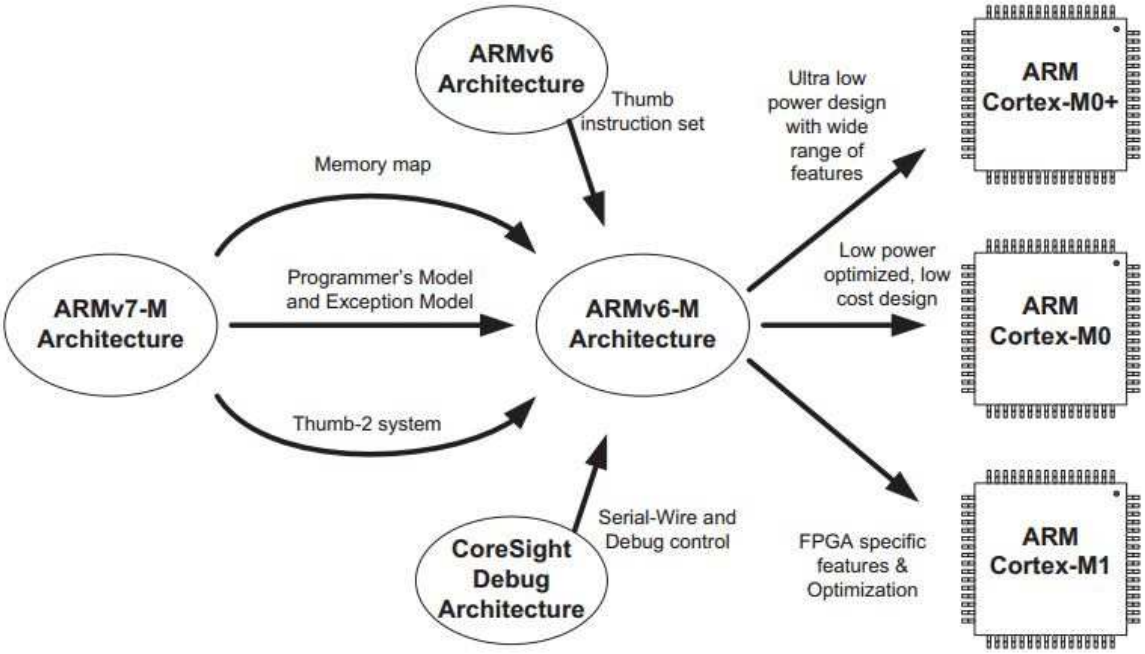


Figure 1.6 : Héritages de l'architecture ARMv6-M sur les autres architectures [9]

Les processeurs Cortex- M4 et M7 sont souvent référencés comme ARMv7E-M. Le « E » comme « *Enhanced DSP Instructions* » dans ARMv7E-M désigne des améliorations concernant l'unité de DSP pour les applications multimédias; mais c'est toujours un ARMv7-M. Et le « F » de

Cortex-M4F fait référence à la présence d'un FPU (*Floating Point Unit*), une unité de traitement des virgules flottantes.

b) ARM Cortex-M3

Aujourd'hui, le Cortex-M3 est le plus largement utilisé de tous les processeurs Cortex-M. C'est en partie parce qu'il est disponible non seulement pour la plus longue période de temps, mais il répond aussi aux exigences d'un microcontrôleur à usage général. Cela signifie généralement qu'il a un bon équilibre entre haute performance, faible consommation d'énergie, et faible coût. Il a été le premier Cortex-M disponible [6].

La figure 1.7 montre les principales composantes du Cortex-M3. Le cœur du Cortex-M3 est un CPU 32 bits. C'est un processeur à jeu d'instructions réduites (RISC) où la plupart des instructions s'exécutent en un seul cycle.

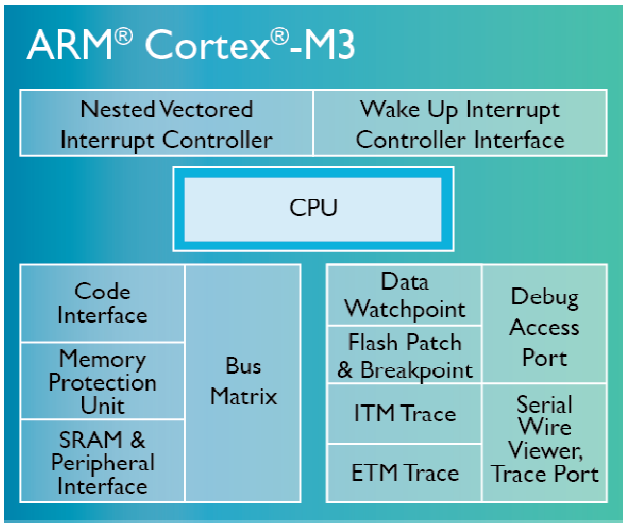


Figure 1.7 : Principales composantes interne du Cortex-M3 [10]

Le CPU Cortex-M3 peut exécuter la plupart des instructions en un seul cycle. Cela est possible parce qu'il dispose d'un pipeline de trois étages avec prédiction de branchement et avec chacune des unités « *Fetch* », « *Decode* », « *Execute* » séparées comme sur la fig 1.8.

Figure 1.8 : Pipeline de trois étages [6]

Ainsi, pendant qu'une instruction est en cours d'exécution, un deuxième est en cours de décodage, et une troisième est en cours de récupération comme illustrée sur la fig 1.9. La même approche a été utilisée sur l'ARM7.

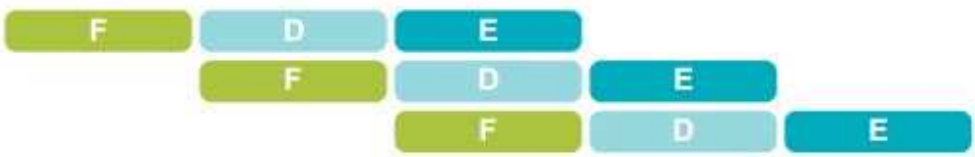


Figure 1.9 : Exécution en parallèle du pipeline [6]

Cependant, lorsque le programme exécute un branchement conditionnel, le pipeline doit attendre d'être vidé et rempli avec de nouvelles instructions avant qu'il puisse continuer. Un tel système fait du branchement sur l'ARM7 très coûteux en termes de puissance de traitement. Par contre, le Cortex-M3 comprend une instruction pour aller chercher l'unité « *branch speculation fetch* » qui gère la prédiction des branchements, ce qui réduit les pénalités en vitesse. Ce principe aide le Cortex-M3 à avoir une puissance de traitement plus favorable soit 1,25 DMIPS / MHz (*Dhrystone Millions Instructions Per Seconde*).

Le processeur Cortex- M3 comprend également une unité d'interruption de vecteur imbriqué qui peut desservir jusqu'à 240 sources d'interruptions. Le NVIC (*Nested Vectored Interrupt Controller*) fournit un accès rapide à la gestion d'interruption déterministe et la routine de service d'interruption prend seulement 12 cycles à chaque fois. Le NVIC contient également une minuterie standard appelée « *sysTick timer* ». C'est un compte à rebours de 24 bits avec rechargement automatique. La « *sysTick timer* » est présente sur tous les différents processeurs Cortex-M. Elle est utilisée pour fournir des interruptions périodiques régulières. Une utilisation typique de ce « *timer* » est de fournir une graduation de minuterie pour les systèmes d'exploitation en temps réel à faible encombrement (RTOS : *Real Time Operating System*). Aussi, à côté du NVIC, il y a le contrôleur de réveil d'interruption (WIC : *Wake up Interrupt Controller*); c'est une petite zone du processeur Cortex-M qui est maintenue en vie lorsque le processeur est en mode faible consommation d'énergie. Le WIC peut utiliser les signaux d'interruption des périphériques de microcontrôleur pour réveiller le processeur Cortex-M du mode faible puissance. Le WIC peut être mis en œuvre de diverses manières et dans certains cas, il ne nécessite pas d'horloge pour fonctionner. Il peut également être dans une région d'alimentation séparée du processeur principal. De cette façon, 99 % du processeur Cortex-M peuvent être placés dans un mode faible puissance avec un minimum de courant utilisé par le WIC [6].

c) ARM Cortex-M0

Le Cortex-M0 a été introduit quelques années après la Cortex-M3, c'est un processeur à usage général. Le Cortex-M0 est un processeur beaucoup plus petit que le Cortex-M3 ; le nombre de portes peut être aussi petit que 12 K dans la configuration minimum. Le Cortex-M0 est généralement conçu pour les microcontrôleurs qui sont destinés à être des dispositifs de très faible coût et /ou destinés à des opérations de faible puissance. Cependant, la chose importante est qu'une fois qu'on comprend le Cortex-M3, on n'aura aucun problème avec le Cortex-M0. Les différences sont essentiellement transparentes pour les langages de haut niveau même si leurs composantes internes, comme le montre la fig 1.10, sont plus ou moins différentes.

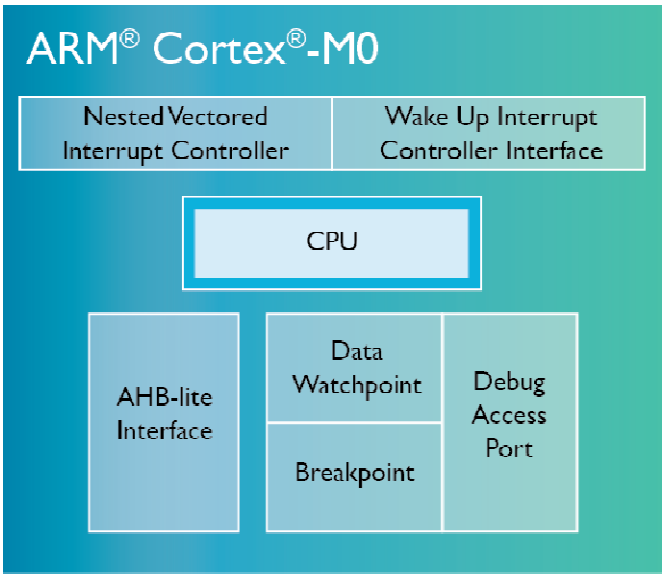


Figure 1.10 : Principales composantes interne du Cortex-M0 [11]

Le processeur Cortex-M0 possède un cœur qui peut exécuter un sous-ensemble du jeu d'instructions *Thumb-2*. Comme le Cortex-M3, il dispose d'un pipeline en trois étapes, mais aucune unité « *branch speculation fetch* » donc pas de prédiction de branchement. Le Cortex-M0 possède également une architecture de bus Von Neumann, de sorte qu'il n'existe qu'un chemin unique pour le code et les données. Même si cela rend la conception simple, les performances peuvent être réduites. Par rapport à la Cortex-M3, le Cortex-M0 possède une performance de 0,84 DMIPS / MHz, moindre que celle du Cortex- M3 mais il rivalise avec un ARM7 qui a trois fois plus de nombre de portes. Le processeur Cortex-M0 a la même NVIC que le Cortex-M3, mais elle est limitée à un maximum de 32 lignes d'interruption provenant des périphériques du microcontrôleur. Le NVIC contient également le « *sysTick timer* » qui est

entièrement compatible avec le Cortex-M3. La plupart des RTOS qui s'exécutent sur le Cortex-M3 et Cortex-M4 peuvent également fonctionner sur le Cortex-M0, mais le vendeur devra faire un port dédié et recompiler le code RTOS. En tant que développeur, la plus grande différence qu'on peut trouver entre l'utilisation du Cortex-M0 et le Cortex-M3 est leur capacité de débogage. Sur le Cortex-M3, il y a un vaste support de débogage en temps réel, alors que le Cortex-M0 possède un support de débogage plus modeste. Sur le Cortex-M0, l'unité de DWT (*Data Watchpoint*) ne supporte pas le traçage de données et il n'est pas équipé de l'ITM (*Instrumentation Trace Macrocell*), donc on se retrouve avec un contrôle de base de l'exécution (c'est-à-dire, « *run* », « *halt* », « *single stepping/breakpoint* » et «*watchpoint* ») et un accès périphériques/mémoire à la volée [6].

Le Cortex-M0 est conçu pour supporter des modes de veille à faible puissance. Comparé à un MCU (*Microcontroller Unit*) 8 ou 16 bits, comme le montre la fig 1.11, il peut rester en mode veille beaucoup plus longtemps parce qu'il exécute moins d'instructions qu'un dispositif 8/16 bits.

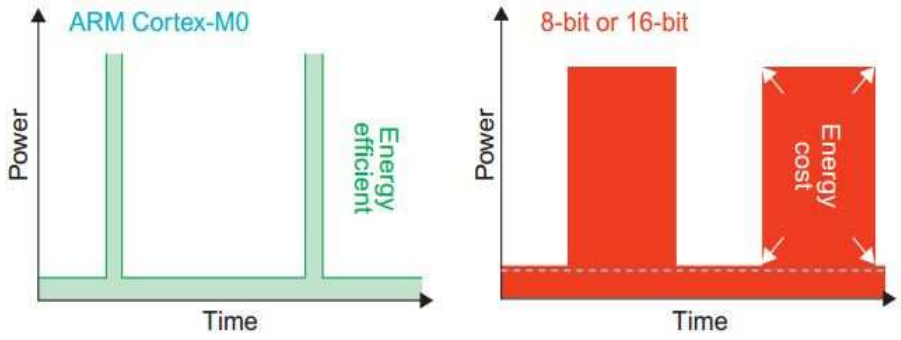


Figure 1.11: Comparaison en consommation entre le Cortex-M0 et un dispositif 8/16 bits [6]

La consommation typique du Cortex-M0 est 16Mw / MHz lors de l'exécution d'instructions et presque nulle lors du mode veille à faible consommation. D'autres architectures 8 et 16 bits peuvent également atteindre des chiffres de faible puissance similaires, mais ils ont besoin d'exécuter beaucoup plus d'instructions pour obtenir le même résultat. Cela signifie cycles supplémentaires et des cycles supplémentaires signifient plus de consommation d'énergie. Si nous prenons comme exemple la multiplication 16×16, le Cortex-M0 peut effectuer ce calcul en un cycle. En comparaison, une architecture 8 bits typique comme le 8051 aura besoin d'au moins 48 cycles et une architecture 16 bits aura besoin de 8 cycles comme le montre le tab III. Ce n'est pas seulement un avantage de performance, mais aussi un avantage d'efficacité énergétique. Et tout comme le Cortex-M3, le Cortex-M0 a également la fonction WIC.

Tableau III: Comparaison entre les processeurs 8, 16 bit et Cortex-M sur le cycle d'exécution [6]

8-Bit Example (8051)	16-Bit Example	ARM Cortex-M
Time: 48 clock cycles* **Code size:** 48 bytes	**Time:** 8 clock cycles **Code size:** 8 bytes	**Time:** 1 clock cycle **Code size:** 2 bytes

d) ARM Cortex-M0+

Le processeur Cortex-M0+ est la dernière génération Cortex-M de très faible puissance. Il a un jeu d'instructions complet compatible avec le Cortex-M0 permettant d'utiliser le même compilateur et les mêmes outils de débogage. Comme on peut s'y attendre, le Cortex- M0+ est une version améliorée de Cortex- M0 comme le montre la fig 1.12.

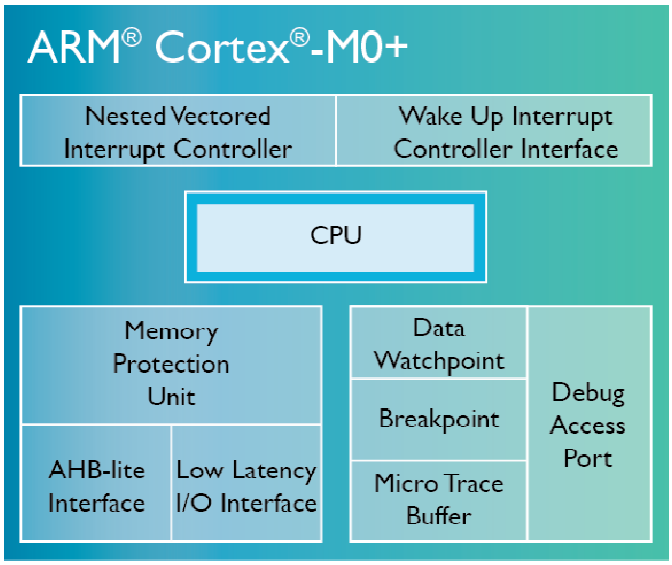


Figure 1.12 : Principales composantes interne du Cortex-M0+ [12]

La caractéristique du Cortex-M0 + est sa consommation d'énergie, qui est juste 11µW/MHz par rapport au 16µW / MHz du Cortex-M0 et au 32µW / MHz du Cortex-M3. Un des modifications architecturales clés du Cortex-M0+ est le changement à un pipeline en deux étages, illustrés sur la fig 1.13. Lorsque le Cortex-M0 et Cortex-M0+ exécute une branche conditionnelle, les instructions dans le pipeline ne sont plus valables. Cela signifie que le pipeline doit être vidé à chaque fois qu'il y a une branche. Une fois que la branche a été prise en main, le pipeline doit être rempli pour reprendre l'exécution. Cela a un impact sur la performance car plusieurs accès à la mémoire flash sont engendrés et chaque accès coûte en énergie ainsi qu'en temps. En passant à un pipeline en deux étages, le nombre d'accès à la mémoire flash diminue et donc la consommation d'énergie d'exécution est également réduite.

Figure 1.13 : Pipeline de deux étages [6]

Une autre caractéristique importante ajoutée à la Cortex-M0+ est une nouvelle interface périphérique E / S (Entrée/Sortie) qui prend en charge l'accès à cycle unique aux registres périphériques. Le cycle unique de l'interface E / S est un élément standard de la mémoire mappée « memory map » du Cortex-M0+ et n'utilise pas d'instructions spéciales ou d'adressage paginé. Les registres situés dans l'interface E / S peuvent être accessibles par des pointeurs du langage C à partir du code d'application. L'interface E / S permet un accès plus rapide aux registres périphériques avec moins de consommation d'énergie. Comme sur la figure 1.14, l'interface E / S à cycle unique est séparé du bus à vitesse élevée (AHB : *Advanced High-Performance Bus*), de sorte qu'il est possible pour le processeur d'aller chercher des instructions par l'interface AHB tout en faisant un accès aux données dans les registres périphériques situés dans l'interface E / S.

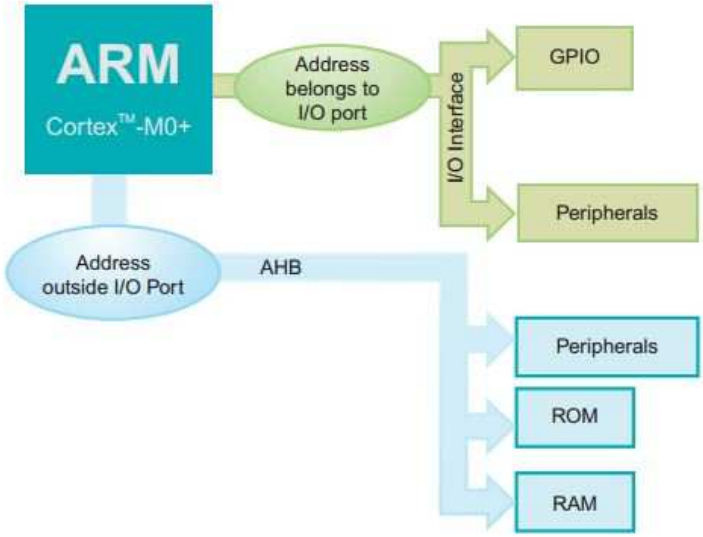


Figure 1.14 : Schéma de la séparation du bus à vitesse élevée et de l'interface E/S [6]

Le Cortex-M0+ dispose également d'une architecture de débogage amélioré comparé au Cortex-M0. Il prend en charge le même accès en temps réel aux registres et aux périphériques SRAM (*Static Random Access Memory*) que le Cortex-M3 et Cortex-M4. En outre, le Cortex-M0+ a une nouvelle fonctionnalité de débogage appelée MTB (*Micro Trace Buffer*). Le MTB permet d'enregistrer des instructions de programme exécuté dans la région du SRAM. Lorsque le code

est arrêté, cette instruction de traçage peut être téléchargée et affichée dans le débogueur. Ceci permet d'obtenir immédiatement un aperçu de l'exécution de code avant que le code ait été interrompu. Bien que le MTB n'ait qu'un nombre limité de mémoire tampon de traçage, il est extrêmement utile pour traquer les bugs insaisissables. Il peut être accessible par un matériel de débogage standard JTAG/SWD (*Joint Test Action Group/ Serial Wire Debugger*) ; on n'a pas besoin d'outil de traçage cher [6].

e) ARM Cortex-M4

Alors que le Cortex-M0 peut être considéré comme un Cortex-M3 moins certaines caractéristiques, le Cortex-M4 est une version améliorée du Cortex-M3. Les fonctionnalités supplémentaires sur le Cortex-M4 sont axées sur le support d'algorithme DSP : les algorithmes typiques telles que la transformée de Fourier rapide (FFT: *Fast Fourier Transform*), les filtres numériques à réponse impulsionnelle finie (FIR: *Finite Impulse Response*), et les algorithmes de contrôle tels que « proportionnelle intégral dérivée » (PID) en boucle de régulation.

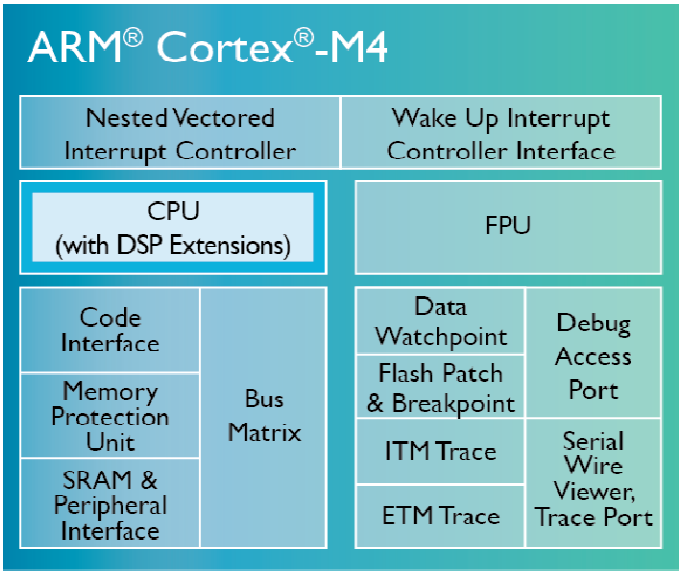


Figure 1.15 : Principales composantes interne du Cortex-M4 [13]

Avec ses fonctions de DSP, le Cortex-M4 a créé une nouvelle génération de dispositifs à base d'ARM, qui fonctionne comme des contrôleurs de signal numérique (DSC : *Digital Signal Controller*). Ainsi, le Cortex-M4 permet de concevoir des dispositifs qui combinent des fonctions de type microcontrôleur avec des traitements de signal en temps réel.
Le Cortex-M4 a la même structure de base que le Cortex-M3 (modes de programmation CPU, NVIC, architecture de débogage *CoreSight*, MPU, interface de bus) comme le montre la fig 1.15.

Les améliorations sur le Cortex-M3 sont en partie sur l'ensemble des instructions où le Cortex-M4 a des instructions DSP supplémentaires sous la forme d'instructions SIMD [6].

L'unité de calcul d'accumulation et de multiplication (MAC : *Multiply-Accumulate*) a également été améliorée de sorte que plusieurs 32×32 instructions arithmétiques soient exécutées en un seul cycle comme listées sur le tableau IV.

Tableau IV: 32×32 instructions exécutées en un seul cycle [6]

Operation	Instructions
16 x 16 = 32	SMULBB, SMULBT, SMULTB, SMULTT
16 x 16 + 32 = 32	SMLABB, SMLABT, SMLATB, SMLATT
16 x 16 + 64 = 64	SMLALBB, SMLALBT, SMLALTB, SMLALTT
16 x 32 = 32	SMULWB, SMULWT
(16 x 32) + 32 = 32	SMLAWB, SMLAWT
(16 x 16) ± (16 x 16) = 32	SMUAD, SMUADX, SMUSD, SMUSDX
(16 x 16) ± (16 x 16) + 32 = 32	SMLAD, SMLADX, SMLSD, SMLSDX
(16 x 16) ± (16 x 16) + 64 = 64	SMLALD, SMLALDX, SMLSLD, SMLSLDX
32 x 32 = 32	MUL
32 ± (32 x 32) = 32	MLA, MLS
32 x 32 = 64	SMULL, UMULL
(32 x 32) + 64 = 64	SMLAL, UMLAL
(32 x 32) + 32 + 32 = 64	UMAAL
32 ± (32 x 32) = 32 (upper)	SMMLA, SMMLAR, SMMLS, SMMLSR
(32 x 32) = 32 (upper)	SMMUL, SMMULR

f) ARM Cortex-M7

Le Cortex-M7 prend la suite logique des très populaires Cortex-M3 et M4, il se positionne comme un complément aux produits existants, représentant un nouveau plafond à très haute performance pour cette gamme de produits. Avec une finesse de gravure équivalente de 90 nm, le M7 est capable d'atteindre un score de 1 000 *CoreMark* (le benchmark de référence pour les systèmes embarqués) en étant cadencé à 200 Mhz, là où le M4 n'atteint que 608 (à une fréquence inférieure, le M4 ne pouvant dépasser 180 MHz). Pour obtenir ces performances, le Cortex-M7 possède un pipeline à six étages. Le M7 gère également le SIMD, les instructions MAC sur un seul cycle, la prédiction de branchement, le calcul de virgule flottante à "double précision" (*binary64*), et possède des performances de traitement numérique du signal (DSP) optimisées par rapport à celles introduites sur le M4.

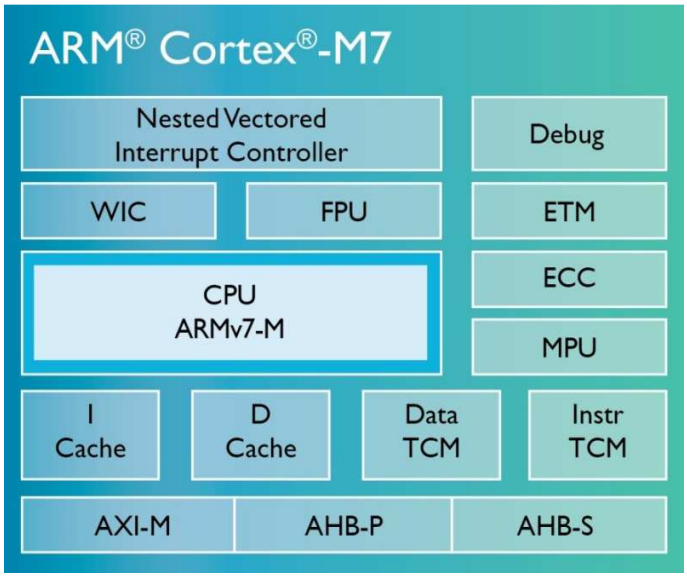


Figure 1.16 : Principales composantes interne du Cortex-M7 [14]

En plus des traditionnels bus AHB, il intègre un bus AXI 64 bits comme le montre la fig 1.16 (qui lui permet de charger deux instructions en même temps) avec une mémoire cache optionnelle (jusqu'à 64 Ko) pour les instructions et les données. De plus, le M7 intègre de la mémoire directement sur le processeur, et les interfaces des mémoires internes et externes sont optimisées pour communiquer entre elles en temps réel. Le résultat de ce travail est une performance impressionnante de 2,14 DMIPS/MHz, une première pour un processeur destiné aux microcontrôleurs. De plus, en réduisant la taille de la puce (passage à une finesse de gravure de 40 nm puis 28 nm) et en augmentant sa fréquence (400 puis 800 MHz), *ARM Ltd* compte éventuellement doubler puis quadrupler ses performances d'ici quelques années (un représentant a estimé l'arrivée des M7 gravés en 28 nm d'ici à trois ans) [15].

CHAPITRE II : LA TECHNOLOGIE ARM CORTEX-M3/M4

II.1. VUE INTERNE DE L'ARM CORTEX-M3

Le processeur Cortex-M3 contient non seulement le cœur du processeur, mais aussi un certain nombre de composants pour la gestion du système, ainsi que de composants de débogage. Ces composants sont reliés entre eux en utilisant un bus à haute performance (AHB), et un bus périphérique avancé (APB : *Advanced Peripheral Bus*). Les AHB et APB font partie des normes *Advanced Microcontroller Bus Architecture* (AMBA). Les blocs MPU, WIC et ETM (*Embedded Trace Macrocell*) sont des blocs facultatifs qui peuvent être inclus dans le système à microcontrôleur lors de la mise en œuvre. La figure 2.1 montre que le processeur Cortex-M3 est constitué de sous-système de processeur.

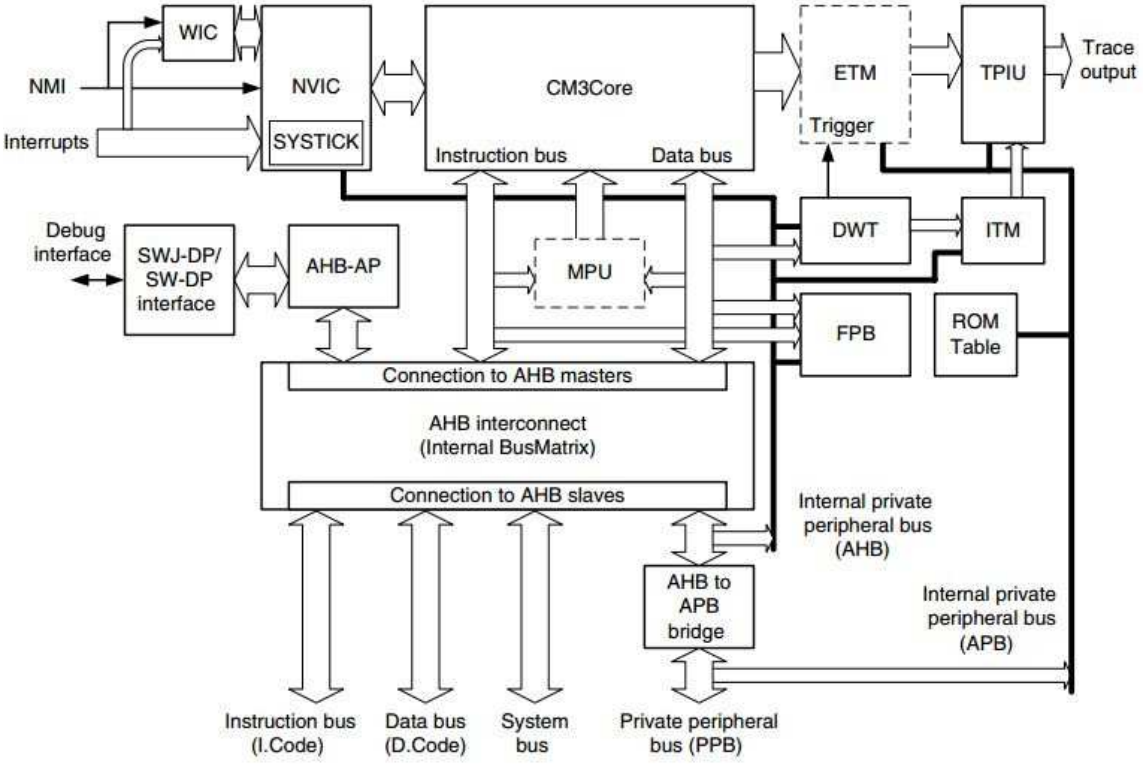


Figure 2.1 : Principaux sous systèmes du Cortex-M3 [8]

Le noyau CPU lui-même est étroitement couplé au contrôleur d'interruption et divers blocs logiques de débogage :

- CM3Core : le noyau Cortex-M3 contient les registres, l'ALU (*Arithmetic Logic Unit*), le chemin de données, et une interface de bus.

- NVIC : le NVIC est un contrôleur d'interruption intégré. Le nombre d'interruptions est personnalisé par les fabricants de la puce. Le NVIC est étroitement couplé à la base avec le CPU et contient un certain nombre de système de registres de commande. Il prend en charge la gestion des interruptions imbriquées, ce qui facilite la gestion des interruptions dans le Cortex-M3. Il est également livré avec une fonction d'interruption vectorisée de sorte que lorsqu'une interruption se produit, il peut entrer directement dans la routine de gestionnaire d'interruption correspondant.
- *SysTick Timer*: le système *Tick (sysTick) Timer* est un compte à rebours de base qui peut être utilisé pour générer des interruptions à intervalles de temps réguliers, même lorsque le système est en mode veille. Il rend la portabilité et la réutilisabilité d'un OS (*Operating System*) entre les appareils Cortex-M3 beaucoup plus faciles parce qu'il n'est pas nécessaire de modifier le code de la minuterie. La minuterie *sysTick* est mise en œuvre dans le NVIC.
- WIC : le WIC est un module interface avec le NVIC mais séparé du processeur principal pour permettre le réveil du système venant des événements d'interruption quand le processeur (y compris le NVIC) est complètement arrêté ou mis hors tension.
- MPU : le bloc de MPU est facultatif. S'il est inclus, le MPU peut être utilisé pour protéger le contenu de la mémoire, par exemple, pour rendre les régions de la mémoire en lecture seule ou pour empêcher des applications de l'utilisateur d'accéder aux données d'applications privilégiées.
- *BusMatrix* : il est utilisé comme le cœur du système de bus interne du Cortex-M3. C'est un réseau d'interconnexion d'AHB, permettant le transfert sur différents bus simultanément dans la mesure où les deux bus (maîtres et esclaves) ne cherchent pas à accéder à la même région de la mémoire. Le *BusMatrix* assure également la gestion de transfert de données supplémentaires, y compris un tampon d'écriture ainsi que les opérations orientées bits.
- AHB to APB : c'est un pont de bus qui est utilisé pour connecter un certain nombre de dispositifs APB tels que les composants de débogage au bus périphérique privé dans le processeur Cortex-M3. En outre, le Cortex-M3 permet aux fabricants de puces de fixer des dispositifs APB supplémentaires au bus périphérique privé externe (PPB : *Private Peripheral Bus*) en utilisant ce bus APB.

Le reste des composants sont pour le soutien de débogage et normalement ne devraient pas être utilisés par le code de l'application :

- SW-DP/SWJ-DP: le *Serial Wire Debug Port* (SW-DP)/*Serial Wire JTAG Debug Port* (SWJ-DP) collabore avec le port d'accès AHB (AHB –AP : AHB – *Access Port*) de sorte que les débogueurs externes puissent générer des transferts AHB pour contrôler les activités de débogage. Il n'y a pas de chaîne de balayage de port JTAG à l'intérieur du cœur du

processeur Cortex-M3; la plupart des fonctions de débogage sont contrôlées par les registres du NVIC à travers les accès AHB. Le SWJ-DP prend en charge le *Serial Wire Protocol* ainsi que le JTAG *Protocol*, alors que le SW-DP ne peut supporter que les *Serial Wire Protocol*.

- AHB-AP : l'AHB-AP permet d'accéder à toute la mémoire du Cortex-M3 à partir de quelques registres. Ce bloc est contrôlé par le SW-DP/SWJ-DP via une interface de débogage générique appelé *Debug Access Port* (DAP). Pour mener à bien les fonctions de débogage, le matériel de diagnostic externe doit accéder à la AHB - AP à travers le SW- DP / SWJ -DP pour générer les transferts AHB requis.
- ETM : l'ETM est un composant optionnel pour l'instruction de traçage, certains produits Cortex-M3 pourraient ne pas avoir la capacité de suivi d'instructions en temps réel. Les informations de traçage sortent par le port de traçage à travers TPIU (*Trace Port Interface Unit*). Les registres de contrôle ETM sont mappés en mémoire et peuvent être contrôlés par le débogueur à partir de la DAP.
- DWT : le DWT permet de mettre en place les «*data watchpoints* » pour le débogage. Lorsqu'une adresse de données ou une valeur de données est trouvée, le DWT peut être utilisé pour générer des événements de point d'observation qui active le débogueur et génère des informations de traçage de données, ou active l'ETM.
- ITM: l'ITM peut être utilisé de plusieurs façons. Le logiciel peut écrire directement dans ce module pour sortir l'information vers le TPIU, ou encore des événements DWT peuvent être utilisés pour générer des paquets de traçage de données par le biais ITM pour la sortie dans un flux de données de traçage.
- TPIU : le TPIU est utilisé pour interfacer le matériel de traçage externe comme les analyseurs de traçage de port. Les informations de traçage sont formatées comme des paquets d'ATB (*Advanced Trace Bus*) et TPIU reformate ces données pour les permettre d'être capturées par des dispositifs externes.
- FPB: le FPB (*Flash Patch and Breakpoint unit*) est utilisé pour fournir des *Flash Patch* et les fonctionnalités du point d'arrêt. Le *Flash Patch* signifie que si un accès à une instruction par la CPU correspond à une certaine adresse, l'adresse peut être reconfigurée à un autre endroit de tel sorte qu'une autre valeur soit récupérée. Alternativement, l'adresse identifiée peut être utilisée pour déclencher un événement de point d'arrêt. La fonction *Flash Patch* est très utile pour les tests, comme l'ajout de code de programme de diagnostic à un dispositif non utilisable dans des situations normales à moins d'utiliser le FPB pour modifier le contrôle du programme.
- ROM (*Read-Only Memory*) table : c'est tout simplement une petite table de consultation pour fournir les informations de *mapping* de mémoire des différents dispositifs du système et des composants de débogage. Les systèmes de débogage utilisent cette table pour localiser les adresses mémoires des composants de débogage. Dans la

plupart des cas, le *mapping* de mémoire doit être fixé à l'emplacement de mémoire standard. Certains des composants de débogage sont facultatifs et des composants supplémentaires peuvent être ajoutés. Les fabricants de puces individuelles peuvent vouloir personnaliser les fonctionnalités de débogage de leur puce. Dans ce cas, le ROM table doit être personnalisé et utilisé pour le débogage logiciel afin de déterminer le *mapping* de mémoire correct et donc de détecter le type de composants de débogage disponibles [8].

II.2. MODELES DU PROGRAMMEUR

a) Mode d'opération et état d'opération

Les processeurs Cortex-M3 et M4 ont deux états de fonctionnement et deux modes d'opération comme le montre la fig 2.2. En outre, les processeurs peuvent avoir des niveaux d'accès privilégiés et non privilégiés. Le niveau d'accès privilégié peut accéder à toutes les ressources du processeur, tandis que le niveau d'accès non privilégié signifie que certaines régions de mémoire sont inaccessibles, et quelques opérations ne peuvent pas être utilisées.

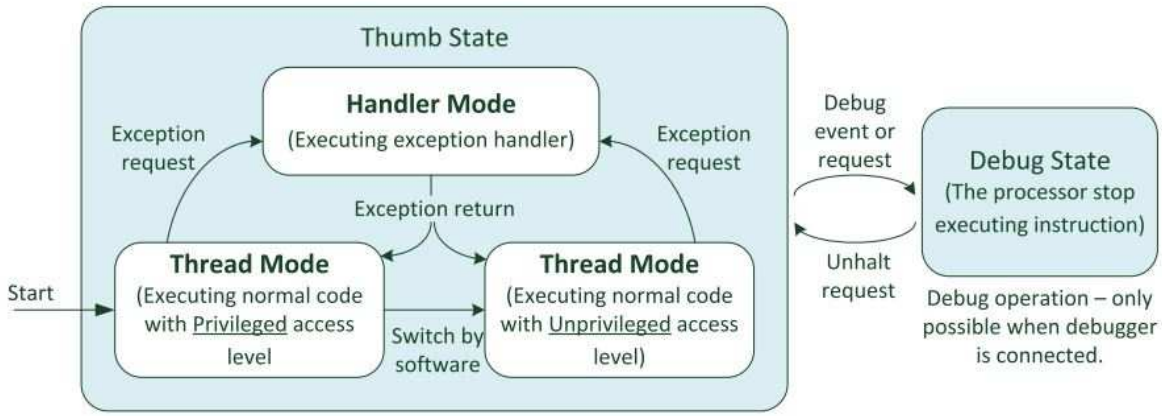


Figure 2.2 : Les modes et les états d'opérations [9]

i. Les états de fonctionnement

- *Debug state* (Etat de débogage): lorsque le processeur est interrompu (par exemple, par le débogueur, ou après avoir atteint un point d'arrêt), il passe à l'état de débogage et arrête l'exécution des instructions. Cet état est activé par une demande d'arrêt à partir du débogueur, ou par des événements de débogage générés à partir de composants de débogage dans le processeur. Cet état permet au débogueur d'accéder ou modifier les valeurs de registres du processeur. La mémoire du système, y compris les périphériques

à l'intérieur et à l'extérieur du processeur, peut être consultée par le débogueur soit dans le *Thumb state* ou le *Debug state*.

- *Thumb state* : si le processeur est en cours d'exécution de code de programme (instructions Thumb), il est dans l'état « Thumb state ». Contrairement aux processeurs ARM classiques comme ARM7TDMI, il n'y a pas de ARM state parce que les processeurs Cortex-M ne supportent pas le jeu d'instructions ARM [9].

ii. Les modes d'opération

- *Handler mode*: c'est le mode lors de l'exécution d'un gestionnaire d'exception comme l'ISR (*Interrupt Service Routine*). En mode Handler, le processeur a toujours le niveau d'accès privilégié.
- *Thread mode*: c'est le mode lors de l'exécution du code d'application normale; le processeur peut être soit dans le niveau d'accès privilégié ou le niveau d'accès non privilégié. Ceci est contrôlé par un registre spécial appelé «*CONTROL*».

Le logiciel peut passer le processeur du mode *Thread* privilégié en mode *Thread* non privilégié. Cependant, il ne peut pas revenir du mode non privilégié au mode privilégié. Si c'est nécessaire, le processeur doit utiliser le mécanisme d'exception pour gérer la commutation. La séparation des niveaux d'accès privilégiés et non privilégiés permet aux concepteurs de systèmes de développer des systèmes embarqués robustes. Ainsi, cela fournit un mécanisme pour protéger la mémoire des accès aux zones critiques et un modèle de sécurité de base. Par exemple, un système peut contenir un noyau d'OS embarqué qui s'exécute au niveau d'accès privilégié, et des tâches d'application qui s'exécutent au niveau d'accès non privilégié. De cette façon, on peut mettre en place des autorisations d'accès mémoire en utilisant l'unité de protection de mémoire (MPU). Cette unité va empêcher une tâche d'application de corrompre la mémoire et les périphériques utilisés par le noyau d'OS et d'autres tâches. Si une tâche d'application se bloque, les tâches d'application restantes et le noyau de l'OS peuvent continuer à s'exécuter.

Le *Thread mode* et le *Handler mode* ont des modèles de programmation très similaires. Toutefois, le *Thread mode* peut utiliser un pointeur de pile différent appelé *Process Stack Pointer* (PSP). Cela permet à la pile de mémoire des tâches d'application d'être séparée de la pile utilisée par le noyau du système d'exploitation, rendant ainsi le système plus fiable [9].

b) Registres

Le processeur Cortex-M3/M4 dispose de 16 registres principaux de R0 à R15. Treize d'entre eux sont des registres à usage général 32 bits, et les trois autres sont des registres à utilisations spécifiques.

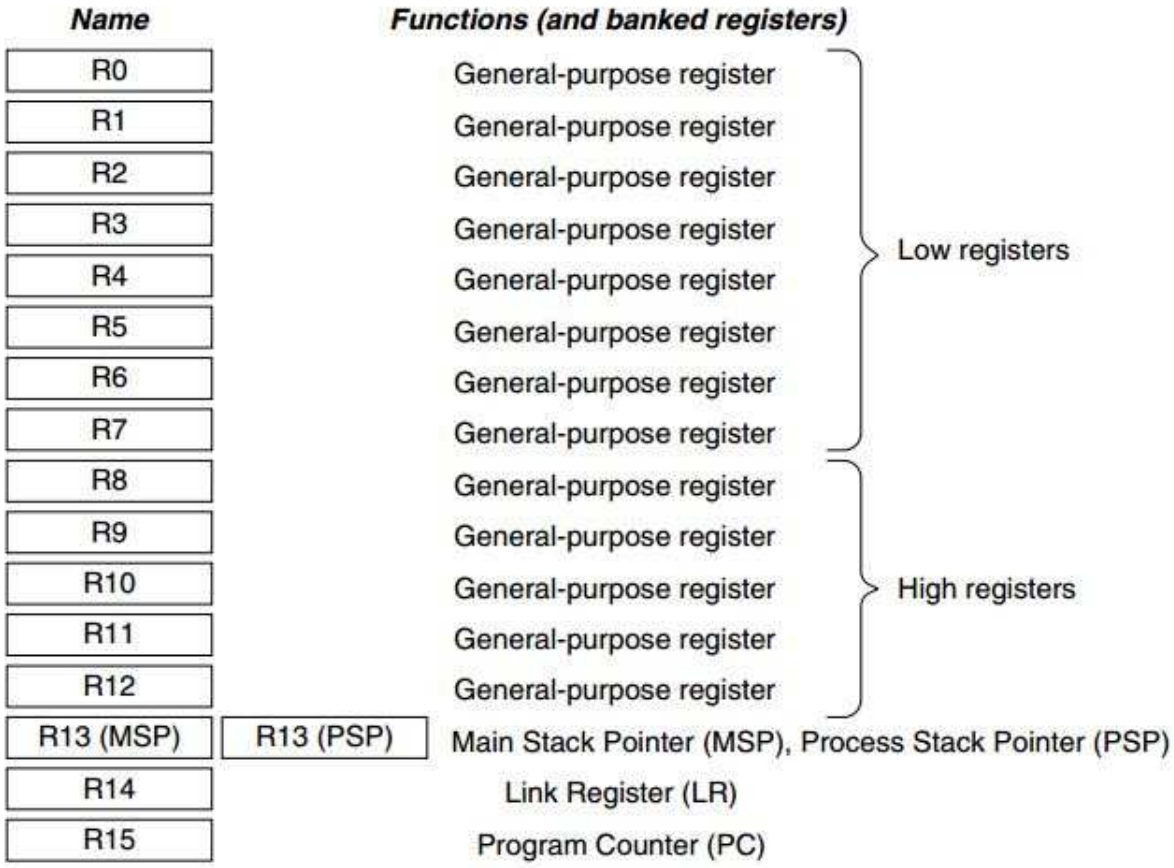


Figure 2.3 : Registres principaux du Cortex-M3 [9]

i. Registres à usage général R0 à R7

Les registres généraux sur la fig 2.3 de R0 à R7 sont appelés «*low registers* ». Ils sont accessibles par toutes les instructions *Thumb* 16-bits et toutes les instructions *Thumb-2* 32-bits. Ils sont tous des registres de 32 bits.

ii. Registres à usage général R8 à R12

Les registres généraux sur la fig 2.3 de R8 à R12 sont appelés « *high registers* ». Ils sont accessibles par toutes les instructions *Thumb-2* 32-bits mais pas par toutes les instructions *Thumb* 16-bits. Ces registres sont tous des registres de 32 bits.

iii. Registres de pointeur de pile R13 (SP)

Le registre R13 est un pointeur de pile (*Stack Pointer*). Il est utilisé pour accéder à la mémoire de pile via des opérations PUSH et POP. Dans le processeur Cortex-M3/M4, il existe deux pointeurs de pile. Cette dualité permet de mettre en place deux mémoires piles séparées. Lorsqu'on utilise le registre R13, on ne peut accéder qu'au registre pointeur de pile en cours; l'autre est inaccessible sauf si on utilise des instructions d'accès aux registres spéciaux. Les deux pointeurs de piles sont les suivants:

- Le pointeur de pile principal (MSP : *Main Stack Pointer*) ou *SP_main* dans la documentation ARM [8]. C'est le pointeur de pile par défaut; il est utilisé par le noyau du système d'exploitation, les gestionnaires d'exception, et tous les codes d'applications qui nécessitent un accès privilégié. Le MSP peut être aussi choisi après un reset, ou lorsque le processeur est en Handler mode.
- Le pointeur de pile de processus (PSP : *Process Stack Pointer*) ou *SP_process* dans la documentation ARM [8]. C'est le pointeur de pile utilisé au niveau du code de l'application (lorsqu'il n'est pas en cours d'exécuter une exception). Le PSP ne peut être utilisé qu'en Thread mode.

Pour la plupart des cas, il n'est pas nécessaire d'utiliser le PSP. De nombreuses applications simples peuvent complètement ne compter que sur le MSP. Le PSP est normalement utilisé quand un OS embarqué est impliqué, où la pile du noyau OS et celle des tâches des applications sont séparées.

iv. Registre de lien R14 (LR)

Le registre R14 est aussi appelé *Link Register* (LR). Il est utilisé pour retenir l'adresse de retour lors de l'appel d'une fonction ou d'un sous-programme. A la fin du sous-programme ou de la fonction, la commande de programme peut retourner au programme appelant et reprendre par le chargement de la valeur de LR dans le compteur de programme (PC : *Program Counter*). Quand un appel de fonction ou sous-programme est fait, la valeur de LR est mise à jour automatiquement. Si une fonction doit appeler une autre fonction ou une sous-routine, elle a besoin de sauvegarder la valeur de LR dans la mémoire pile en premier. Sinon, la valeur actuelle dans LR sera perdue lorsque la fonction appel est effectuée.

v. Registre compteur de programme R15 (PC)

Le registre R15 est le compteur de programme (PC). Il est accessible en lecture et en écriture. En raison de la nature en pipeline du processeur Cortex-M3, quand on lit ce registre, on verra que la valeur est différente de l'emplacement de l'instruction d'exécution, normalement par 4. Par exemple:

```
0x1000 : MOV R0, PC ; R0 = 0x1004
```

Écrire dans le PC entraînera un branchement. Etant donné qu'une adresse d'instruction doit être alignée en demi-mot, le LSB (*Least Significant Bit*) de la valeur du PC est toujours 0. Cependant, lorsqu'on utilise des instructions de lecture en mémoire pour mettre à jour le PC, on doit définir la valeur de LSB du PC à 1 pour indiquer le *Thumb state*. Sinon, une exception peut être déclenchée, car il indique une tentative de passer à l'utilisation des instructions ARM, qui n'est pas prise en charge. Dans les langages de programmation de haut niveau (y compris C, C++), le réglage du LSB est géré automatiquement par le compilateur. Dans la plupart des cas, les branchements et les appels de fonction sont traités par des instructions dédiées à ces opérations. Il est moins fréquent d'utiliser des instructions de traitement de données pour mettre à jour le PC. Toutefois, la valeur du PC est utile pour accéder à des données littérales stockées dans la mémoire programme.

c) Registres spéciaux

A part les registres de la banque de registre, il ya un certain nombre de registres spéciaux, comme illustrés sur la fig 2.4, dans le Cortex-M3/M4.

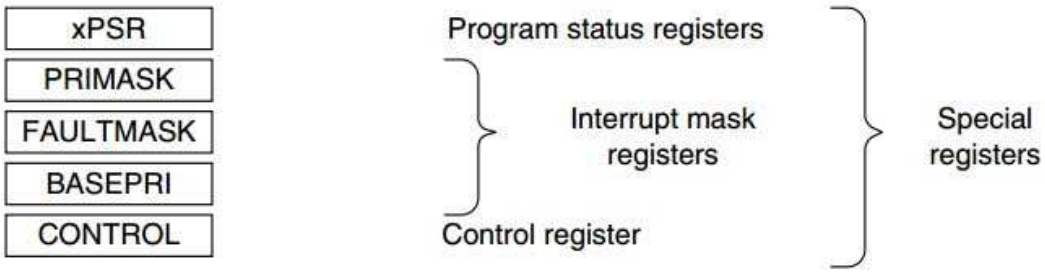


Figure 2.4 : Registres spéciaux [8]

Ces registres spéciaux sont :

- Les registres d'état de programme (xPSR : *Program Status Registers*).
- Les registres de masque d'interruption (PRIMASK, FAULTMASK et BASEPRI).
- Le registre de contrôle (CONTROL).

Les registres spéciaux ne sont pas mappés en mémoire donc ils ne possèdent pas d'adresse mémoire. Ils ne peuvent être accessibles que via les instructions d'accès : MSR et MRS.

i. Les registres d'état de programme (xPSR)

Ils sont au nombre de trois comme l'indique la fig 2.5:

- Registre d'état de programme d'application (APSR : *Appilcation Program Status Register*)
- Registre d'état de programme d'interruption (IPSR : *Interruption Program Status Register*)
- Registre d'état de programme d'exécution (EPSR : *Execution Program Status Register*)

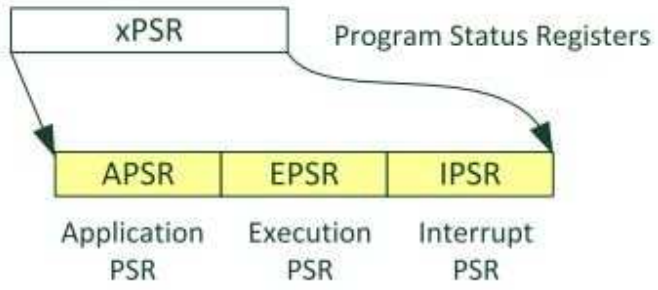


Figure 2.5 : Registres d'état de programme [8]

Les trois PSR peuvent être consultés ensemble ou séparément. On peut modifier le registre APSR en utilisant l'instruction MSR, mais les registres EPSR et IPSR sont en lecture seule. Comme illustrés sur le tableau V, ces trois registres sont des registres de 32-bits et chaque bit ont leurs descriptions listées sur le tableau VI.

Tableau V: Description des registres d'état de programme [9]

	31	30	29	28	27	26:25	24	23:20	19:16	15:10	9	8	7	6	5	4:0
APSR	N	Z	C	V	Q				GE*							
IPSR												Exception Number				
EPSR						ICI/IT	T			ICI/IT						

*GE is available in ARMv7E-M processors such as the Cortex-M4. It is not available in the Cortex-M3 processor.

Tableau VI: Description des attributs des registres d'état de programme [9]

Bit	Description
N	Negative flag
Z	Zero flag
C	Carry (or NOT borrow) flag
V	Overflow flag
Q	Sticky saturation flag (not available in ARMv6-M)
GE[3:0]	Greater-Than or Equal flags for each byte lane (ARMv7E-M only; not available in ARMv6-M or Cortex®-M3).
ICI/IT	Interrupt-Continuable Instruction (ICI) bits, IF-THEN instruction status bit for conditional execution (not available in ARMv6-M).
T	Thumb state, always 1; trying to clear this bit will cause a fault exception.
Exception Number	Indicates which exception the processor is handling.

Ces trois registres peuvent être accédés comme un seul combiné suivant le tab VII.

Tableau VII: Description des registres d'état de programme combiné [9]

	31	30	29	28	27	26:25	24	23:20	19:16	15:10	9	8	7	6	5	4:0
xPSR	N	Z	C	V	Q	ICI/IT	T		GE*	ICI/IT		Exception Number				

*GE is available in ARMv7E-M processors such as the Cortex-M4. It is not available in the Cortex-M3 processor.

ii. Les registres de masque d'interruption (PRIMASK, FAULTMASK et BASEPRI)

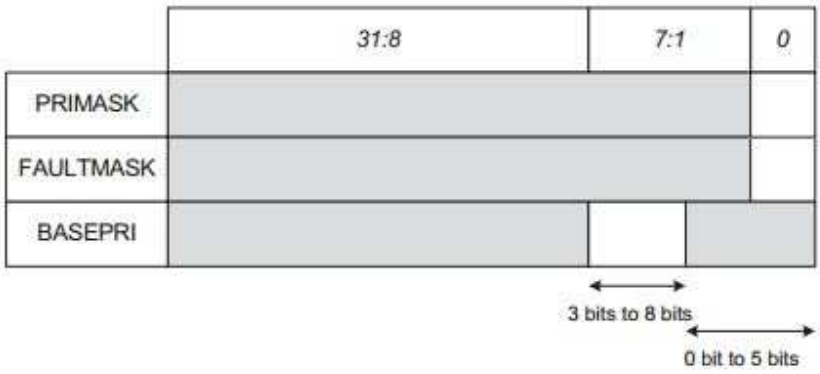
Les registres PRIMASK, FAULTMASK et BASEPRI sont utilisés pour désactiver les exceptions ou pour masquer des interruptions fondées sur des niveaux de priorité. Ils ne sont accessibles qu'au niveau d'accès privilégié (à l'état non privilégié, l'écriture dans ces registres est ignorée et la lecture dans ces registres retourne la valeur zéro). Par défaut, ces registres sont tous à zéro, ce qui signifie que le masquage (désactivation d'exception / interruption) n'est pas actif.

Le registre PRIMASK est un registre de 1-bit. Lorsqu'il est actif, il bloque toutes les exceptions (y compris les interruptions) en dehors du *Non-Masquable Interrupt* (NMI) et le *HardFault exception*. En effet, il augmente le niveau de priorité de l'exception courant de 0, qui est le niveau le plus élevé pour une exception / interruption programmable. L'utilisation la plus courante pour PRIMASK est de désactiver toutes les interruptions pour un processus critique de temps. Après que le processus critique de temps soit terminé, le PRIMASK doit être autorisé à réactiver les interruptions.

Le registre FAULTMASK est un registre de 1-bit. Il est très similaire à PRIMASK. Lorsqu'il est actif, il autorise seulement le NMI, et toutes les interruptions et les exceptions (comme le *Fault Handling Exception*) sont désactivées. Un OS pourrait utiliser FAULTMASK pour désactiver temporairement la gestion des défauts lorsqu'une tâche s'est « crashée ». Dans ce scénario, un certain nombre de défauts pourraient se produire lorsqu'une tâche se bloque. Une fois que le noyau commence le nettoyage, il pourrait ne pas vouloir être interrompu par d'autres défauts causés par le processus qui s'est « crashé ». Par conséquent, le FAULTMASK donne le temps au noyau de l'OS de faire face aux défauts.

Afin de permettre un masquage d'interruption plus souple, l'architecture ARMv7-M offre également le registre BASEPRI, qui masque des exceptions ou des interruptions basées sur le niveau de priorité. La capacité du registre BASEPRI dépend du nombre de niveaux de priorité implémenté dans la conception et est déterminée par les vendeurs du microcontrôleur. La plupart des microcontrôleurs Cortex-M3 ou M4 ont huit niveaux de priorité d'exception programmable ou 16 niveaux, et dans ces cas, les capacités de BASEPRI seront 3 bits ou 4 bits, respectivement ; et elle peut même monter jusqu'à 8 bits comme le montre le tab VIII. Quand il est mis sur une valeur non nulle, il bloque toutes les exceptions (y compris les interruptions) de même niveau de priorité ou de niveau de priorité inferieure, tout en autorisant les exceptions et les interruptions avec un niveau de priorité plus élevé [9].

Tableau VIII: Les registres PRIMASK, FAULTMASK et BASEPRI [9]



Les registres FAULTMASK et BASEPRI ne sont pas disponibles dans ARMv6-M (par exemple, Cortex-M0).

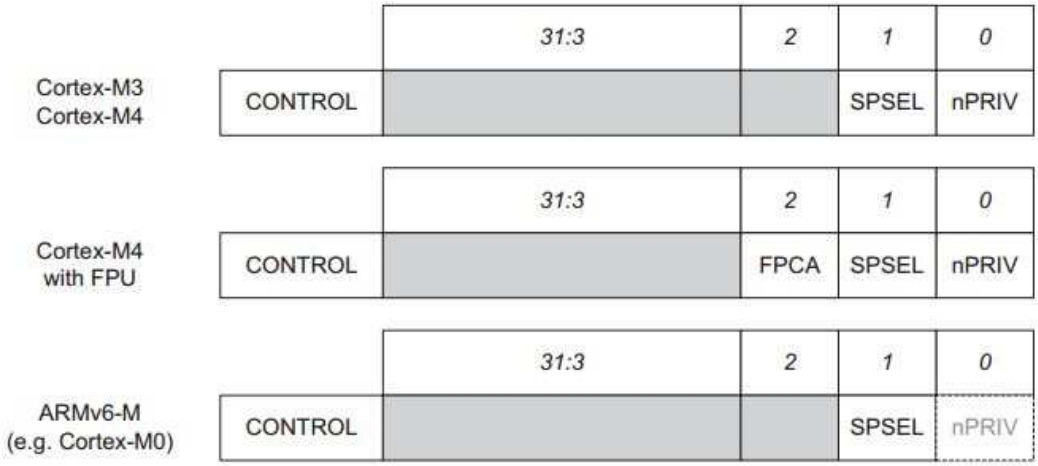
iii. Le registre de contrôle (CONTROL)

Le registre de contrôle définit:

- La sélection du pointeur de pile (*Main Stack Pointer/ Process Stack Pointer*)
- Le niveau d'accès en Thread mode (*Privileged / Unprivileged).*

De plus, pour le processeur Cortex-M4 avec une unité à virgule flottante, comme le montre le tab IX, un bit supplémentaire du registre de contrôle indique si le contexte actuel (code en cours d'exécution) utilise l'unité à virgule flottante ou non.

Tableau IX: Les registres de contrôle des Cortex-M3/M4/M4F et M0 [9]



Le registre de contrôle ne peut être modifié que dans le niveau d'accès privilégié et peut être lu dans les deux niveaux d'accès privilégiés et non privilégiés. Sa définition est comme suit :

- nPRIV (bit 0) : définit le niveau d'accès en *Thread mode*. Lorsque ce bit est 0 (par défaut), le niveau d'accès est privilégié en *Thread mode.* Lorsque ce bit est 1, le niveau est non privilégié en *Thread mode*. En Handler mode, le processeur est toujours dans le niveau d'accès privilégié.
- SPSEL (bit 1) : définit la sélection du pointeur de pile. Lorsque ce bit est 0 (par défaut), le *Thread mode* utilise le MSP comme le montre la fig 2.6. Lorsque ce bit est 1, il utilise le PSP. En mode *Handler*, ce bit est toujours 0 et l'écriture à ce bit est ignorée.
- FPCA (bit 2): *Floating Point Context Active*. Ce bit est seulement disponible dans les Cortex-M4 avec unité de virgule flottante. Le mécanisme de gestion des exceptions utilise ce bit pour déterminer si des registres de l'unité de virgule flottante ont besoin d'être sauvés quand une exception s'est produite. Lorsque ce bit est 0 (par défaut), l'unité de virgule flottante n'a pas été utilisée dans le contexte actuel et donc il n'est pas nécessaire de sauver les registres. Lorsque ce bit est 1, le contexte actuel a utilisé des instructions virgule flottante et on a donc besoin de sauver les registres. Le bit FPCA est réglé automatiquement quand une instruction virgule flottante est exécutée.

Après la réinitialisation, le registre de contrôle est 0. Cela signifie que le *Thread mode* utilise le MSP et l'accès privilégié. Les programmes en *Thread mode* au niveau privilégié peuvent basculer la sélection du pointeur de pile ou passer à un niveau d'accès non privilégié en écrivant dans le registre de contrôle. Cependant, une fois le nPRIV défini, le programme en cours d'exécution dans le *Thread mode* ne peut plus accéder au registre de contrôle [8].

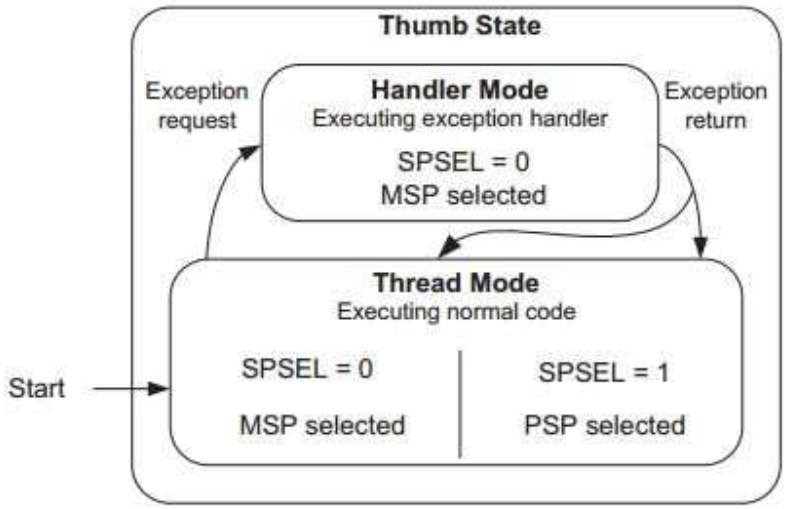


Figure 2.6: Sélection du pointeur de pile [8]

Un programme de niveau d'accès non privilégié ne peut pas revenir au niveau d'accès privilégié. Cela est essentiel pour fournir un modèle de base de l'utilisation de la sécurité. Par exemple, un système embarqué peut contenir des applications non fiables en cours d'exécution dans le niveau d'accès non privilégié. L'autorisation d'accès de ces demandes doit être limitée pour éviter les failles de sécurité ou pour empêcher une application non fiable de « crasher » l'ensemble du système. S'il est nécessaire de revenir à l'utilisation du niveau d'accès privilégié en *Thread mode*, alors le mécanisme d'exception est nécessaire. Au cours du *Handler mode*, le gestionnaire d'exception peut effacer le bit nPRIV. Lors du retour en *Thread mode*, le processeur sera en niveau d'accès privilégié comme illustré sur la fig 2.7.

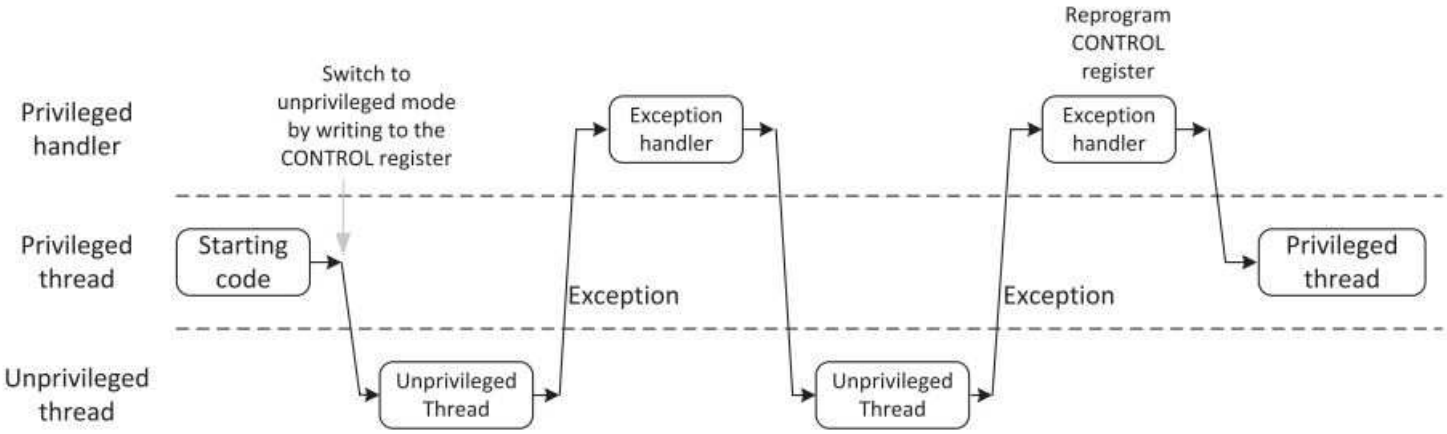


Figure 2.7: Basculement entre « *Privileged thread mode* » et « *Unprivileged thread mode*» [9]

Quand un OS embarqué est utilisé, le registre de contrôle pourrait être reprogrammé à chaque changement de contexte pour permettre à certaines tâches de l'application de s'exécuter avec le niveau d'accès privilégié et les autres avec le niveau d'accès non privilégié. Quatre combinaisons différentes de nPRIV et SPSEL sont possibles, bien que seulement trois d'entre eux soient couramment utilisés dans des applications réelles (voir annexe 2). Dans la plupart des applications simples, sans un système d'exploitation intégré, il n'est pas nécessaire de modifier la valeur du registre de commande. Toute l'application peut s'exécuter dans le niveau d'accès privilégié et peut utiliser seulement le MSP.

d) Registres virgule flottante

Le processeur Cortex-M4 dispose d'une unité de virgule flottante en option. Cela offre des registres supplémentaires pour le traitement de données en virgule flottante, ainsi qu'un registre d'état et de contrôle de virgule flottante (FPSCR : *Floating Point Status and Control Register*). Chacun des registres 32-bits S0 à S31 ("S" pour simple précision) peut être consulté à

l'aide d'instructions virgule flottante, ou consulté comme une paire, avec le symbole de D0 à D15 («D» pour double mot / double précision). Par exemple, S1 et S0 sont jumelés pour devenir D0, et S3 et S2 sont jumelés pour devenir D1 comme le montre la fig 2.8. Bien que l'unité de virgule flottante dans le Cortex-M4 ne supporte pas le calcul en virgule flottante à double précision, on peut toujours utiliser des instructions virgule flottante pour transférer des données en double précision.

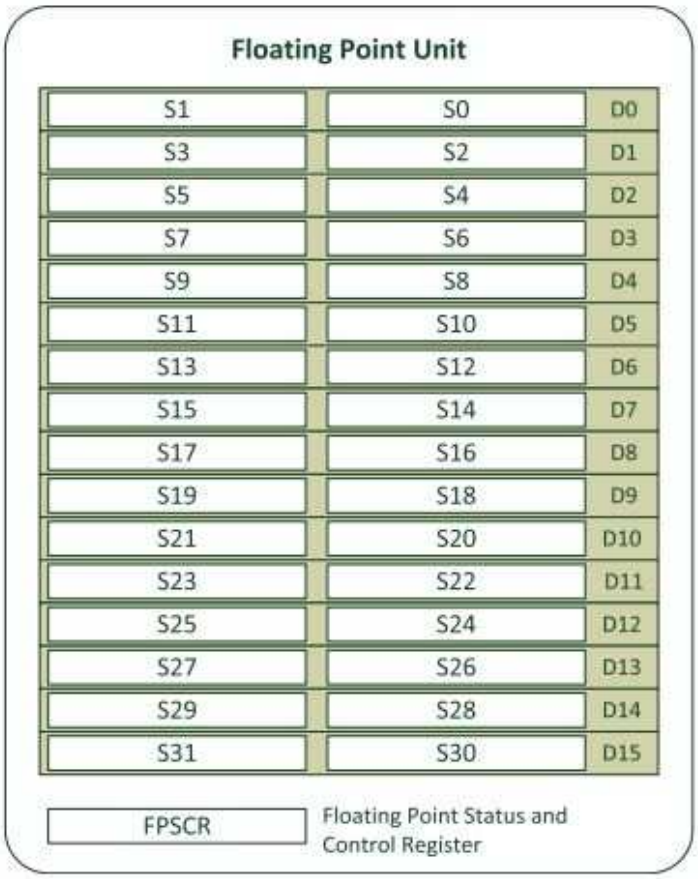


Figure 2.8: Registres dans la FPU [9]

i. Registre d'état et de contrôle de virgule flottante

Le FPSCR contient divers champs de bits comme illustrés dans le tab X pour différentes raisons:

- pour définir certains comportements de fonctionnement en virgule flottante
- pour fournir des informations d'état sur les résultats de l'opération en virgule flottante

Par défaut, le comportement est configuré pour être conforme à la norme IEEE 754 (opération de simple précision). Dans les applications normales, il n'est pas nécessaire de modifier les paramètres de contrôle de l'opération en virgule flottante. On peut voir dans l'annexe 2 la liste des descriptions des champs de bits dans FPSCR.

Tableau X: Champs de bit dans le registre FPSCR [9]

	31	30	29	28	27	26	25	24	23:22	21:8	7	6:5	4	3	2	1	0
FPSCR	N	Z	C	V		AHP	DN	FZ	RMode	Reserved	IDC	Reserved	IXC	UFC	OFC	DZC	IOC

Reserved

Les bits d'exception dans FPSCR peuvent être utilisés par le logiciel pour détecter des anomalies dans les opérations en virgule flottante.

ii. Registres de mémoire mappée de virgule flottante

En plus des bancs de registre de virgule flottante et du FPSCR, l'unité de virgule flottante possède aussi plusieurs registres de mémoire mappée supplémentaires dans le système. Par exemple, le registre de contrôle d'accès du coprocesseur (CPACR : *CoProcessor Access Control Register*) est utilisé pour activer ou désactiver l'unité de virgule flottante comme le montre le tab XI. Par défaut, l'unité de virgule flottante est désactivée pour réduire la consommation d'énergie.

Tableau XI: Champs de bit dans le registre CPACR [9]

	31:24	23:22	21:20	19:0
CPACR	Reserved	CP11	CP10	Reserved

Bit field encoding:
00 – Access denied
01 – Privileged access only
10 – Reserved (unpredictable)
11 – Full accesses

II.3. SYSTEME DE MEMOIRE

a) Mémoire mappée (*Memory map*)

L'espace d'adressage de 4 Go sur les processeurs Cortex-M3 est divisé en un certain nombre de régions de mémoire comme le montre la fig 2.9. Le partitionnement est basé sur des usages typiques de sorte que les différentes zones soient conçues pour être utilisées principalement pour:

- L'accès au code de programme (par exemple, dans la région de CODE)
- L'accès aux données (par exemple, dans la région de SRAM)

- Les périphériques (par exemple, dans la région périphérique)
- Le contrôle interne du processeur et les composants de débogage (par exemple le PPB)

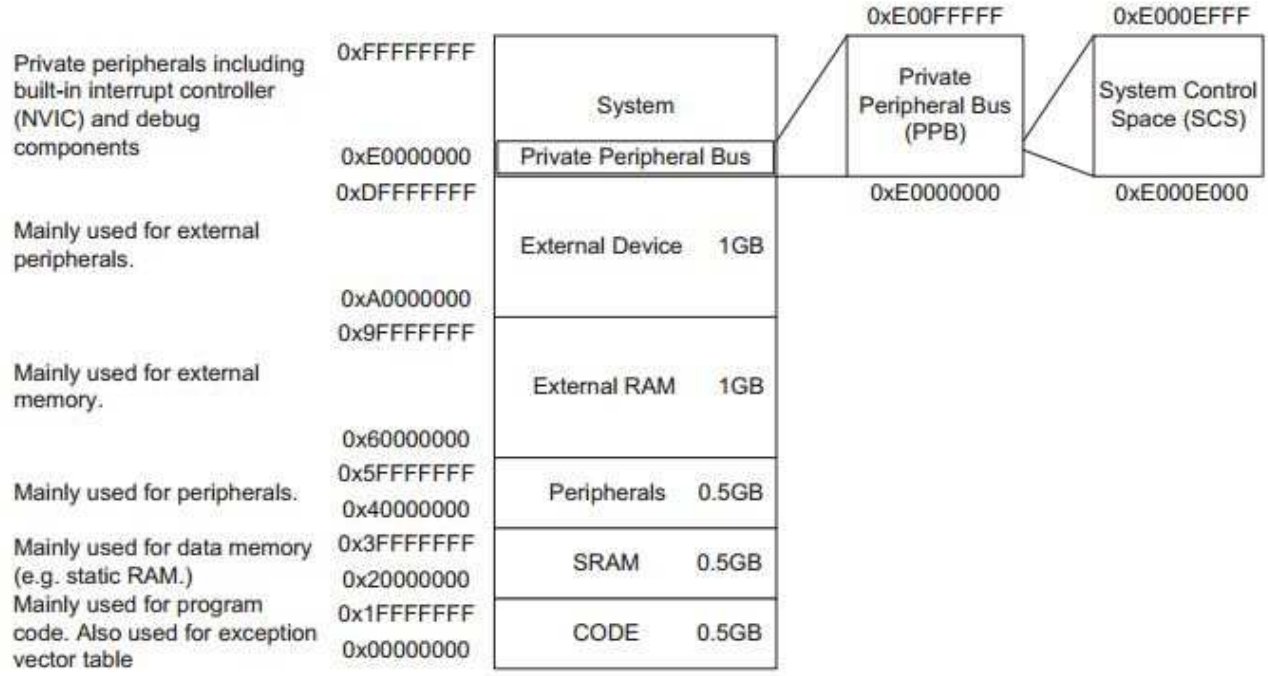


Figure 2.9: Mémoire mappée [8]

L'architecture permet également une grande flexibilité pour permettre aux régions de la mémoire d'être utilisées à d'autres fins. Par exemple, les programmes peuvent être exécutés à partir du CODE ainsi que dans la région de SRAM, et un microcontrôleur peut également intégrer des blocs SRAM dans la région CODE.

La disposition de la mémoire mappée est cohérente entre tous les processeurs Cortex-M. Par exemple, l'espace d'adressage PPB accueille les registres NVIC, les registres de configuration du processeur, ainsi que des registres pour les composants de débogage. Il en est de même sur tous les appareils Cortex-M. Cela rend plus facile la portabilité du logiciel d'un appareil Cortex-M à l'autre, et permet une meilleure réutilisation du logiciel. Il rend également plus facile pour les fournisseurs d'outils, comme le contrôle de débogage pour les appareils Cortex-M3/M4, de travailler de la même manière.

b) Mémoire pile (*Stack memory*)

Comme dans presque toutes les architectures de processeur, les processeurs Cortex-M ont besoin de mémoire pile pour fonctionner. C'est une sorte de mécanisme d'utilisation de mémoire qui permet à une partie de la mémoire à devenir comme une mémoire tampon LIFO (*Last In First Out*) pour le stockage de données. Les processeurs ARM utilisent la mémoire

principale du système pour les opérations de mémoire de la pile, et ont l'instruction PUSH pour stocker des données dans la pile et l'instruction POP pour récupérer les données de la pile. Le pointeur courant de pile (R13) sélectionnée est automatiquement ajusté pour chaque opération de PUSH et POP.

La mémoire pile peut être utilisée pour:

- Le stockage temporaire des données originales quand une fonction en cours d'exécution doit utiliser des registres (dans la banque de registre) pour le traitement de données. Les valeurs peuvent être restaurées à la fin de la fonction de sorte que le programme qui a appelé la fonction ne perde pas ses données.
- Le transfert des informations à des fonctions ou des sous-programmes.
- Stocker des variables locales.
- Maintenir l'état du processeur et les valeurs des registres dans le cas où des exceptions se produiraient.

Les processeurs Cortex-M utilisent un modèle de mémoire pile appelée "*full descending stack*". Lorsque le processeur est lancé, le SP est réglé pour pointer à la fin de l'espace mémoire réservé pour la mémoire pile. Pour chaque opération PUSH, le processeur décrémente premièrement la SP, puis stocke la valeur dans l'emplacement de mémoire indiqué par SP. Pendant les opérations, le SP pointe à l'emplacement de mémoire où les dernières données ont été sauvées dans la pile comme le montre la fig 2.10. Dans une opération de POP, la valeur dans l'emplacement de mémoire indiqué par SP est lue, puis la valeur de SP est incrémentée automatiquement [9].

L'utilisation la plus courante des instructions PUSH et POP est de sauver le contenu des banques de registre quand un appel de fonction/sous-routine est fait.

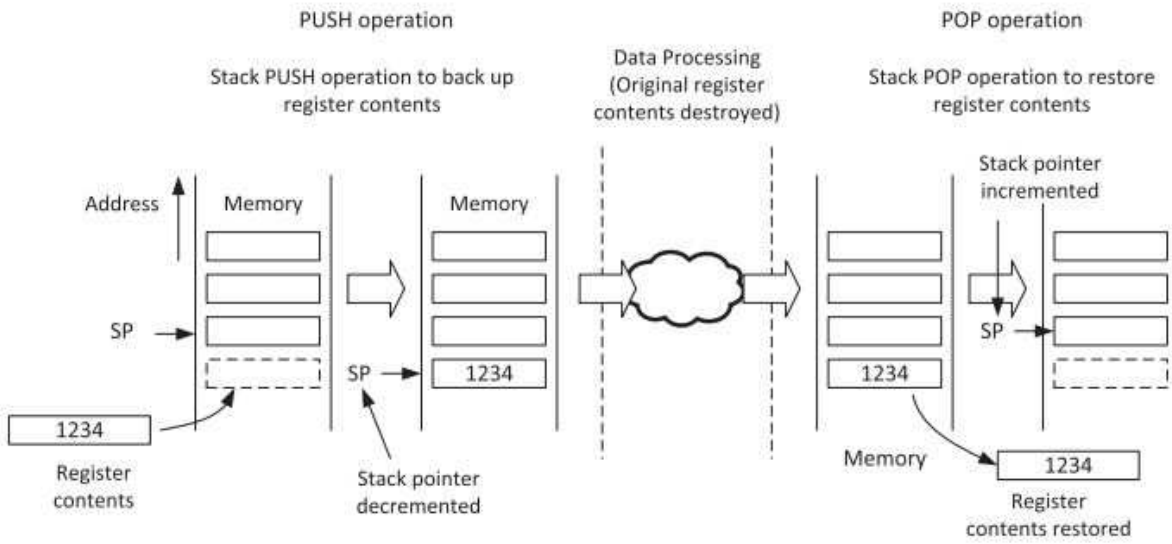


Figure 2.10: PUSH AND POP [9]

Le MSP est le pointeur de la pile par défaut utilisé par tous les gestionnaires d'exceptions et après la réinitialisation. Le PSP est un pointeur de pile alternatif qui ne peut être utilisé qu'en Thread mode. Il est généralement utilisé pour les tâches d'application dans les systèmes embarqués fonctionnant sous un OS embarqué. En effet, lorsque les systèmes embarqués utilisent un système d'exploitation intégré, ils utilisent souvent des zones de mémoire séparées pour la mémoire pile des applications et la mémoire pile du « *kernel* ».

c) MPU (*Memory Protection Unit*)

Le MPU est un dispositif programmable qui peut être utilisé pour définir les permissions d'accès en mémoire (par exemple, un accès privilégié uniquement ou accès complet) et les attributs de la mémoire (par exemple, le *buffer*, la mise en cache) pour différentes régions de mémoire.
Le MPU est utilisé pour faire un système embarqué plus robuste, et dans certains cas, il peut rendre le système plus sûr en:

- Empêchant les tâches d'application de corrompre la pile ou les mémoires de données utilisées par d'autres tâches ou par le noyau du système d'exploitation.
- Empêchant les tâches non privilégiées d'accéder à certains périphériques qui peuvent être critiques pour la fiabilité ou la sécurité du système
- Définissant la SRAM ou l'espace de RAM comme non-exécutable (*eXecute Never*: XN) pour empêcher les attaques d'injection de code [9].

Si un accès mémoire viole les droits d'accès définis par le MPU, ou accède à un emplacement de mémoire qui n'est pas défini dans les régions MPU, le transfert sera bloqué et une exception (*Fault exception*) sera déclenchée. Le gestionnaire d'exception déclenché pourra être soit « *MemManage Fault* » (*Memory Management*) s'il est actif ou « *HardFault exception* », selon les niveaux de priorité courants. Le gestionnaire d'exception peut alors décider si le système doit être réinitialisé ou tout simplement mettre fin à la tâche en question. Le MPU doit être programmé et activé avant utilisation, il peut être mis en place de plusieurs façons.

Dans les systèmes sans OS embarqué, le MPU peut être programmé pour avoir une configuration statique. La configuration peut être utilisée pour des fonctions telles que:

- La définition d'une région RAM / SRAM à être en lecture seule pour protéger les données importantes de corruption accidentelle
- La définition d'une portion d'espace RAM / SRAM au bas de la pile inaccessible pour détecter un débordement de pile
- La définition d'une région RAM / SRAM à être XN pour prévenir les attaques d'injection de code

- La définition des paramètres d'attribut de mémoire qui peuvent être utilisés par la mémoire cache ou par les contrôleurs de mémoire.

Dans les systèmes avec un OS embarqué, le MPU peut être programmé à chaque changement de contexte de sorte que chaque tâche d'application puisse avoir une configuration différente de MPU. De cette façon, on peut:

- Définir les autorisations d'accès mémoire afin que les opérations de la pile d'une tâche d'application ne peuvent accéder qu'à leur propre l'espace de pile alloué, empêchant ainsi les corruptions d'autres piles dans le cas d'une fuite de pile.
- Définir des autorisations d'accès mémoire de sorte qu'une tâche d'application ne puisse avoir accès qu'à un ensemble limité de périphériques
- Définir des autorisations d'accès mémoire de sorte qu'une tâche d'application ne puisse accéder à ses propres données et/ou programmes de données (elles sont beaucoup plus difficiles à mettre en place, car dans la plupart des cas, le système d'exploitation et le code de programme sont compilés ensemble, donc les données pourraient être mélangées ensemble dans la « *memory map* ») [9].

Les systèmes avec un OS intégré peuvent également utiliser une configuration statique.

II.4. EXCEPTION ET INTERRUPTION

Les interruptions sont une caractéristique commune disponible dans presque tous les microcontrôleurs. Les interruptions sont généralement des événements générés par le matériel (par exemple, par les périphériques) qui provoquent des changements dans le contrôle des flux du programme en dehors d'une séquence programmée normale. Quand un périphérique ou matériel interrompt le processeur, généralement la séquence suivante se produit:

1- Le périphérique affirme une demande d'interruption au processeur
2- Le processeur interrompt la tâche en cours d'exécution
3- Le processeur exécute une routine de service d'interruption (ISR) pour desservir le périphérique, et éventuellement nettoyer la demande d'interruption par le logiciel si nécessaire
4- Le processeur reprend la tâche préalablement suspendue

Tous les processeurs Cortex-M offrent le NVIC pour la gestion des interruptions. En plus des demandes d'interruption, il y a d'autres événements appelés «*exceptions*» qui ont besoin d'entretien. Dans la terminologie ARM, une interruption est un type d'exception. Dans un microcontrôleur Cortex-M, le NVIC reçoit les demandes d'interruption provenant de diverses sources, comme le montre la fig 2.11.

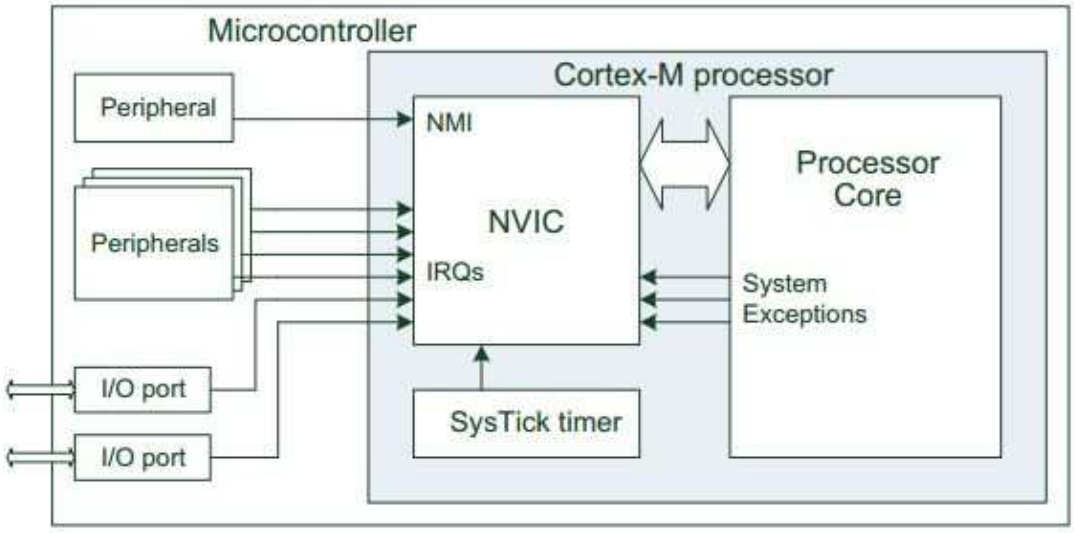


Figure 2.11: Diverses sources d'interruption [9]

Les processeurs Cortex-M offrent une architecture d'exception riche en fonctionnalités qui prend en charge un certain nombre d'exceptions du système et d'interruptions externes. Les exceptions du système sont numérotées de 1 à 15, et au-dessus de 15, pour les entrées d'interruption (entrées au processeur, mais pas nécessairement accessibles sur les broches d'E/S) communément appelée IRQ (*Interrupt Request*) comme indiquée dans le tab XII. La plupart des exceptions, y compris toutes les interruptions, ont des priorités programmables, et quelques exceptions du système ont la priorité fixée.

Les différents microcontrôleurs Cortex- M3/M4 peuvent avoir des nombres différents de sources d'interruption (à partir de 1 à 240) et des nombres différents de niveaux de priorité. La raison en est que les concepteurs de puces peuvent configurer la conception du Cortex-M3 ou Cortex-M4 pour différentes exigences d'application.

Tableau XII : Types d'exceptions [9]

Exception Number	Exception Type	Priority	Function
1	Reset	–3 (Highest)	Reset
2	NMI	–2	Nonmaskable interrupt
3	Hard fault	–1	All classes of fault, when the corresponding fault handler cannot be activated because it is currently disabled or masked by exception masking
4	MemManage	Settable	Memory management fault; caused by MPU violation or invalid accesses (such as an instruction fetch from a nonexecutable region)
5	Bus fault	Settable	Error response received from the bus system; caused by an instruction prefetch abort or data access error
6	Usage fault	Settable	Usage fault; typical causes are invalid instructions or invalid state transition attempts (such as trying to switch to ARM state in the Cortex-M3)
7–10	—	—	Reserved
11	SVC	Settable	Supervisor call via SVC instruction
12	Debug monitor	Settable	Debug monitor
13	—	—	Reserved
14	PendSV	Settable	Pendable request for system service
15	SYSTICK	Settable	System tick timer
16–255	IRQ	Settable	IRQ input #0–239

Une partie du processeur fusionnée dans l'unité NVIC est le SCB (*System Control Block*). Le SCB contient divers registres pour:

- La commande des configurations du processeur (par exemple, les modes de faible puissance)
- Les informations sur l'état de défaut (*fault status register*)
- Le VTOR (*Vector Table Offset Register*)

Le SCB est mappé en mémoire. Comme pour les registres NVIC, les registres de la SCB sont accessibles à partir du *System Control Space* (SCS).

CHAPITRE III : PROGRAMMATION EN C SUR UN MICROCONTROLEUR ARM CORTEX-M3/M4

III.1. LES MICROCONTROLEURS ARM CORTEX-M3/M4

Le processeur Cortex-M3 n'est que l'unité centrale de traitement (CPU) d'un microcontrôleur. Un certain nombre d'autres éléments sont nécessaires pour l'ensemble du microcontrôleur Cortex-M3. On sait qu'ARM Ltd ne produit pas les puces directement mais il ne vend que la licence de l'architecture du CPU aux fabricants de la puce. On appelle cela : licence de propriété intellectuelle *« IP* (*Intellectual Property*) *Licencing* » [9]. En plus des modèles de processeur, *ARM Ltd* octroie aussi des licences IP sur d'autres éléments de la puce comme le montre la fig 3.1.

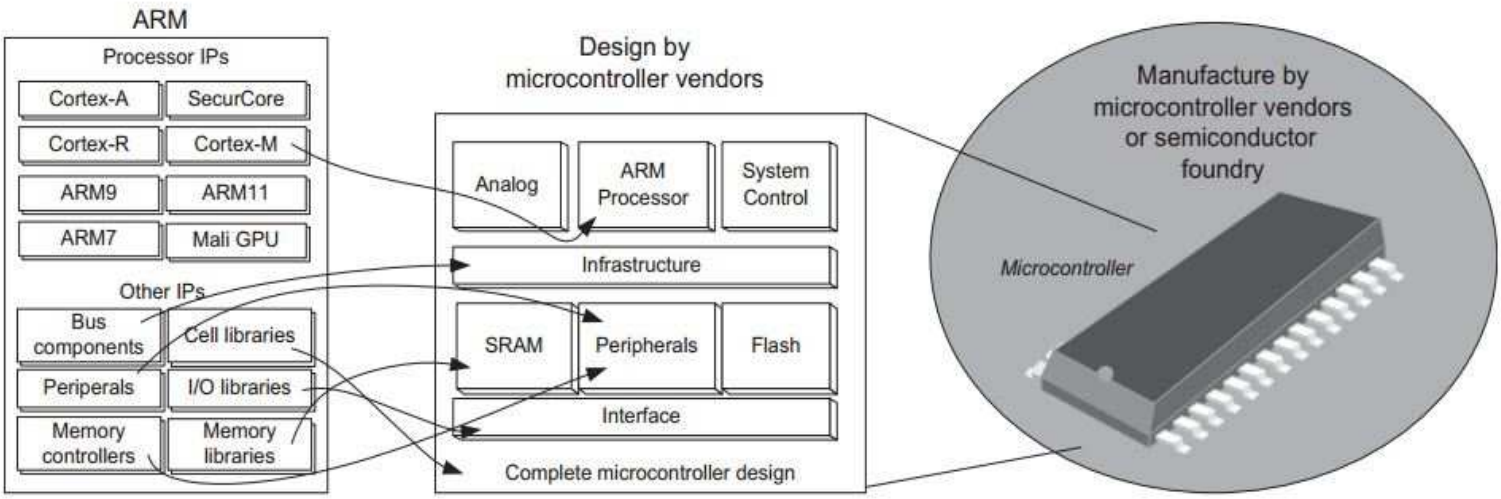


Figure 3.1 : Les produits IP d'ARM [9]

Après que les fabricants des puces obtiennent la licence du processeur Cortex-M3 avec éventuellement d'autres éléments de *ARM IP*, ils peuvent les mettre dans leurs conceptions. Ils peuvent ajouter d'autres éléments, de leur choix, venant de leurs propres conceptions et/ou venant des licences IP d'autres firmes. En effet, bien que de nombreux fournisseurs de microcontrôleurs utilisent les processeurs ARM Cortex-M comme choix de leur CPU, le système de mémoire, la mémoire mappée, les périphériques, et les caractéristiques de fonctionnement (par exemple, la vitesse d'horloge et la tension) peuvent être conçus différemment d'un produit à l'autre. Cela permet aux fabricants de microcontrôleurs d'ajouter des fonctionnalités supplémentaires dans leurs produits. Par conséquent, les caractéristiques (taille de mémoire, périphériques et fonctions) des puces à base de processeur Cortex-M3 de différents fabricants seront variées d'une puce à l'autre.

Dans la conception typique de microcontrôleur, le processeur prend seulement une petite partie de la surface du silicium. Les autres parties sont prises principalement par des mémoires, la génération d'horloge (par exemple, PLL : *Phase Locked Loop*), des systèmes de bus, des différents contrôleurs et des périphériques (les unités d'interfaces d'E / S, les interfaces de communication, les timers, ADC : *Analog to Digital Converter*, DAC : *Digital to Analog Converter*, Ethernet, USB : *Universal Serial Bus*, PWM : *Pulse Width Modulation*, etc.), comme indiquées sur la figure 3.2.

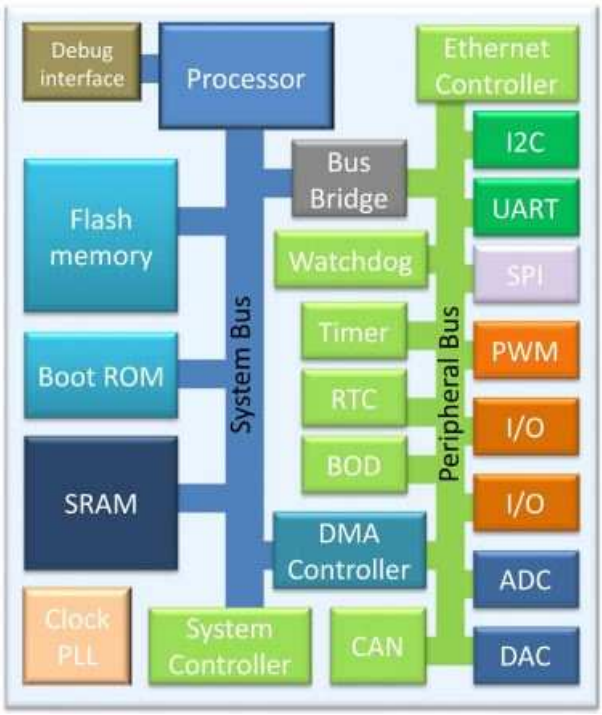


Figure 3.2 : Les différents blocs dans un microcontrôleur [9]

III.2. DEPLOIEMENT

a) Choix du microcontrôleur ARM Cortex-M3/M4

Dans la plupart des projets, les caractéristiques et les spécifications du microcontrôleur vont certainement affecter le choix des produits Cortex-M3/M4. Par exemple :

- Les périphériques: pour de nombreuses applications, les différents supports périphériques peuvent être le principal critère. Cependant, plus le nombre de périphériques augmente, plus la consommation d'énergie du microcontrôleur et le prix sont affectés.
- La mémoire: les microcontrôleurs Cortex-M3 peuvent avoir une mémoire Flash de plusieurs kilo-octets à plusieurs méga-octets. En outre, la taille de la mémoire interne

peut être également importante. Habituellement, ces facteurs auront aussi un impact direct sur le prix.

- La vitesse d'horloge : cela peut aussi être un critère car certaines applications nécessitent une vitesse d'horloge assez élevée et d'autres non.
- Les boitiers : le Cortex-M3 peut être disponible dans de nombreux packages différents, en fonction de la décision des fabricants de puces. De nombreux Cortex-M3 sont disponibles en boitiers à broches peu nombreuses, ce qui les rend appropriés pour les applications à faible coût [8].

En plus de la mémoire, les options de périphériques, et la vitesse de fonctionnement, un certain nombre d'autres facteurs font d'un produit Cortex -M3 différent d'un autre. Donc le choix peut être affecté car le *design* fourni par *ARM Ltd* contient un certain nombre de fonctionnalités configurables, telles que :

- Le nombre d'interruptions externes
- Le nombre de niveaux de priorité d'interruption (largeur de registres de niveau de priorité)
- Avec Unité de protection de mémoire (MPU) ou sans MPU
- Avec *Embedded Trace Macrocell* (ETM) ou sans ETM
- Le choix de l'interface de débogage (*Serial-Wire* (SW), *Join Test Action Group* (JTAG), ou les deux) [9].

Actuellement, *ARM Ltd* a vendu jusqu'à 240 licences pour le Cortex-M. Les principaux puissants fournisseurs de microcontrôleurs ARM Cortex-M sont listés sur la fig 3.3.

Figure 3.3 : Les principaux grands fabricants de puces à ARM Cortex- M [16]

Pour notre travail, on a choisi le microcontrôleur STM32F407ZGT6 ARM Cortex-M4 de *STMicroelectronics*, dont les caractéristiques se trouvent en annexe 3.

b) Outils de développement

Pour programmer un Cortex-M3, un certain nombre d'outils est nécessaire. Typiquement, ils comprennent les éléments suivants:

- Un compilateur et / ou un assembleur : logiciel pour compiler les codes en langage C ou en langage d'assemblage. Presque toutes les suites de compilateur C comprennent un assembleur.
- Un simulateur Jeu d'instructions: logiciel pour simuler l'exécution de l'instruction pour le débogage dans les premiers stades de développement des logiciels. Cette étape est facultative.
- Un *In-Circuit Emulator* (ICE) ou une sonde de débogage: un dispositif matériel pour connecter la hôte de débogage (généralement un ordinateur personnel) au circuit de cible. L'interface peut être soit JTAG ou SW.
- Un microcontrôleur Cortex- M3 ou M4
- Un Trace capture : Un matériel et logiciel en option pour capturer les instructions de traçages ou les sorties des modules *Data Watchtpoint* and *Trace* (DWT) et *Instrumentation Trace* Macrocell (ITM). Parfois la fonction Trace capture est intégrée dans l'ICE.
- Un système d'exploitation intégré. Cela aussi est facultatif car plusieurs applications n'ont pas besoin de fonctionner sur un OS.

i. Kit de développement

Les outils logiciels cités précédemment sont parfois inclus dans un seul logiciel appelé un IDE (*Integrated Developpement Environment*). Sur le marché, il y a plusieurs kits de développement pour les ARM Cortex-M mais les plus courants sont :

- *IAR Embedded Workbench for ARM* de IAR
- *TrueSTUDIO* d'*Atollic*
- Eclipse comme IDE, avec les outils GNU et *GNU ARM Eclipse Plug-ins*
- *CoIDE* de *CooCox*
- *Keil MDK-ARM* de *Keil*
- *ARM Developpement Studio*
- *ARM mbed* (IDE en ligne)

Cette liste n'est pas exhaustive. Il se peut que les fabricants de puces possèdent leurs propres IDE pour leurs microcontrôleurs; c'est le cas, par exemple de *LPCXpresso* de NXP (anciennement connu sous le nom de *Phillips semiconductors*) qui ne prend en charge que les microcontrôleurs de la famille LPC de NXP.

Pour notre travail, l'environnement de développement sera *Keil MDK-ARM* 5. Le *MDK-ARM* de *Keil* a l'avantage de prendre en charge tous les microcontrôleurs ARM Cortex-M puisqu'il appartient à la société *ARM Ltd* lui-même. Il est payant mais il existe une version lite téléchargeable gratuitement sur leur site. Il contient plusieurs composants logiciels, tels que :

- L'IDE de *Microvision*
- Les outils de compilation d'ARM comme:
    - Compilateur C/C++
    - Assembleur
    - Utilitaires et éditeurs de liens
    - Débogueur
    - Simulateur
    - *RTX Real-Time OS Kernel*
    - Algorithme de programmation *Flash*
    - Référence pour les codes start-up pour plus de 1000 microcontrôleurs.

ii. Compilateur et assembleur

Beaucoup d'outils de développement sont disponibles pour les microcontrôleurs ARM. La majorité d'entre eux supportent le langage C et le langage d'assemblage. Des projets intégrés peuvent être développés en C ou en langage d'assemblage, ou en un mélange des deux. Dans la plupart des cas, la génération du programme peut être résumée dans un diagramme représenté sur la fig 3.4.

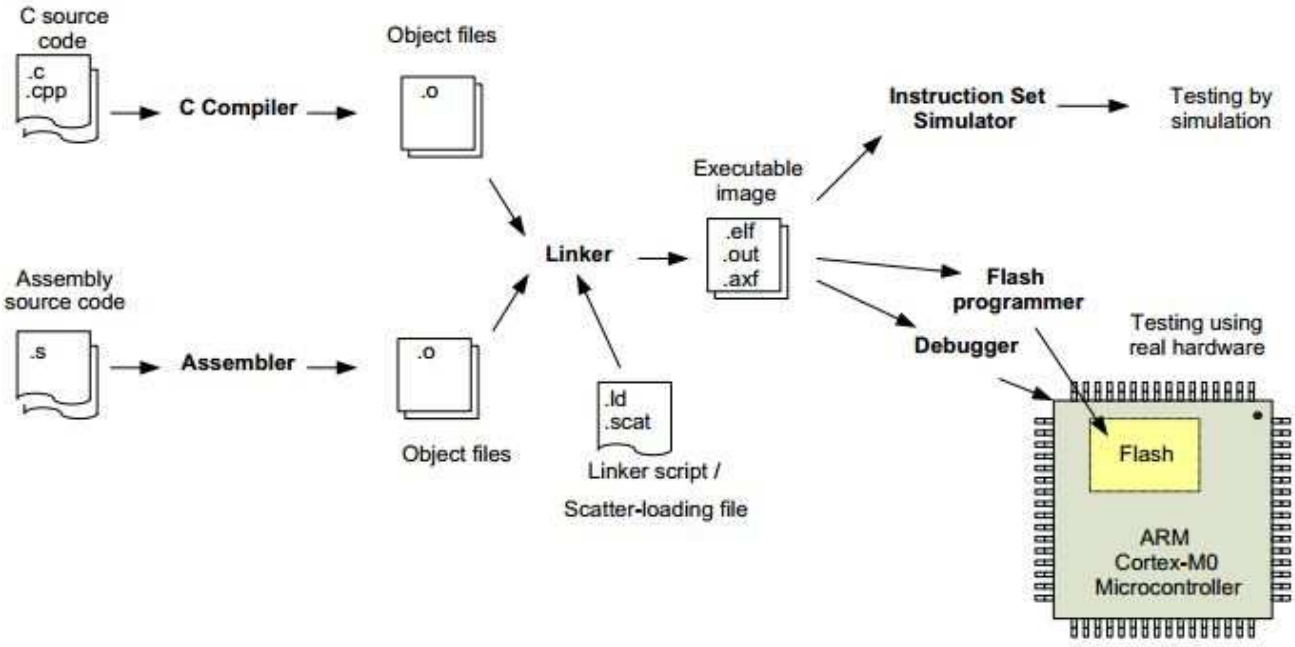


Figure 3.4 : Exemple de flux de compilation pour un ARM Cortex-M [17]

Il existe de nombreux compilateurs C/C++ pour l'ARM. Il se peut que les fournisseurs d'IDE ou éventuellement les fabricants des puces possèdent leur propre compilateur de langage C /C++. Mais il existe deux compilateurs très utilisés : le GNU C/C++ Compiler et ARM C/C++ Compiler. Etant donné que nous travaillons sur *Microvision Keil MDK-ARM*, notre compilateur sera logiquement ARM C/C++ Compiler. Ces compilateurs supportent l'ISO Standard C /C++.

Dans la plupart des applications simples, les programmes peuvent être complètement écrits en langage C. Le compilateur C compile le code du programme C en fichiers objets puis génère le fichier d'images du programme exécutable en utilisant l'éditeur de liens. Dans le cas du compilateur GNU C, la compilation et les étapes de liaison sont souvent fusionnées en une seule étape. Les projets qui nécessitent la programmation d'assemblage utilisent l'assembleur pour générer du code objet à partir du code source de montage. Les fichiers objets peuvent ensuite être liés à d'autres fichiers objets dans le projet pour produire une image exécutable. Outre le code de programme, les fichiers objet et l'image exécutable peuvent également contenir diverses informations de débogage.

Un fichier image pour le microcontrôleur Cortex-M3 contient souvent les composants suivants:

- Routine de démarrage (*startup routine*)
- Code de programme (code d'application, de données et de systèmes)
- Code de la bibliothèque C (codes de programme pour les fonctions de la bibliothèque C insérés au moment de la liaison)

Selon les outils de développement, il est possible de spécifier la mise en mémoire pour l'éditeur de lien en utilisant les options de ligne de commande. Cependant, dans des projets utilisant le compilateur GNU C, un script de liaison est normalement requis pour spécifier la mise en mémoire. Dans les outils de développement ARM, les scripts d'éditeur de liens sont souvent appelés « *scatter-loading files* ». Si on utilise le kit de développement de microcontrôleur *Keil* (MDK), le fichier *scatter-loading* est généré automatiquement à partir de la fenêtre de mise en mémoire [17].

iii. Débogueur

Après que l'image exécutable est générée, on peut la tester en la téléchargeant dans la mémoire *flash* ou RAM interne du microcontrôleur. L'ensemble du processus peut être assez facile; la plupart des suites de développement viennent avec un environnement de développement intégré convivial (IDE). Lorsqu'ils travaillent ensemble avec un débogueur (un ICE, une sonde de débogage, ou un adaptateur USB-JTAG), on peut créer un projet, une application, et télécharger l'application embarquée dans le microcontrôleur en quelques étapes comme présentées sur la fig 3.5.

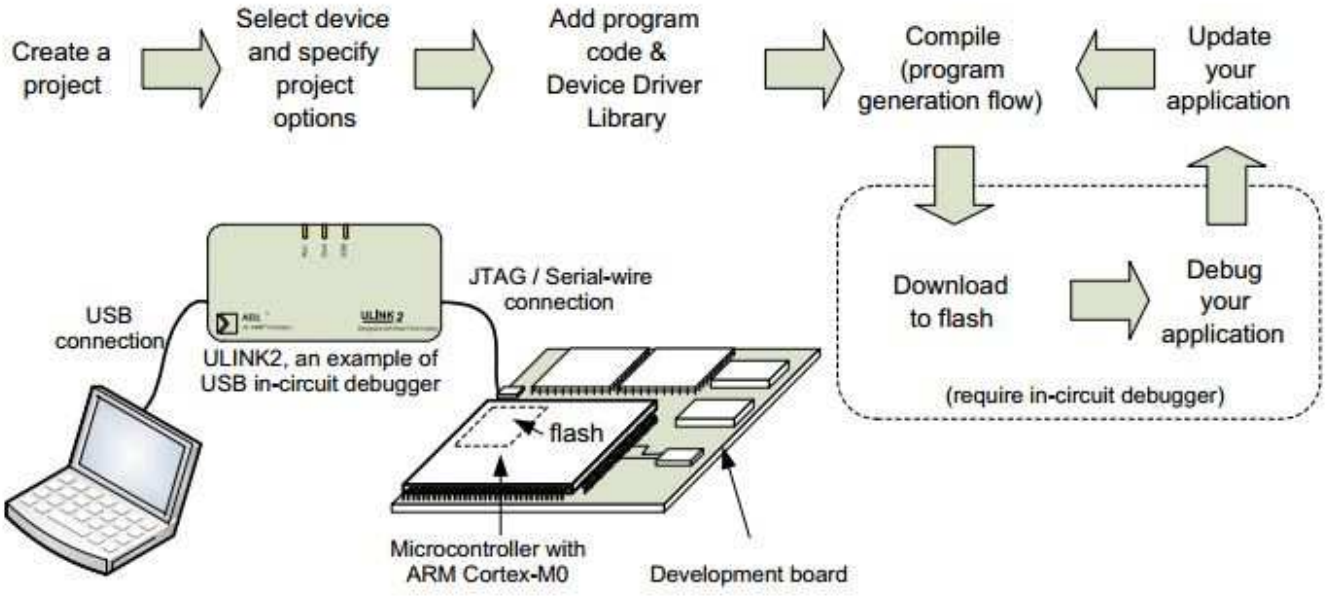


Figure 3.5 : Exemple de flux de développement pour un ARM Corte-M [17]

Le logiciel de débogage de *Keil* dans le *MDK-ARM* peut être utilisé avec un certain nombre de différents adaptateurs de débogage (appelé aussi débogueur). Cela comprend des adaptateurs de débogage commerciaux tels que:

- *Keil ULINK2, ULINK Pro, ULINK-ME*
- *Signum Systems JTAGjet*
- *J-Link, J-Trace de Segger*
- *ST-LINK, ST-LINK V2*
- *Silicon Labs UDA Debugger*
- *Stellaris ICDI (Texas Instrument)*
- *NULink Debugger*
- *DSTREAM*

La fonction de programmation *flash* peut être effectuée par le logiciel de débogage dans la suite de développement comme le montre la fig 3.6. Dans certains cas, la fonction est assurée par un utilitaire de programmation *flash* téléchargeable à partir du site Web du fournisseur du microcontrôleur. Alors, le programme peut être testé en exécutant sur le microcontrôleur, et en connectant le débogueur au microcontrôleur. L'exécution du programme peut être contrôlée et les opérations peuvent être observées. Toutes ces actions seront effectuées via l'interface de débogage du processeur Cortex-M3.

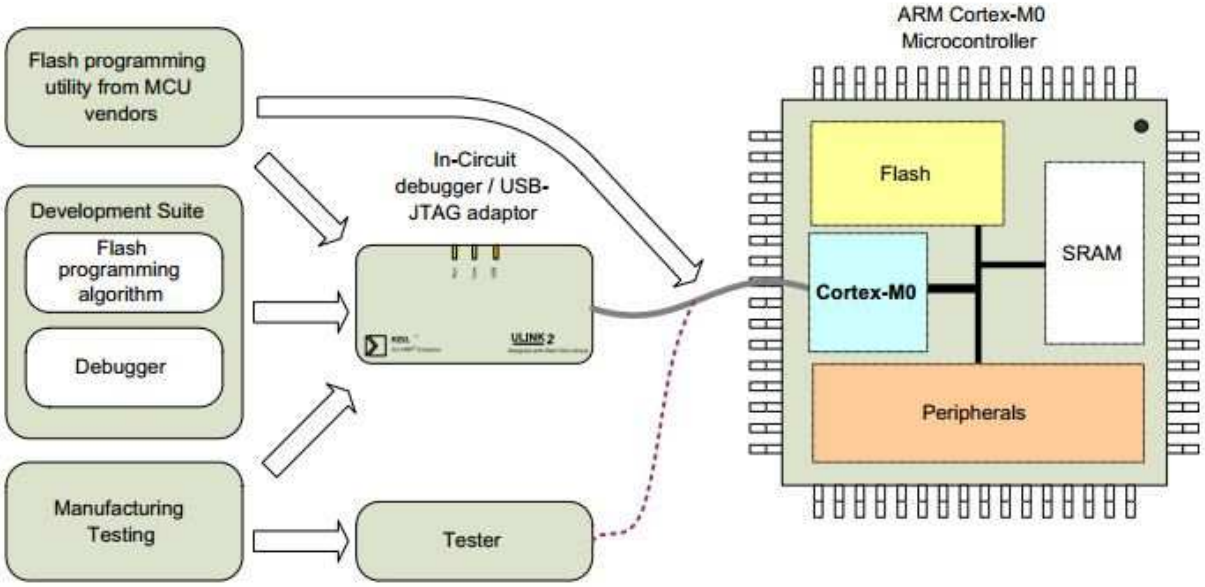


Figure 3.6 : Exemple d'interface de débogage pour l'ARM Cortex-M [17]

Pour les codes de programmes simples, on peut également tester le programme en utilisant un simulateur. Cela permet d'avoir une visibilité complète à la séquence d'exécution du programme et permet de tester sans matériel réel. Certaines suites de développement fournissent des simulateurs qui peuvent également imiter le comportement périphérique.
Dans notre cas, nous n'allons pas utiliser un simulateur étant donné que notre microcontrôleur est fabriqué par *STMicroelectronics*, donc nous allons utiliser un débogueur *ST-LINK V2* de *STMicroelectronics* comme le montre la fig 3.7.

Figure 3.7 : Débogueur *ST-LINK V2*

iv. Types de données du langage C

Le langage C prend en charge un certain nombre de types de données "standard". Cependant, la mise en œuvre de type de données peut dépendre de l'architecture du processeur et du compilateur C. Dans les processeurs ARM dont le Cortex-M3, les

implémentations de type de données figurant dans le tab XIII sont prises en charge par tous les compilateurs C.

Tableau XIII: Types de données prise en charge par le ARM Cortex-M [17]

C and C99 (stdint.h) Data Type	Number of Bits	Range (Signed)	Range (Unsigned)
char, int8_t, uint8_t	8	−128 to 127	0 to 255
short int16_t, uint16_t	16	−32768 to 32767	0 to 65535
int, int32_t, uint32_t	32	−2147483648 to 2147483647	0 to 4294967295
long	32	−2147483648 to 2147483647	0 to 4294967295
long long, int64_t, uint64_t	64	− (2^63) to (2^63 − 1)	0 to (2^64 − 1)
float	32	$-3.4028234 \times 10^{38}$ to 3.4028234×10^{38}	
double	64	$-1.7976931348623157 \times 10^{308}$ to $1.7976931348623157 \times 10^{308}$	
long double	64	$-1.7976931348623157 \times 10^{308}$ to $1.7976931348623157 \times 10^{308}$	
pointers	32	0x0 to 0xFFFFFFFF	
enum	8/16/32	Smallest possible data type, except when overridden by compiler option	
bool (C++ only), _Bool (C only)	8	True or false	
wchar_t	16	0 to 65535	

Dans la programmation ARM, on peut aussi également se référer à la taille des données comme le mot, le demi-mot, et l'octet comme listée sur le tab XIV.

Tableau XIV : Définition de taille de données pour l'ARM Cortex-M [17]

Terms	Size
Byte	8-bit
Half word	16-bit
Word	32-bit
Double word	64-bit

III.3. L'API CMSIS

Le CMSIS (*Cortex Microcontroller Software Interface Standard*) est une couche d'abstraction matérielle indépendante et standard pour la série de processeurs Cortex-M. C'est un API (*Application Programming Interface*) qui définit les interfaces d'outils génériques.

L'objectif principal de CMSIS est d'améliorer la portabilité des logiciels et leur réutilisation dans différents microcontrôleurs. Cela permet aux logiciels provenant de différentes sources de s'intégrer harmonieusement ensemble.

La spécification de base CMSIS prend une petite quantité de ressources (environ 1 k de code et à seulement 4 octets de RAM) et standardise juste la façon dont on accède aux processeurs Cortex-M et aux registres du microcontrôleur. En outre, CMSIS n'affecte pas vraiment la façon dont on écrit les codes et ne force pas à adopter une méthodologie particulière. Il fournit simplement un *Framework* qui permet d'intégrer du code tiers et de réutiliser le code sur de futurs projets [18].

CMSIS se compose de plusieurs spécifications distinctes (CORE, DRIVERS, DSP, RTOS, SVD : *System View Description*, et DAP) qui rendent le code source plus portable entre les outils et les dispositifs. Il est structuré comme le montre la fig 3.8.

Les spécifications complètes CMSIS peuvent être téléchargées sur le site de CMSIS. Chacune de ces spécifications sont intégrées dans *Keil MDK-ARM*.

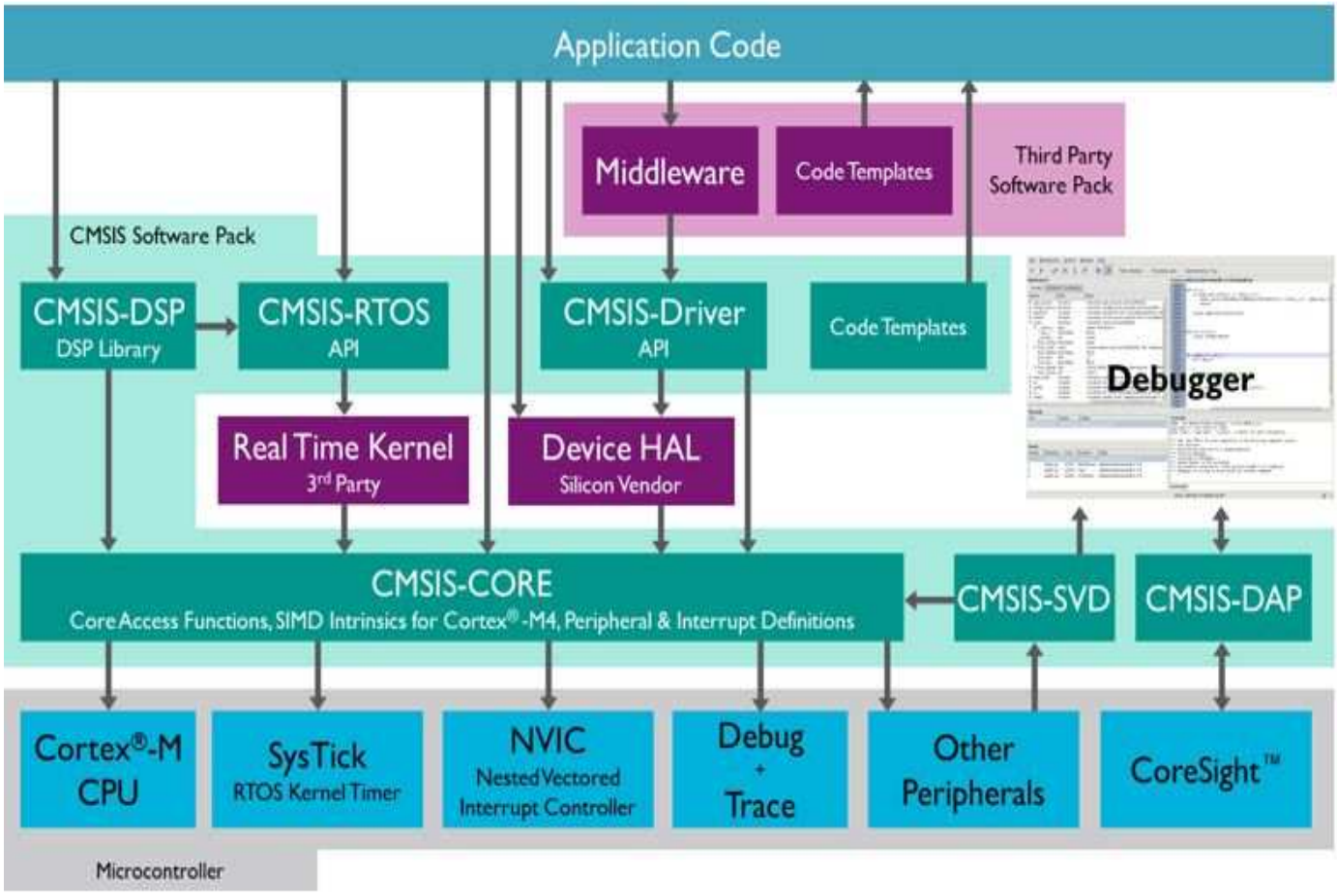


Figure 3.8 : Structure du CMSIS [18]

a) CMSIS-CORE

Le CMSIS-CORE met en œuvre le système d'exécution de base pour un dispositif Cortex-M et donne à l'utilisateur l'accès au cœur du processeur et aux périphériques de l'appareil. Il est défini par:

- Une couche d'abstraction matérielle (HAL : *Hardware Abstraction Layer*) pour les registres du processeur Cortex-M avec définitions normalisées pour le *sysTick*, le NVIC, les registres de bloc de contrôle du système, les registres de MPU, les registres FPU, et les fonctions d'accès de base.
- Des noms d'exceptions de système pour interfacer les exceptions sans avoir de problèmes de compatibilité.
- Des méthodes pour organiser les fichiers d'en-tête qui rend facile l'appréhension de nouveaux produits de microcontrôleurs Cortex-M et améliore la portabilité des logiciels. Cela comprend les conventions d'appellation pour les interruptions spécifiques à l'appareil.
- Des méthodes pour l'initialisation du système à être utilisées par chaque fournisseur MCU. Par exemple, la fonction normalisée *SystemInit ()* est essentielle pour la configuration du système d'horloge de l'appareil.
- Des fonctions intrinsèques utilisées pour générer des instructions CPU qui ne sont pas prises en charge par les fonctions standards C.
- Une variable pour déterminer la fréquence d'horloge de système qui simplifie la configuration de la minuterie *sysTick* [19].

Pour utiliser le CMSIS-CORE, les fichiers suivants sont ajoutés à l'application embarquée:

- Un fichier de démarrage « startup_ <device> .s » avec le gestionnaire de réinitialisation et la table de vecteurs d'exception.
- Des fichiers de configuration système « system _<device> .c » et « system _<device> .h » avec la configuration générale de l'appareil (c'est à dire pour l'horloge et la configuration de BUS).
- Un fichier d'en-tête de l'appareil <device>.h qui donne accès au cœur du processeur et à tous les périphériques.

La figure 3.9 illustre tous les fichiers nécessaires pour le CMSIS-CORE.

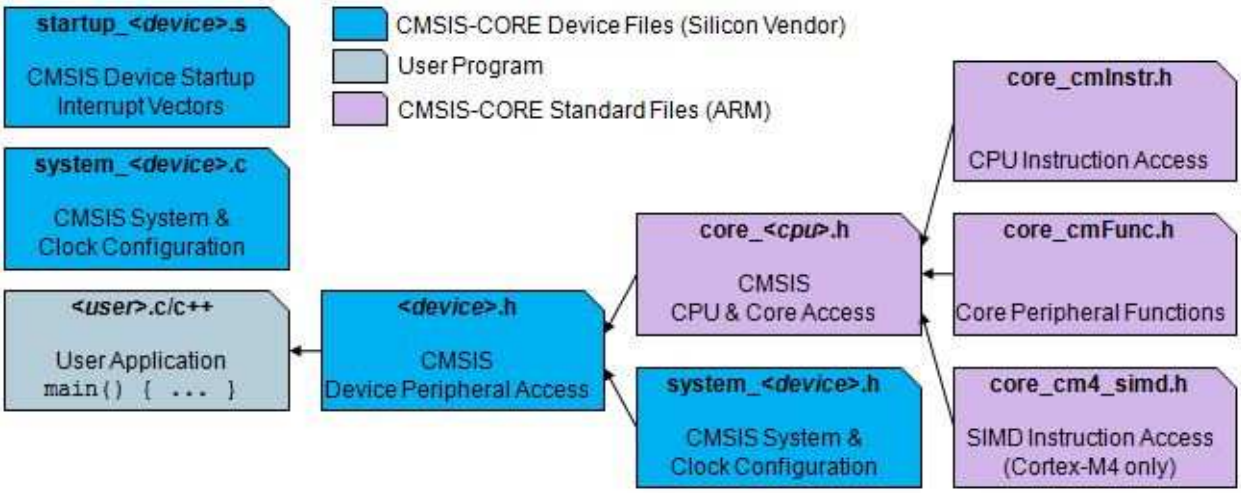


Figure 3.9 : Les fichiers du CMSIS-CORE [19]

La figure 3.10 donne un exemple de programme minimal typique (fichier main.c) pour débuter à programmer en langage C sur un ARM-Cortex :

```
#include <device.h>                // File name depends on device used

void SysTick_Handler (void) {    // SysTick Interrupt Handler
  ;                              // Add user code here
}

void TIM1_UP_IRQHandler (void) {     // Timer Interrupt Handler
  ;                                  // Add user code here
}

void timer1_init(int frequency) {      // Set up Timer (device specific)
  NVIC_SetPriority (TIM1_UP_IRQn, 1); // Set Timer priority
  NVIC_EnableIRQ (TIM1_UP_IRQn);      // Enable Timer Interrupt
}

// The processor clock is initialized by CMSIS startup + system file
void main (void) {                   // user application starts here
 if (SysTick_Config (SystemCoreClock / 1000)) { // SysTick 1mSec
   :                                            // Handle Error
  }
  timer1_init ();      // setup device-specific timer
  while (1)  { }      // Endless Loop (the Super-Loop)
}
```

Figure 3.10 : Exemple de programme minimal pour l'ARM Cortex-M

Un autre avantage de CMSIS-CORE n'est pas seulement de fournir une interface de configuration du système du processeur (interruptions, exceptions, horloge du cœur) mais aussi de fournir des fonctions de très bas niveau pour accéder au cœur du processeur comme des fonctions d'accès aux registres du processeur illustrées par la fig 3.11 et des fonctions propres pour les instructions du CPU et du SIMD.

```
x = __get_BASEPRI();     // Read BASEPRI register
x = __get_PRIMARK();     // Read PRIMASK register
x = __get_FAULTMASK();   // Read FAULTMASK register
__set_BASEPRI(x);        // Set new value for BASEPRI
__set_PRIMASK(x);      // Set new value for PRIMASK
__set_FAULTMASK(x);    // Set new value for FAULTMASK
__disable_irq();       // Set PRIMASK, disable IRQ
__enable_irq();        // Clear PRIMASK, enable IRQ
```

Figure 3.11 : Exemple de fonction C de bas niveau [9]

b) <u>CMSIS-DRIVERS</u>

La spécification CMSIS-DRIVERS est une API qui décrit les interfaces des pilotes de périphériques pour les piles middleware et les applications de l'utilisateur comme représentés sur la fig 3.12. L'API CMSIS-DRIVER est conçu pour être générique et indépendante d'un RTOS spécifique, le rendant réutilisable dans un large éventail de dispositifs de microcontrôleurs. Au fil du temps, l'API CMSIS-DRIVERS s'étend avec d'autres groupes pour couvrir de nouveaux cas d'utilisation. C'est comme le cas d'*HAL library* de *STMicroelectronics*. Ces packs additionnels peuvent contenir des interfaces spécifiques supplémentaires pouvant étendre les pilotes de périphériques standards couverts par cette spécification CMSIS-DRIVERS, par exemple la mémoire BUS, GPIO, ou DMA [20].

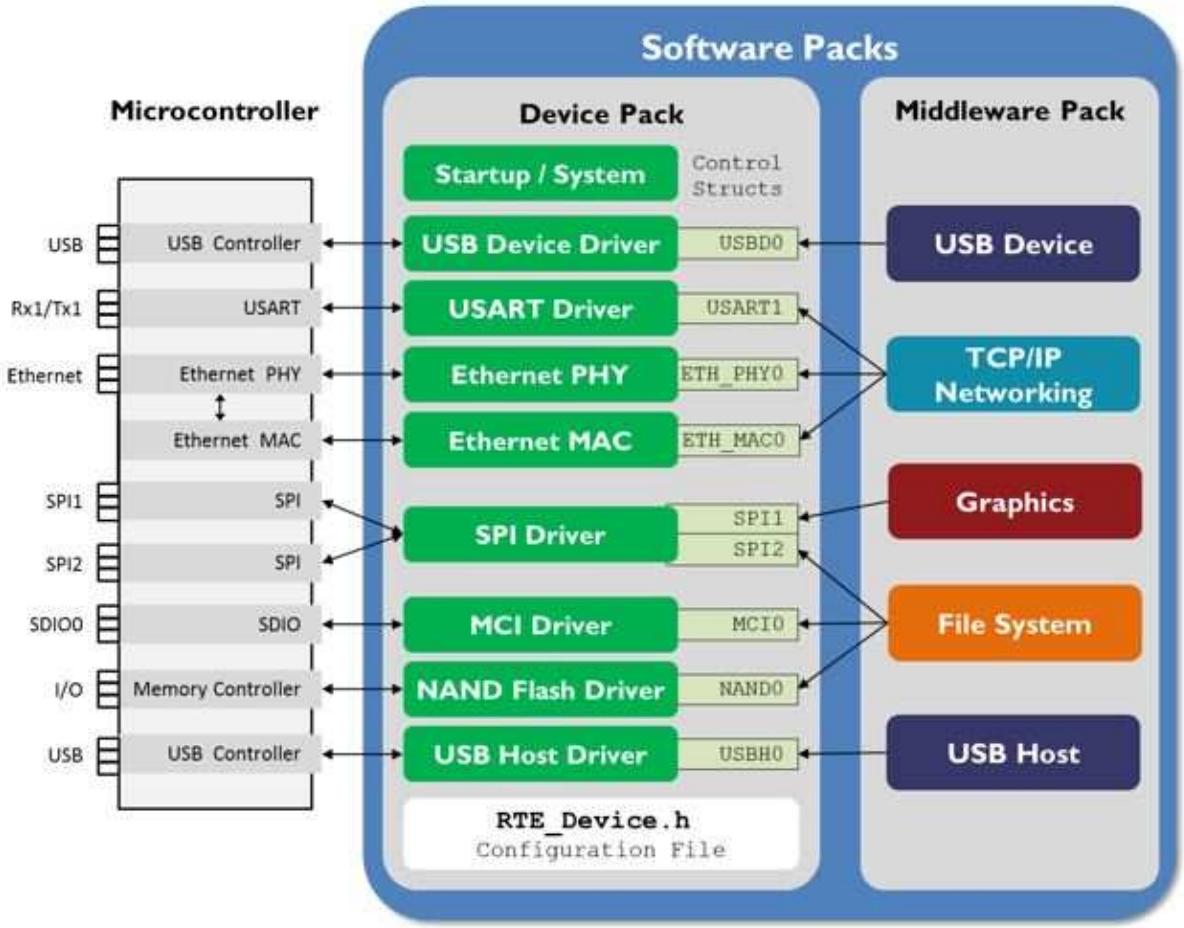


Figure 3.12 : L'interface CMSIS-DRIVER [20]

L'interface périphérique standard du driver permet de connecter les périphériques du microcontrôleur par exemple avec des middlewares qui implémentent des piles de communication (*stack lwIP*), des systèmes de fichiers, ou des interfaces utilisateurs graphiques. Chaque interface de pilote de périphérique peut fournir plusieurs instances reflétant les multiples interfaces physiques du même type dans un dispositif.

Pour la configuration de ces drivers, CMSIS met à disposition le fichier RTE_device.h. On peut le configurer manuellement avec un accès direct à partir de code C mais Keil MDK-ARM possède une « *configuration wizard* », comme le montre la fig 3.13, qui permet de faciliter la configuration des librairies de ces interfaces drivers. La « *configuration wizard* » inclut aussi plusieurs configurations comme celle de l'horloge, le RTOS… Elle permet d'éviter de toucher directement au code C des systèmes ou des librairies lorsqu'on personnalise la configuration.

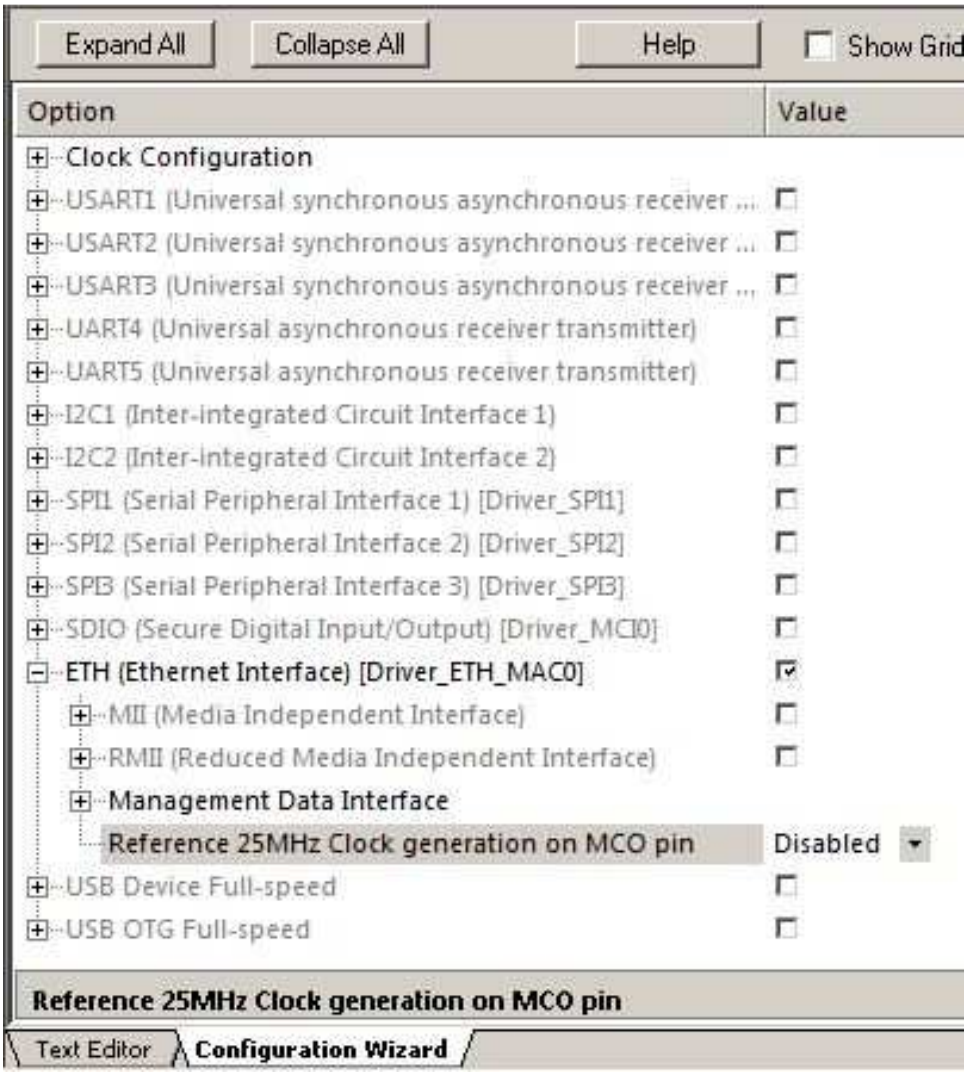


Figure 3.13 : *Configuration wizard*

c) CMSIS-DSP

Le CMSIS-DSP est une bibliothèque de logiciel qui rassemble une suite de fonctions de traitement de signal commun pour une utilisation sur les appareils basés sur les processeurs Cortex-M. La bibliothèque est divisée en un certain nombre de fonctions couvrant chacune une catégorie spécifique comme décrite sur la fig 3.14.

Basic math functions	**Matrix functions**
Vector multiplication	Matrix initialization
Vector subtraction	Matrix addition
Vector addition	Matrix subtraction
Vector scale	Matrix multiplication
Vector shift	Matrix inverse
Vector offset	Matrix transpose
Vector negate	Matrix scale
Vector absolute	**Transforms**
Vector dot product	Complex FFT functions
Fast math functions	Real FFT functions
Cosine	DCT type IV functions
Sine	**Controller functions**
Square root of number	**Sine cosine**
Complex math functions	**PID**
Complex conjugate	Vector park transform
Complex dot product	Vector inverse park transform
Complex magnitude	Vector Clarke transform
Complex magnitude squared	Vector inverse Clarke transform
Complex by complex multiplication	**Statistical functions**
Complex by real multiplication	Power
Filters	Root mean square
Convolution	Standard deviation
Partial convolution	Variance
Correlation	Maximum
FIR filter	Minimum
FIR decemation	Mean
FIR lattice filter	**Support functions**
Infinite impulse response lattice filter	Vector copy
FIR sparse filter	Vector fill
FIR filter interpolation	Convert 8-bit integer value
Biquad cascade IIR filter using direct form I structure	Convert 16-bit integer value
Biquad cascade IIR filter 32 × 64 using direct form I structure	Convert 32-bit integer value
Biquad cascade IIR filter using direct form II transposed structure	Convert 32-bit FPU
Least mean squares FIR filter	**Interpolation functions**
Least mean squares normalized FIR filter	Linear interpolation function
	Bilinear interpolation function

Figure 3.14 : Librairie CMSIS-DSP [21]

Nous allons illustrer un exemple d'utilisation de la librairie DSP pour un contrôleur PID (Proportionnelle Intégrale Dérivée). Pour accéder à la bibliothèque, il faut ajouter son fichier d'en-tête : #include "arm_math.h"

Nous devons aussi ajouter un #define pour configurer le fichier d'en-tête de DSP pour le processeur que nous utilisons. Les « *defines* » sont:

```
ARM_MATH_CM4      //Cortex-M4
ARM_MATH_CM3      //Cortex-M3
ARM_MATH_CM0     // Cortex-M0
```

Dans cet exemple, nous allons utiliser l'algorithme PID. Toutes les principales fonctions de la bibliothèque DSP ont deux appels de fonction: une fonction d'initialisation et une fonction de processus comme le montre la fig 3.15:

```
void arm_pid_init_f32 (arm_pid_instance_f32 *S,int32_t resetStateFlag)
__STATIC_INLINE void arm_pid_f32 (arm_pid_instance_f32 *s, float32_t in)
```

Figure 3.15: Les fonctions essentielles pour l'algorithme PID

La fonction d'initialisation est transmise à une structure de configuration qui est propre à l'algorithme. La structure, comme le montre la fig 3.16, détient les constantes pour l'algorithme, et des tableaux de mémoire d'état :

```
typedef struct
{
  float32_t A0; /**,The derived gain, A0=Kp+Ki+Kd. */
  float32_t A1; /**,The derived gain, A1=-Kp - 2 Kd. */
  float32_t A2; /**,The derived gain, A2=Kd. */
  float32_t state[3]; /**,The state array of length 3. */
  float32_t Kp; /**,The proportional gain. */
  float32_t Ki; /**,The integral gain. */
  float32_t Kd; /**,The derivative gain. */
} arm_pid_instance_f32;
```

Figure 3.16 : Déclaration de la structure du PID

La structure permet de définir des valeurs pour les coefficients du PID. Elle comprend également des variables provenant des gains A0, A1, A2 et ainsi qu'un petit tableau qui maintient les variables d'état locales. La figure 3.17 montre un exemple de PID.

```
int32_t main(void)
 {
    arm_pid_instance_f32 S;        //structure declaration
    int i; S.Kp=1; S.Ki=1; S.Kd=1; //coefficients
    setPoint=10;                   //reference
    arm_pid_init_f32 (&S,0);       //pid initialization
    while(1)
    {
       error=setPoint-motorOut;    // error
       motorIn5arm_pid_f32 (&S,error); // pid calculation
       motorOut=transferFunction(motorIn,time); /* pid simulated onto a transfer function
                                                   but we could output it in a real system*/
       time+=1;
       for(i=0;i<100000;i++);
       }
 }
```

Figure 3.17 : Exemple de programme principale du PID

d) CMSIS-RTOS

L'API CMSIS-RTOS est une interface de système d'exploitation temps réel générique pour les dispositifs à base de processeur ARM Cortex-M. CMSIS-RTOS fournit une API normalisée pour un RTOS et donne donc plusieurs avantages aux utilisateurs et aux industries de logiciel [22]. Le *Keil RTX RTOS* a été le premier RTOS à utiliser l'API CMSIS-RTOS et il est devenu comme une implémentation de référence open source. RTX peut être compilé avec GNU et avec le compilateur d'IAR. Typiquement, le *Keil RTX RTOS* nécessitera 500 octets de RAM et 5 KB de code [8]. Nous avons maintenant aujourd'hui une génération de petits microcontrôleurs à faible coût qui ont assez de puissance, de mémoire et de traitement pour soutenir l'utilisation d'un RTOS.

Le CMSIS-RTOS est généralement fourni comme une bibliothèque. Pour ajouter la fonctionnalité RTOS à une application basée sur CMSIS existants, la bibliothèque RTOS (et généralement avec un fichier de configuration) doit être ajoutée. La fonctionnalité disponible de la bibliothèque RTOS est définie dans le fichier d'en-tête « cmsis_os.h » qui est spécifique pour chaque application CMSIS-RTOS. Donc, pour permettre au code C d'accéder à l'API CMSIS RTOS, il faut ajouter cette inclusion aux fichiers d'application. La structure du fichier CMSIS-RTOS est présentée sur la fig 3.18.

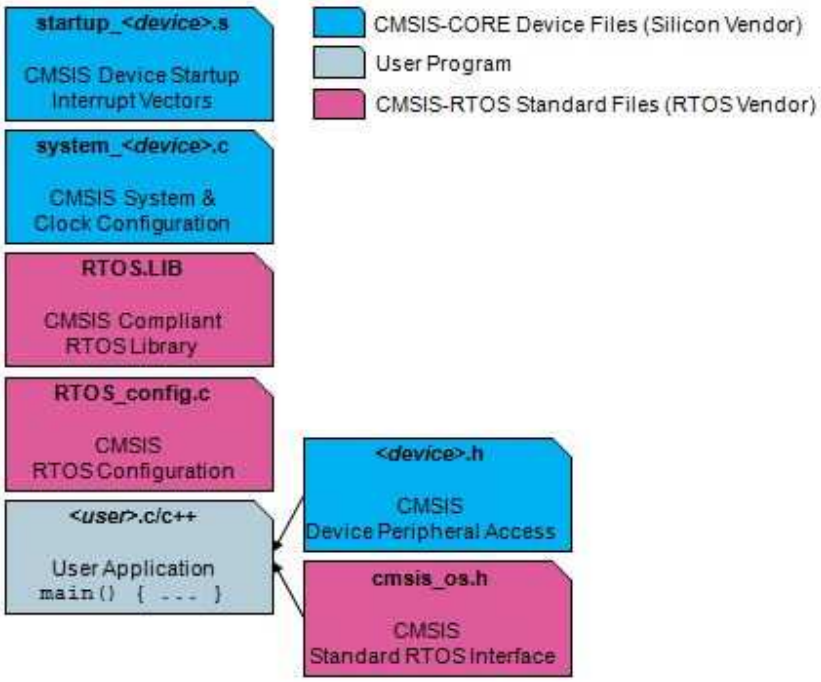


Figure 3.18 : Les fichiers du CMSIS-RTOS [22]

Selon la mise en œuvre du CMSIS-RTOS, l'exécution peut commencer avec la fonction principale comme le premier *Thread.* Ceci a l'avantage qu'un programmeur d'application peut utiliser d'autres bibliothèques middleware qui créent des *Thread* en interne. Cependant, la partie restante de l'application utilisateur utilise simplement le *Thread* principal. Par conséquent, l'utilisation du RTOS peut être invisible pour le programmeur d'application, mais les bibliothèques peuvent utiliser des fonctionnalités CMSIS-RTOS.

Une fois les fichiers ajoutés à un projet, l'utilisateur peut commencer à travailler avec les fonctions CMSIS-RTOS. Pour utiliser le *Keil RTX RTOS* sur *Keil MDK-ARM*, il faut laisser le débogueur savoir qu'on utilise le RTOS afin qu'il puisse fournir un soutien de débogage supplémentaire. Cela se fait en sélectionnant *RTX Kernel* dans les Options pour le menu *Target*. Un exemple de code minimal pour la mise en place d'un RTOS est présenté sur la fig 3.19.

```
#include "cmsis_os.h"                          // CMSIS-RTOS header file

void job1 (void const *argument)  {            // thread function 'job1'
  while (1)  {
    :                                          // execute some code
    osDelay (10);                              // delay execution for 10 milliseconds
  }
}
osThreadDef(job1, osPriorityAboveNormal, 1, 0); // define job1 as thread function

void job2 (void const *argument)  {            // thread function 'job2'
  osThreadCreate(osThread(job1),NULL);         // create job1 thread
  while (1)  {
    :                                          // execute some code
  }
}
osThreadDef(job2, osPriorityNormal, 1, 0);     // define job2 as thread function

int main (void) {                              // program execution starts here
  osKernelInitialize ();                       // initialize RTOS kernel
    :                                          // setup and initialize peripherals
  osThreadCreate (osThread(job2));
  osKernelStart ();                            // start kernel with job2 execution
}
```

Figure 3.19 : Exemple de programme minimal pour le RTOS *RTX*

L'API CMSIS-RTOS fournit les attributs et les fonctionnalités suivantes :

- La gestion des *Thread* qui permet de définir, créer et contrôler les *Thread*.
- L'*Interrupt Service Routine* (ISR) qui peut appeler de nombreuses fonctions CMSIS-RTOS.
- Trois différents types d'événements de *Thread* qui peuvent supporter la communication entre plusieurs *Thread* et / ou ISR:

- Signal: c'est un *flag* qui peut être utilisé pour indiquer les conditions spécifiques à un *Thread*. Les signaux peuvent être modifiés dans un ISR ou d'un ensemble d'autres threads.
- Message: c'est une valeur 32 bits qui peut être envoyée à un *Thread* ou un ISR. Les messages sont enregistrés dans une file d'attente.
- Mail: c'est un bloc de mémoire de taille fixe qui peut être envoyé à un *Thread* ou à un ISR. Les mails sont enregistrés dans une file d'attente et l'allocation de mémoire est fournie.

- La gestion de Mutex et la gestion de Sémaphore.
- Le temps dans le CPU qui peut être géré avec les fonctionnalités suivantes:
  - Un paramètre de délai d'attente qui est incorporé dans de nombreuses fonctions CMSIS-RTOS pour éviter le blocage du système. Quand un délai d'attente est spécifié, le système attend jusqu'à ce qu'une ressource soit disponible ou un événement se produise. En attendant, d'autres *Thread* sont planifiés.
  - La fonction *osDelay* qui met un *Thread* dans l'état d'attente pour une période de temps spécifiée.
  - La fonction générique d'attente *osWait* pour des événements qui sont affectés à un Thread.
  - La fonction *osThreadYield* qui fournit une commutation entre T*hread* et passe l'exécution à l'autre *Thread* de la même priorité [22].

e) CMSIS-SVD ET CMSIS-DAP

La spécification CMSIS-SVD définit un fichier de description de visualisation du système (SVD : *System Viewer Description*). Ce fichier est fourni par le fournisseur de la puce et contient une description complète des registres périphériques de microcontrôleur dans un format XML (*eXtensive Markup Language*)[6]. Ce fichier est ensuite importé par l'outil de développement qui l'utilise pour construire automatiquement les fenêtres de débogage périphériques pour le microcontrôleur. Dans *Keil MDK-ARM*, on peut avoir accès à la visualisation du système pendant le débogage dans l'onglet *Peripherals*.

La spécification CMSIS-DAP définit le protocole d'interface d'une unité de débogage du matériel qui se trouve entre l'ordinateur hôte et le port d'accès de débogage (DAP) du microcontrôleur. Cela permet à tout ensemble d'outils logiciels qui supporte CMSIS-DAP de se connecter à n'importe quelle unité de débogage matériel qui prend également en charge CMSIS-DAP. *Keil MDK-ARM* supporte le CMSIS-DAP.

III.4. INTERRUPTION

a) Configuration de table de vecteur

Lorsque le processeur Cortex-M accepte une demande d'exception, le processeur a besoin de déterminer l'adresse de départ du gestionnaire d'exception (ou l'ISR si l'exception est une interruption). Ces informations sont stockées dans la table de vecteur dans la mémoire. Par défaut, la table de vecteur commence à l'adresse de mémoire 0, et l'adresse de vecteur est agencée en fonction du nombre d'exception multiplié par 4. La table de vecteur est normalement définie dans les codes de démarrage fournis par les fournisseurs de microcontrôleurs.

Habituellement, l'adresse de départ (0x00000000) devrait être la mémoire de démarrage, et elle sera généralement soit des dispositifs de mémoire *flash* ou ROM, et la valeur ne peut pas être modifiée au moment de l'exécution. Cependant, dans certaines applications, il est utile de pouvoir modifier ou définir des vecteurs d'exception au moment de l'exécution. Pour gérer cela, les processeurs Cortex-M3 et Cortex-M4 supportent une fonctionnalité de transfert de table de vecteur. Avant que la table de vecteur ne soit transférée, on peut copier le contenu de la table de vecteur existant vers un nouvel emplacement de table de vecteur. Cela comprend les adresses de vecteurs pour les gestionnaires d'erreurs, le NMI, les appels système... Après que les contenus de la table de vecteur nécessaires soient mis en place et que la table de vecteur soit relocalisée, on peut ajouter de nouveaux vecteurs à la table de vecteur. La fonctionnalité de transfert de table de vecteur fournit un registre programmable appelé VTOR (*Vector Table Offset Register).* Ce registre définit l'adresse de début de la mémoire utilisée en tant que table de vecteur. Le registre VTOR peut être consulté par « SCB-> VTOR» [8].

La figure 3.20 donne un exemple de transfert de table de vecteur :

```
// HW_REG is a macro to convert address value to pointer
#define HW_REG(addr) (*((volatile unsigned long *)(addr)))
#define NEW_VECT_TABLE 0x20008000 // An SRAM region for vector table
     NVIC_SetPriorityGrouping(5);
     ...
     HW_REG((NEW_VECT_TABLE +0x8)) = HW_REG(0x8); // Copy NMI vector
     HW_REG((NEW_VECT_TABLE +0xC)) = HW_REG(0xC); // Copy HardFault
     ...
     SCB->VTOR = NEW_VECT_TABLE; // Relocate vector table to SRAM
     ...
     HW_REG(4*(7+16)) = (unsigned) IRQ7_Handler; // Setup vector
```

Figure 3.20 : Exemple de code de transfert de table de vecteur [9]

b) Configuration de la priorité d'interruption

Dans les Cortex-M, si une exception est acceptée par le processeur, son gestionnaire s'exécute dépendamment de la priorité de l'exception. Certaines exceptions (*reset*, NMI, et *HardFault*) ont des niveaux de priorité fixée. Leurs niveaux de priorité sont représentés par des nombres négatifs pour indiquer qu'ils sont d'une priorité plus élevée que d'autres exceptions. Les autres exceptions ont des niveaux de priorité programmable, qui vont de 0 à 255. Par défaut, après une réinitialisation, toutes les exceptions avec une priorité programmable sont au niveau de priorité 0. Pour définir la priorité de demande d'interruption, par exemple à 0xC, on peut utiliser la fonction sur la fig 3.21

```
NVIC_SetPriority(IRQ4_IRQn, 0xC); // This function
// automatically shifts the priority value to implemented bits
// in the priority level registers
```

Figure 3.21 : Fonction de définition de la priorité de l'interruption [8]

La constante IRQ4_IRQn ci-dessus est juste un exemple d'identificateur d'interruption. Lors de l'utilisation des fonctions de contrôle d'interruption CMSIS, il est recommandé d'utiliser les identificateurs d'interruption définis dans le fichier d'en-tête device.h pour aider à la lisibilité et la portabilité. On peut utiliser la fonction *NVIC_SetPriority* avec une autre fonction CMSIS qui calcule la valeur de niveau de priorité basé sur la mise en priorité de préemption, de sous priorité et de la priorité de groupe comme l'illustre la fig 3.22.

```
NVIC_SetPriority(IRQ4_IRQn, NVIC_EncodePriority(PriorityGroup,
   PreemptPriority, SubPriority));
```

Figure 3.22 : Fonction de définition de la priorité selon divers paramètres [8]

c) Activation de l'interruption

A l'intérieur du NVIC, deux adresses de registres distincts sont utilisées pour activer et désactiver les interruptions. Cette dualité assure que chaque interruption peut être activée ou désactivée sans affecter ou perdre l'autre état d'activation d'interruption. Pour définir une activation, on doit écrire 1 à l'emplacement de bit dans les registres de SETEN dans le NVIC. De même, pour effacer une interruption, on doit écrire un 1 au bit correspondant dans le CLREN. Pour les utilisateurs de CMSIS, l'activation/désactivation peut être accédée par les fonctions "*NVIC_EnableIRQ*" et "*NVIC_DisableIRQ*".

La figure 3.23 est un exemple d'activation d'interruption pour l'UART (*Universal Asynchronous Receiver Transmitter*):

```
NVIC_EnableIRQ(UART1_IRQn);  // Enable UART#1 interrupt
                             // UART1_IRQn is MCU specific and is defined
                             // in the device driver library

NVIC_DisableIRQ(UART1_IRQn); // Disable UART#1 interrupt
```

Figure 3.23 : Fonction de d'activation/désactivation de l'interruption pour l'UART [9]

d) Gestionnaire d'interruption

Dans le Cortex-M3, les gestionnaires d'interruption peuvent être programmés entièrement en C. La figure 3.24 est un exemple de gestionnaire de l'UART :

```
void UART1_Handler(void) {
  ... // processing task for the peripheral
  return;
}
```

Figure 3.24 : Gestionnaire d'interruption pour l'UART [8]

Pour les utilisateurs de CMSIS, le nom du gestionnaire d'interruption doit correspondre au nom de gestionnaire d'interruption définie par le fournisseur du microcontrôleur pour veiller à ce que le vecteur soit mis en place correctement dans la table de vecteur. On peut trouver le nom de la fonction de gestionnaire dans le tableau de vecteur à l'intérieur du code de démarrage (*startup code*).

Pour les utilisateurs du compilateur d'*ARM RealView* ou du kit *Keil MDK-ARM*, on peut ajouter le mot-clé __irq comme le montre la fig 3.25.

```
__irq void UART1_Handler(void) {
  ... // process IRQ request for the peripheral
  ... // Deassert IRQ request in peripheral
  return;
}
```

Figure 3.25 : Gestionnaire d'interruption pour l'UART pour les utilisateurs de *MDK-ARM* [8]

CHAPITRE IV : MISE EN PLACE D'UN RTOS POUR LES TRAITEMENTS DE DONNEES EN TEMPS REEL

IV.1. DESCRIPTION DU TRAVAIL ET DES MATERIELS

Le but du travail est d'implémenter un petit serveur sur un microcontrôleur ARM afin :

- d'acquérir des données via des capteurs de température et de vitesse, et ensuite de les envoyer via internet sur l'ordinateur client qui sera l'interface affichant les données.
- de construire un analyseur de spectre et ainsi d'envoyer le résultat des données via internet sur l'ordinateur client qui sera l'interface affichant les données.
- de commander un moteur de manière asservi en implémentant un correcteur numérique sur le microcontrôleur.
- de générer un signal numérique test pour l'analyseur de spectre.

Toutes ces tâches sont implémentées sur un RTOS pour bénéficier du temps réel. Le but de véhiculer les données via internet est de pouvoir avoir accès à ces données et à la commande du moteur même si la distance qui sépare le serveur (microcontrôleur) du client est considérable. Il suffit d'avoir une connexion internet et de se connecter au serveur.

a) Les tâches effectuées par le microcontrôleur

Pour bénéficier de la puissance du microcontrôleur, nous allons mettre en place un RTOS sur le microcontrôleur. La présence du RTOS permettra l'exécution des tâches simultanément et en temps réel. Les tâches sont choisies de manière à exploiter la puissance du microcontrôleur. Elles sont:

- Le calcul du correcteur numérique PID (Proportionnel Intégral Dérivé) qui assurera l'asservissement en vitesse d'un moteur courant continu.
- La génération de signal PWM pour la commande du moteur courant continu.
- L'acquisition de données via un encodeur rotatif en quadrature qui est utilisé ici comme un capteur de vitesse du moteur. L'encodeur permet la boucle de retour lors de la régulation de la vitesse du moteur.
- L'acquisition de données via un capteur de température numérique.
- La génération de signal sinusoïdale test pour la FFT.
- Le calcul de la FFT (*Fast Fourier Transform*) pour l'analyseur de spectre en temps réel.
- La gestion des piles de protocoles TCP (*Transmission Control Protocol*)/IP (*Internet Protocol).*

- La création d'un socket serveur TCP/UDP (*User Datagram Protocol*) pour l'envoi des données via internet.
- Le pilotage d'un module PHY pour l'ETHERNET.

Parmi ces tâches, il y en aura qui s'exécutera de manière séquentielle mais pas forcément de façon simultanée, par exemple le calcul du correcteur numérique PID et l'acquisition de données via l'encodeur, étant donné que le correcteur a besoin des valeurs venant du capteur.

b) Description des matériels utilisés

Comme matériels, on aura besoin d' :

- Un microcontrôleur STM32F407ZGT6 ARM Cortex-M4F de *STMicroelectronics*
- Un module PHY KS8721BL de *Micrel*.
- Un capteur de température numérique DS18B20 de *DALLAS.*
- Un moteur courant continu avec réducteur.
- Un encodeur rotatif en quadrature.
- Un Full-bridge driver L298N de *STMicroelectronics*.

i. Microcontrôleur STM32F407ZGT6

Figure 4.1 : Le microcontrôleur STM32F407ZGT6

Le microcontrôleur STM32F407ZGT6 représenté sur la fig 4.1 est basée sur le processeur RISC à haute performance ARM Cortex-M4F 32-bit fonctionnant à une fréquence de 168 MHz au maximum. Le noyau Cortex-M4 dispose d'une unité de virgule flottante (FPU) simple précision qui prend en charge toutes instructions de traitement de données ARM. Il met également en œuvre un ensemble complet d'instructions DSP et une unité de protection de la mémoire (MPU) qui améliore la sécurité des applications.

ii. PHY KS8721BL

Figure 4.2 : L'Ethernet PHY KS8721BL

L'Ethernet PHY KS8721BL représenté sur la fig 4.2 est un élément qui fonctionne à la couche physique du modèle de réseau OSI. C'est un module émetteur-récepteur des normes 10BASE-T/100BASE-TX/FX qui utilise les interfaces MII (*Media-Independent Interface*) et RMII (*Reduced MII*) pour transmettre et recevoir les données. Les détails se trouvent en annexe.
Ce module est soudé sur un même platine d'*Olimex* (voir annexe 3).

iii. Capteur de température DS18B20

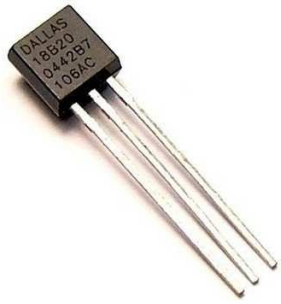


Figure 4.3 : Capteur de température DS18B20

Le DS18B20 représenté sur la fig 4.3 est un thermomètre numérique qui possède une résolution de 9 à 12 bits centigrades et une alarme interne programmable. Il communique via un seul bus appelé « *one-Wire* » qui par définition ne requiert qu'une seule ligne de donnée pour la communication avec le processeur. Il dispose d'une plage de températures de fonctionnement de -55 ° C à + 125 ° C. Les informations spécifiques se trouvent en annexe 3.

iv. Moteur à courant continu avec encodeur

Figure 4.4 : DC moteur avec réducteur

Le motoréducteur représenté sur la fig 4.4 est un moteur à courant continu 6 à 12 V avec réducteur de rapport 1:30 délivrant ainsi un couple 8kg.cm à 12 V. Sa vitesse de rotation est de 350 rpm à 12 V et sa consommation à vide est de 350 mA à 12 V. Elle possède un encodeur rotatif quadrature intégré avec 64 impulsions/révolution rattaché à l'arbre du moteur à l'arrière comme le montre la fig 4.5.

Figure 4.5 : Encodeur rotatif en quadrature

IV.2. IMPLEMENTATION DU PROTOCOLE TCP/IP

Si on veut mettre en place un réseau de connexion internet (TCP/IP), il est nécessaire de suivre le modèle OSI représenté sur la fig 4.6.

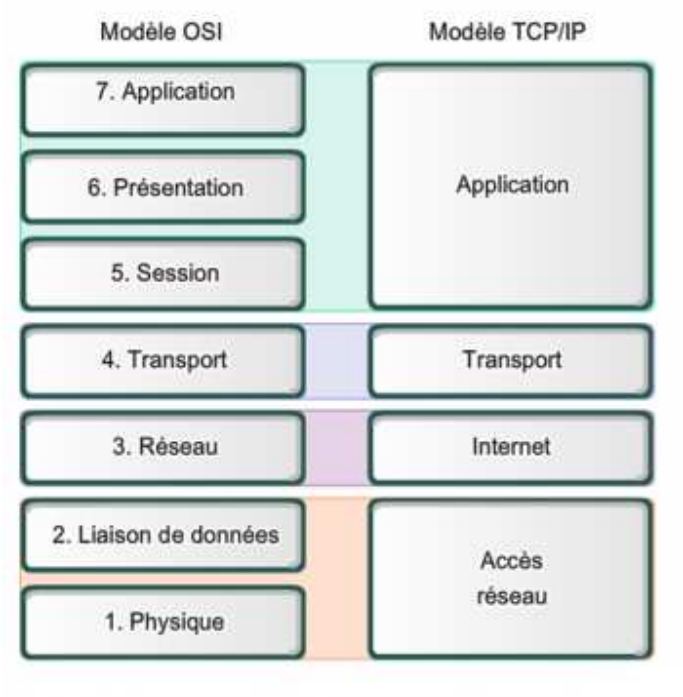


Figure 4.6 : Modèle OSI et modèle TCP/IP

Les deux premières couches (physique et liaison de données) du modèle OSI (*Open Systems Interconnection*) sont prises en main par des matériaux (*hardware* ; la couche physique par le PHY KS8721BL et la couche de liaison de données par *10/100 Fast Ethernet MAC* (*Media Access Control*). Ce dernier est un module intégré à l'intérieur du microcontrôleur STM32F407ZGT6. Ces modules ont besoin d'être programmés, il faudra donc créer des interfaces de pilote pour pouvoir les faire fonctionner. Comme on a vu dans le chapitre précédent, CMSIS-DRIVER fournit un API pour développer une interface standard pour le pilote du MAC intégré dans le processeur. Tous les outils nécessaires se trouvent à l'intérieur des bibliothèques STM32F4x7_eth.c, STM32F4x7_eth.h, STM32F4x7_conf_eth.h avec le fichier de configuration RTE_device.h. Pour le contrôle du PHY, la puce possède un module intégré DMA (*Direct Memory Access*) pour l'Ethernet qui sera utilisé pour accéder aux registres et aux instructions à l'intérieur de la mémoire du PHY pour la création du driver. Les API de bas niveaux nécessaires pour programmer le DMA sont aussi fournies par le CMSIS-DRIVER.

La troisième et la quatrième couche du modèle OSI (Réseau) sont prises en charge par des logiciels (software). En effet, notre projet utilise la pile de protocole lwIP (*Lightweight TCP/IP stack*) qui est une implémentation du protocole TCP/IP. L'un des objectifs de l'implémentation de lwIP est de réduire l'utilisation des ressources tout en ayant un module TCP le plus complet possible [23]. Cela rend l'utilisation de lwIP parfaitement adaptée aux systèmes embarqués avec quelques dizaines de kilooctets de RAM disponibles et environ de la place pour 40 kilooctets de code en ROM [23]. Après la mise en place du réseau, on peut

implémenter la quatrième couche qui est la couche de transport, c'est-à-dire on peut créer des sockets TCP ou UDP pour le transport des données. Notre projet implémente un serveur qui accepte les demandes de connexion venant des clients via Internet. Et c'est là que le RTOS entre en jeu car pour pouvoir gérer plusieurs connexions il faut créer un thread comme le montre la fig 4.7, qui peut être instancié plusieurs fois et travailler simultanément.

```
void tcpecho_init(void)
{
  sys_thread_new("tcpecho_thread", tcpecho_thread, NULL, DEFAULT_THREAD_STACKSIZE, TCPECHO_THREAD_PRIO);
}
```

Figure 4.7 : Fonction de création de thread

C'est à l'intérieur de ce *thread* que le serveur TCP s'exécute et établit la connexion avec les clients. Le principe de fonctionnement du serveur est illustré par la fig 4.8.

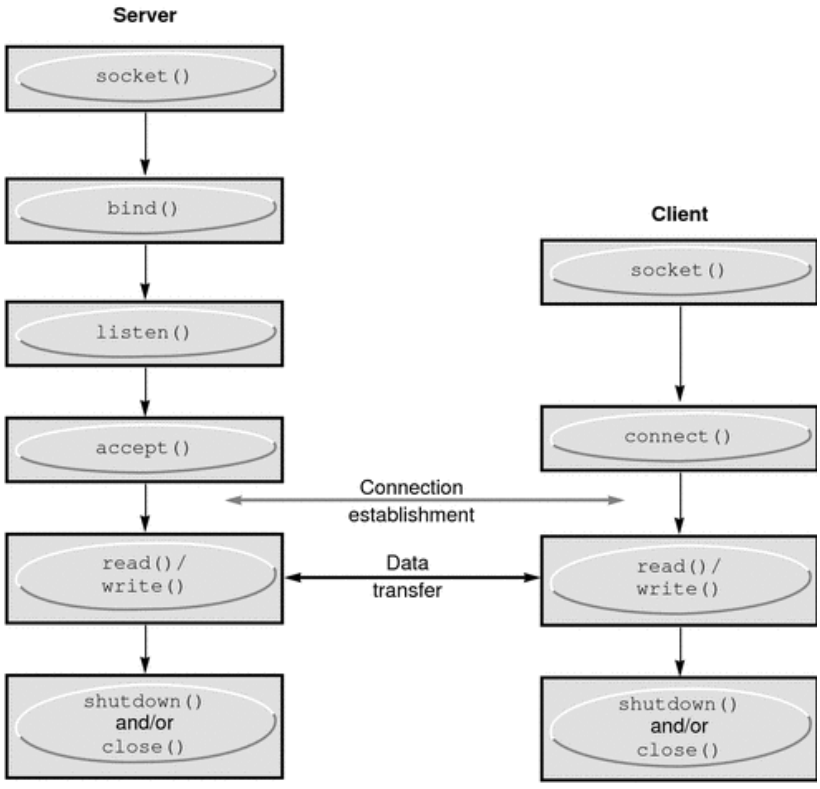


Figure 4.8 : Fonctionnement du serveur et du client [24]

IV.3. IMPLEMENTATION DU DSP

a) Analyseur de spectre numérique

Un analyseur de spectre est un outil qui permet de visualiser les différentes fréquences contenues dans un signal avec leur amplitude respective. Il est très utilisée en traitement de signal : allant des petits appareils comme les oscilloscopes jusqu'à des applications industrielles comme l'analyse de défaut mécanique par analyse vibratoire. L'outil mathématique utilisé pour le réaliser numériquement est la transformée de Fourier d'un signal à temps discret. Or, la transformée de Fourier d'un signal à temps discret n'est pas sous une forme appropriée pour être obtenue par un calculateur numérique. En effet un calculateur ne peut traiter que des nombres et de plus en quantité limitée par la taille de sa mémoire. A cause de la nécessité de la transformée de Fourier en traitement de signal, il est nécessaire de le mettre sous une forme pratiquement utilisable: la transformée de Fourier discret [25]. Elle a pour formule l'équation (4.1):

$$X_n = \sum_{k=0}^{N-1} x(k) e^{\frac{-i2\pi nk}{N}} \qquad (4.1)$$

Tel que Xn l'amplitude du spectre du signal, x(k) l'echantillon correspondant, N le nombre total d'echantillons.

La FFT est l'algorithme de calcul efficace et rapide de la transformée de Fourier discrète d'un signal.

L'implémentation de la FFT dans le microcontrôleur se fait par l'utilisation de l'API CMSIS-DSP. En effet, comme on a vu dans le chapitre précédent, la bibliothèque DSP fournit les deux fonctions essentielles pour calculer la FFT. Ce sont:

```
arm_rfft_init_f32(&RFFT, &CFFT,  L, 0,1);
arm_rfft_f32(&RFFT, inSignalF32,outSignalF32);
```

Les variables RFFT et CFFT sont respectivement les instances des structures arm_cfft_radix4_instance_f32 et de arm_rfft_instance_f32.
L est le nombre d'échantillons du signal stocké dans le tableau de float inSignalF32. Le tableau de float outSignalF32 stocke le résultat de l'opération FFT. De plus pour calculer l'amplitude du résultat de la FFT, il est nécessaire d'appeler la fonction :

```
arm_cmplx_mag_f32 (outSignalF32, outSignalF32, L);
```

Dans notre cas, nous allons simuler un signal sinusoïdal à une fréquence de 770Hz avec une fréquence d'échantillonnage de 8000Hz, c'est-à-dire $\sin\left(\frac{2\pi fk}{fe}\right)$ avec f la fréquence du signal et fe la fréquence d'échantillonnage. Ce choix respecte bien le théorème de Shannon sur l'échantillonnage d'un signal pour éviter le repliement de spectre. Voici le code pour générer un tableau de float qui stocke un signal sinusoïdal échantillonné :

```
for (k=0; k<L; k++)
    {
      inSignalF32 [k] = (float) 0.5f * sinf(2.0f * PI * f* k /fe);
     }
```

Avec :
f : la fréquence du signal
fe : la fréquence d'échantillonnage
sinf : la fonction sinus qui retourne des valeurs en format float
PI : la constante 3.14
L'amplitude du signal est de 0.5.

En principe, la transformée de Fourier d'un sinus fait apparaitre un pic sur la fréquence du signal comme le montre la fig 4.9.

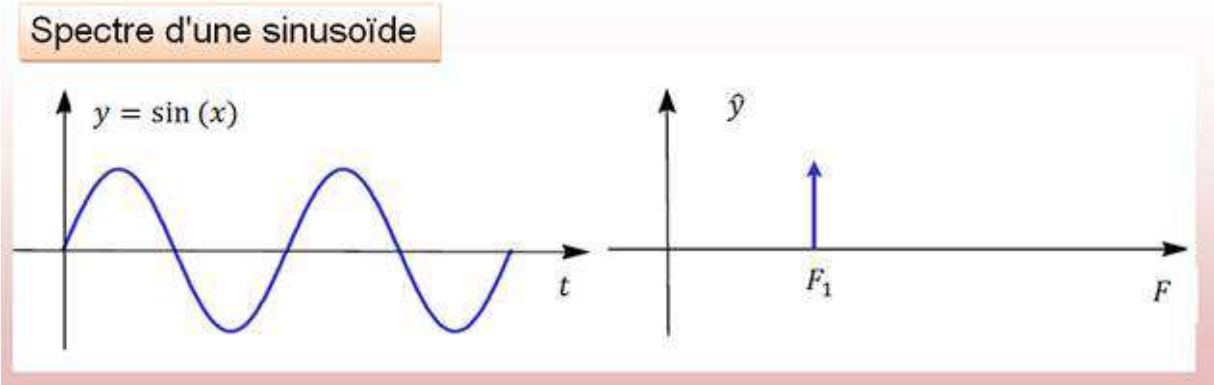


Figure 4.9 : Spectre d'une sinusoïde [26]

Cependant dans notre cas, le spectre sera un peu différent à cause de plusieurs raisons :

- L'échantillonnage du signal à fe rend périodique le spectre et le multiplie par 1/Te.
- Le fait de prendre un nombre limité d'échantillons (par exemple dans notre cas 128 échantillons) signifie une troncation du signal par une fenêtre de largeur To=L. Et cela a pour effet de convoluer le spectre avec un sinus cardinal qui s'annule tous les 1/To.

Cette troncation va donc changer l'allure du spectre car elle introduit des erreurs dues à des ondulations; des erreurs qu'on peut chercher à atténuer en choisissant une fenêtre de pondération. Dans notre cas, on va utiliser la fenêtre de Hanning décrite par l'équation (4.2)

$$h(k) = \begin{cases} 0{,}5\left(1 - \cos\left(\frac{2\pi k}{T_0}\right)\right) & si\ k\ \in [0; T_0] \\ 0\ ailleurs \end{cases} \qquad (4.2)$$

Le code C pour générer cette fonction est la suivante :

```
for (k=0; k<L; k++)
    {
      hanning_window_f32 [k] = (float) (0.5f *(1.0f - cosf(2.0f*PI*k / L)));
     }
```

Il suffit alors de multiplier les échantillons par le résultat de cette fonction avant de procéder à la transformée de Fourier. Pour cela il faut utiliser la fonction :

```
arm_mult_f32 (hanning_window_f32, inSignalF32, inSignalF32, L);
```

b) Asservissement en vitesse d'un moteur à courant continu

La régulation numérique de la vitesse d'un moteur à courant continu nécessite d'utiliser un correcteur. Dans notre cas, nous allons implémenter un correcteur PID numérique dans l'ARM Cortex-M4. Mais avant tout, il est nécessaire de calculer la fonction de transfert du moteur pour pouvoir faire l'étude de la conception sur MATLAB et ainsi de déduire et utiliser les coefficients du correcteur dans l'algorithme PID implémentée dans l'ARM Cortex-M. L'équation (4.3) montre la fonction de transfert du moteur à courant continu ; les détails des calculs de la fonction de transfert du moteur se trouvent en annexe 4.

$$H(p) = \frac{K_t}{JLp^2 + (JR + Lb)p + Rb + K_e K_t} \qquad (4.3)$$

Avec :

K_t : Constante de couple du moteur

J : Moment d'inertie de l'arbre du rotor

K_e : Constante de force électromotrice

R : Résistance interne du moteur

L : Inductance interne du moteur

b : Coefficient de frottement visqueux du moteur

On peut alors modéliser le système avec Simulink dans Matlab comme le montre la fig 4.10.

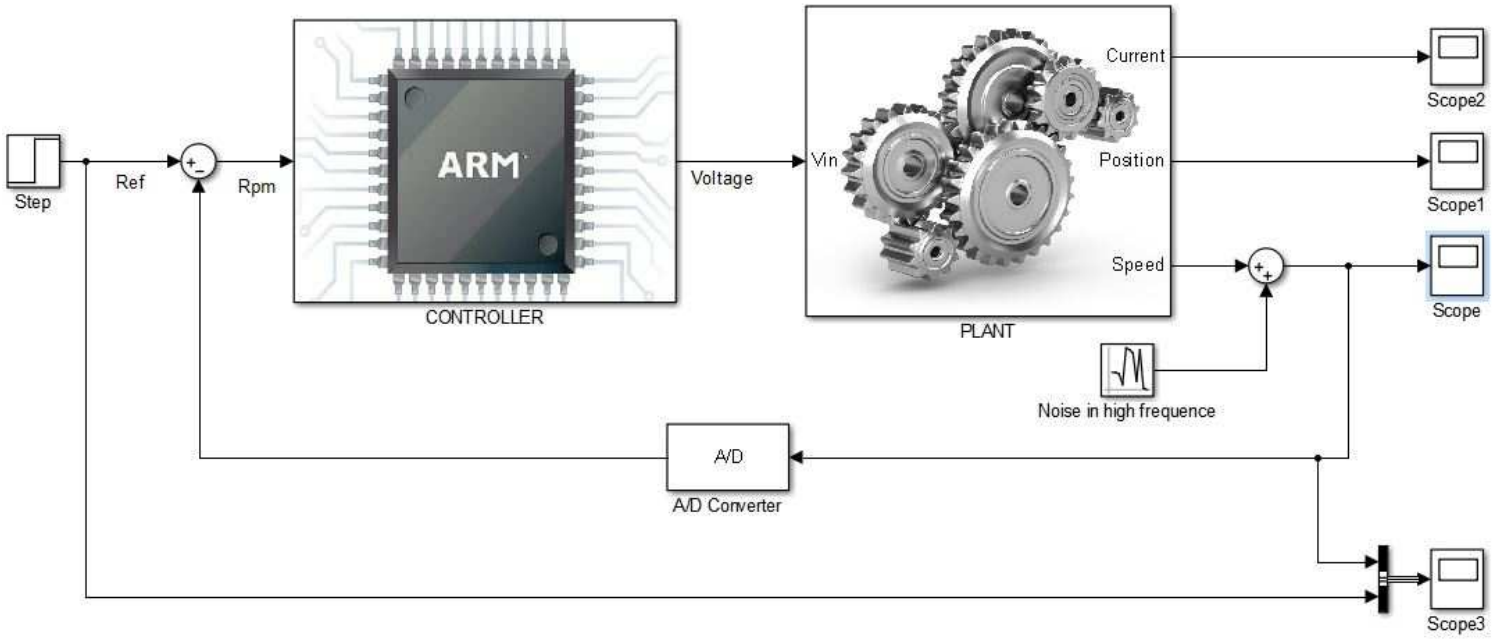


Figure 4.10 : Modélisation sur Matlab du control du moteur à courant continu

Dans le bloc CONTROLLER de la fig 4.10, il y a le correcteur PID numérique qui est implémenté dans le processeur.

Dans le bloc PLANT de la fig 4.10, il y a deux blocs comme représentés sur la fig 4.11.

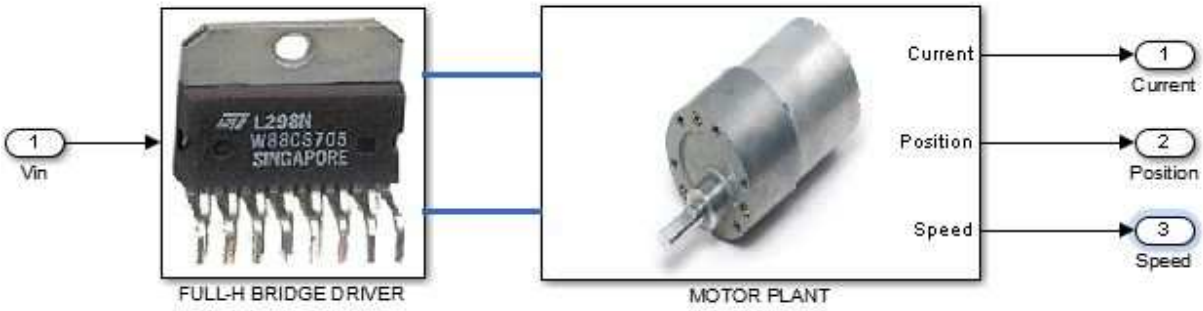


Figure 4.11 : Driver et moteur à courant continu

Le bloc “FULL-H BRIDGE DRIVER” de la fig 4.11 contient le modèle du driver du moteur et le modèle de la génération du signal PWM. Ces modèles sont représentés par la fig 4.12.

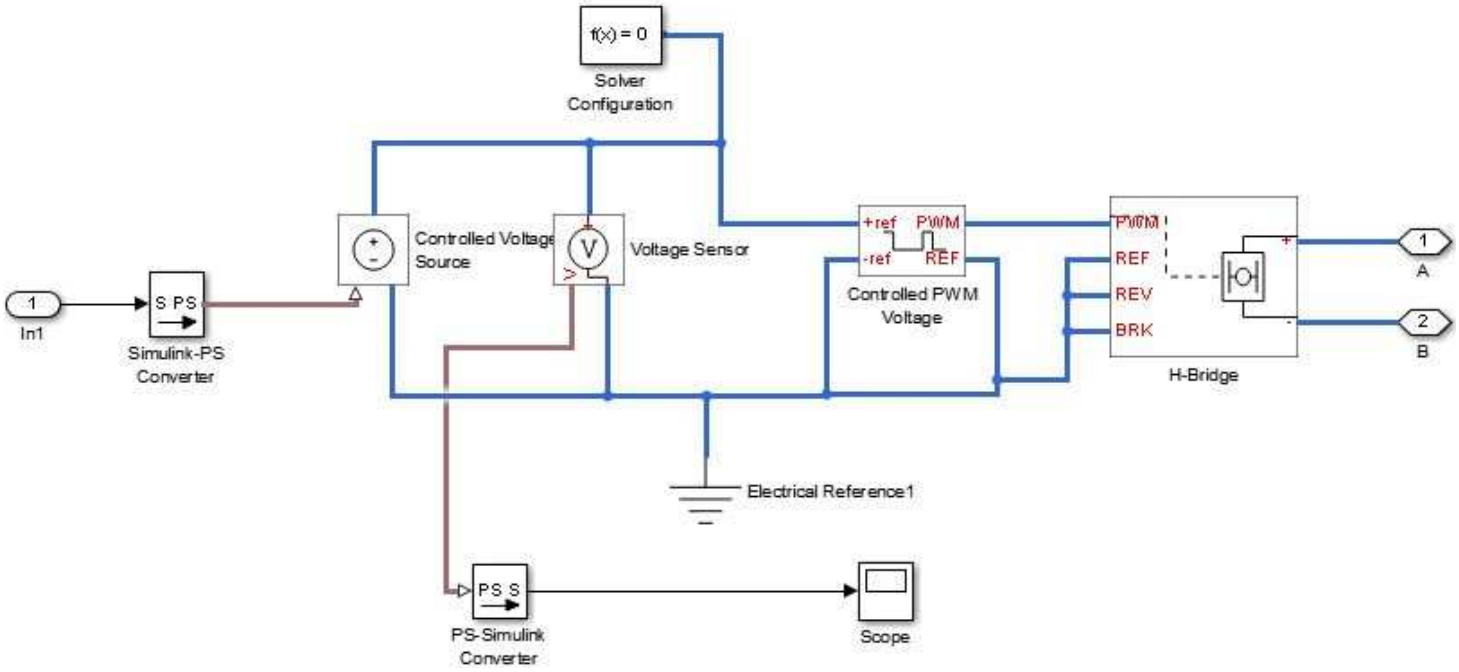


Figure 4.12 : Modèle du PWM et du Full-H bridge

Le bloc MOTOR PLANT de la fig 4.11 contient le modèle du moteur et des composants nécessaires pour obtenir le courant, la position et la vitesse comme le montre la fig 4.13. En effet, dans le bloc DC MOTOR MODEL de la fig 4.13 se trouve la fonction de transfert du moteur H(p) calculée dans l'annexe.

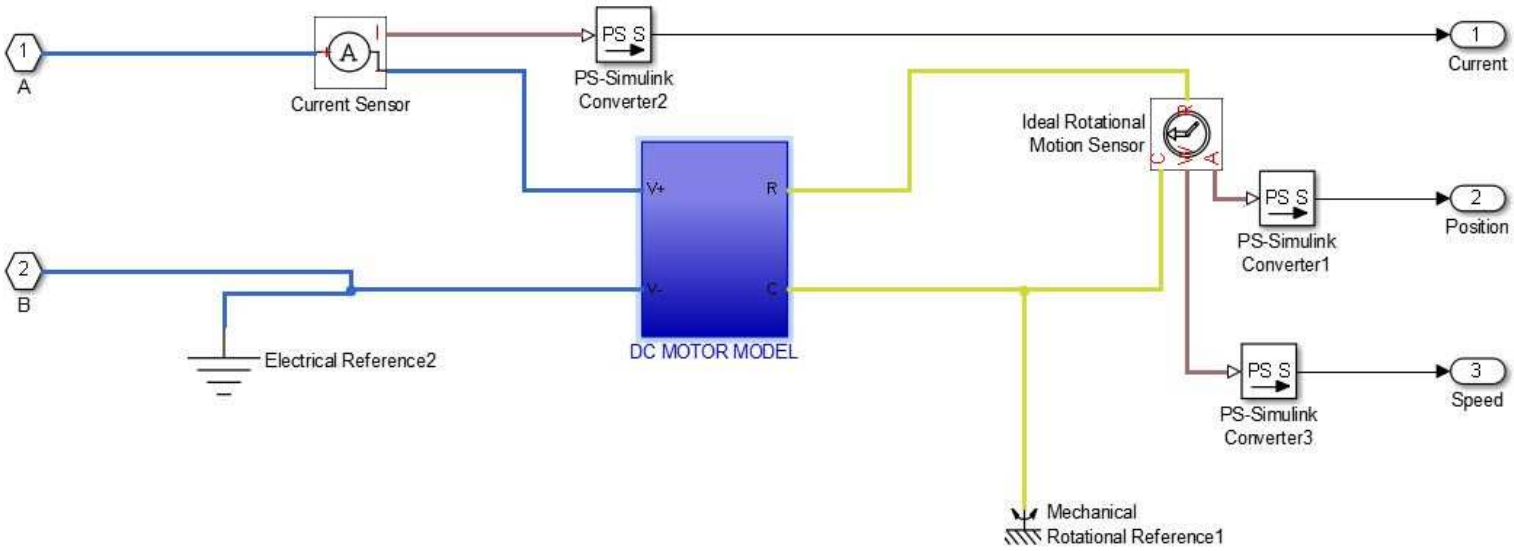


Figure 4.13 : Modèle du moteur à courant continu

Etant donné que le système est placé dans une boucle à retour unitaire, la sortie va essayer de traquer une référence à l'entrée. Lorsqu'on lance la simulation en réglant les gains proportionnel, intégral et dérivé respectivement à 1 ,1 et 0, la réponse du système à un signal échelon unité est représentée en ligne discontinue sur la fig 4.14.

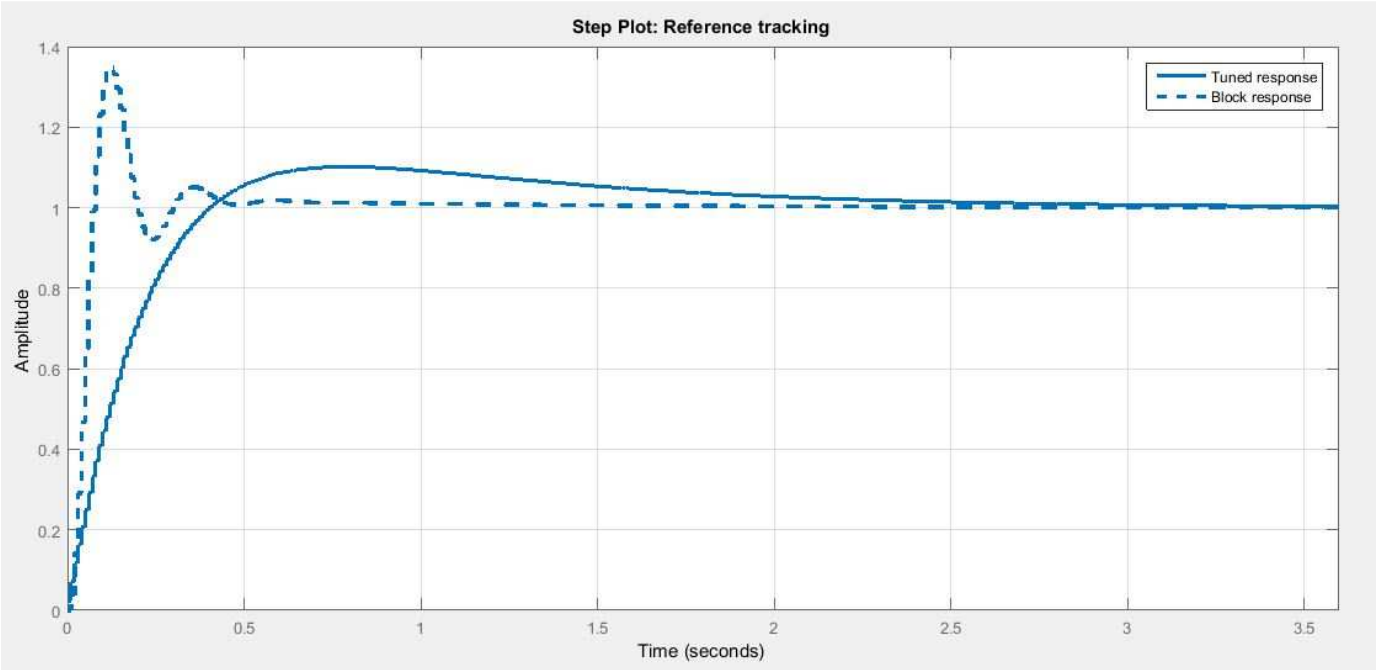


Figure 4.14 : Réponse du système à un échelon unité

Comme on peut le constater sur la fig 4.14, la réponse à un échelon présente un dépassement de 35 % et une oscillation très nette avant la stabilisation, même si cette réponse présente un bon temps de réponse. De plus, elle présente une marge de gain de 15,5 dB et une marge de phase de 37 ° qui n'est pas vraiment satisfaisante.

Par contre la réponse en ligne continue sur la fig 4.14 représente une rectification automatique des gains PID proposée par Matlab. La réponse est plus stable et l'oscillation a disparu mais les performances comme le temps de réponse se trouvent affectées.

Pour pallier à cela, il faut rectifier manuellement les gains, tout en surveillant et en ajustant si nécessaire les diagrammes de Bode, comme le montre la fig 4.15, de sorte à obtenir une bonne performance. Dans notre cas, un temps de montée de 0.1s, un dépassement inferieur à 7% et une marge de phase supérieur à 80° sont satisfaisants.

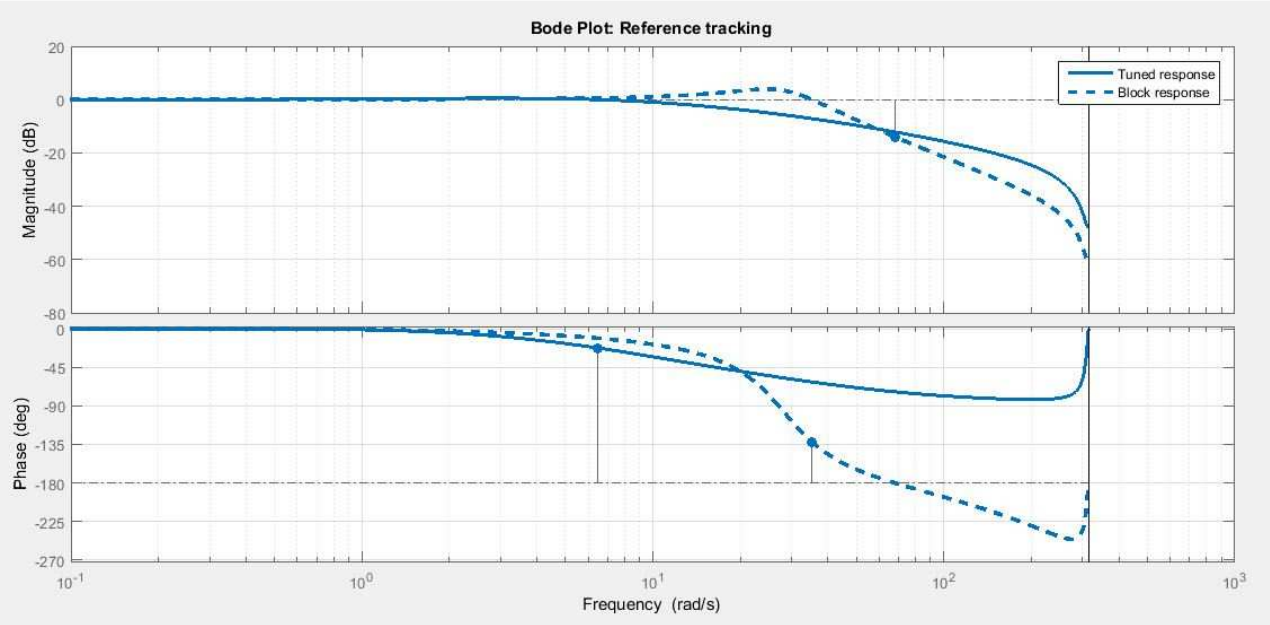


Figure 4.15 : Diagramme de Bode

Comme précédemment, les lignes discontinues de la fig 4.15 représentent le diagramme de Bode avec les coefficients Proportionnel, Intégral et Dérivée réglés respectivement à 1 ,1 et 0. Et les lignes continues de la fig 4.15 représentent le diagramme avec les coefficients qu'on a ajusté manuellement de sorte à satisfaire les exigences. Comme on peut le constater sur la fig 4.15, les marges de gain et de phase ont considérablement augmenté après l'ajustement manuel. Il est possible de localiser la mesure de la marge de phase sur le diagramme de Bode. Il suffit de repérer, grâce au diagramme de gain, la position de la pulsation de coupure à 0 dB puis, dans le diagramme de phase, à cette pulsation, de mesurer la marge de phase comme l'écart entre $-\pi$ et le déphasage correspondant [27].

Les gains sont ajustés de sorte à satisfaire un compromis entre les exigences. On peut alors retrouver la nouvelle réponse à un échelon unité représenté en ligne continue sur la fig 4.16.

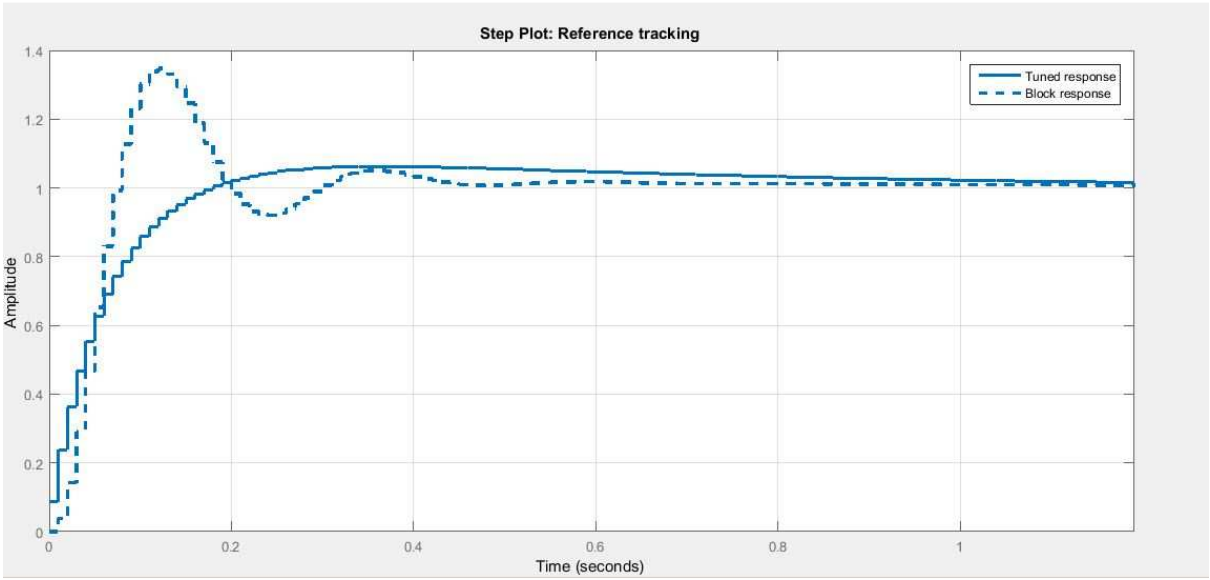


Figure 4.16 : Nouvelle réponse du système après ajustement

Cette nouvelle réponse en ligne continue sur la fig 4.16 satisfait les exigences qu'on a imposées, comme le montre la fig 4.17 qui représente une comparaison entre les caractéristiques des deux réponses de gains PID différents. (La réponse en ligne discontinue de la fig 4.16 étant toujours la réponse avec les gains PID réglés respectivement à 1 ,1 et 0.)

Controller Parameters

	Tuned	Block
P	0.51571	1
I	0.78007	1
D	0.024638	0
N		

Performance and Robustness

	Tuned	Block
Rise time	0.11 seconds	0.05 seconds
Settling time	1.12 seconds	0.43 seconds
Overshoot	6.23 %	34.8 %
Peak	1.06	1.35
Gain margin	Inf dB @ NaN rad/s	15.5 dB @ 67.9 rad/s
Phase margin	89 deg @ 16.8 rad/s	36.9 deg @ 24.1 rad/s
Closed-loop stability	Stable	Stable

Figure 4.17 : Comparaison des caractéristiques entre les deux réponses

Maintenant que les gains Kp, Ki et Kd sont trouvés, on peut lancer la simulation et observer le résultat comme le montre la fig 4.18 (la courbe de gauche représente la référence en entrée et celle de droite la réponse à cette entrée.)

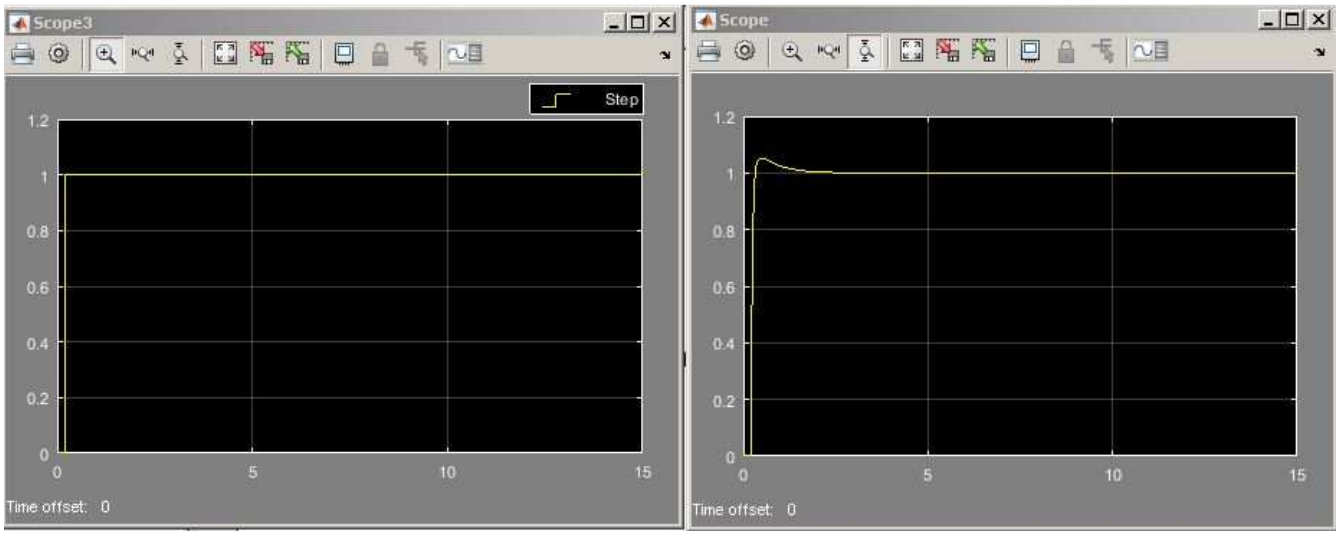


Figure 4.18 : Résultat final de la réponse

Ensuite, il suffit d'implémenter les coefficients de Kp, Ki et Kd dans l'instance de la structure arm_pid_instance_f32 de CMSIS-DSP. Le code C pour instancier cette structure est la suivante :

```
arm_pid_instance_f32 Pid ;
Pid.Kp = 0,5157;
Pid.Ki = 0,78;
Pid.Kd = 0.0246;
```

IV.4. IMPLEMENTATION DES INTERFACES DE COMMANDES DU MOTEUR ET DES CAPTEURS

a) Génération de signal PWM pour la commande du moteur

Le PWM est un train de signal carré permettant de commander le moteur ainsi de faire varier sa vitesse au choix. Générer un tel signal est chose aisée pour notre microcontrôleur en utilisant les compteurs (*Timer*). En effet, le compteur compte avec une période déterminée. Alors, on peut basculer le niveau logique de la sortie comme on le veut quand la valeur de comptage atteint une valeur désirée comme illustrée sur la fig 4.19. Ainsi, on pourra générer un train de signal carré à une fréquence dépendant du compteur tout en contrôlant le rapport cyclique.

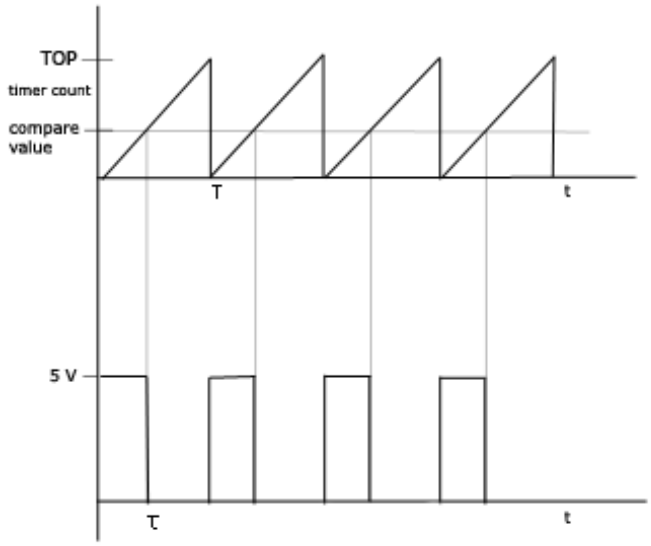


Figure 4.19 : Génération du signal PWM

Et c'est le rapport cyclique qui est le responsable de la variation de la tension d'après la formule de la tension moyenne de l'équation (4.4) [28]:

– (4.4)

En calculant l'intégrale sur le domaine où la tension n'est pas nulle on a l'équation (4.5) :

– (4.5)

Or τ est fonction de la période T donc – avec τ < T. D'où avec α <1 le rapport cyclique.

Comme on a dit : générer un tel signal est un travail facile pour le microcontrôleur, ce qui est intéressant, c'est l'utilisation du DMA pour générer le PWM. En effet, lorsqu'on utilise le correcteur PID pour corriger la vitesse, il est nécessaire de varier le PWM rapidement pour avoir le temps réel. Il faut alors accéder directement dans la mémoire et changer le rapport cyclique pour générer presque instantanément le PWM correspondant. Sa fréquence est réglée à 100Hz.

Le PWM passera dans le Full-bridge driver L298N pour commander le moteur à courant continu. Les détails concernant le driver L298N se trouvent en annexe 3.

b) Le thermomètre numérique

La figure 4.20 montre un schéma bloc du DS18S20. La ROM de 64 bits stocke le numéro de série unique dont l'identifiant Famille (octet le moins significatif des 64 bits : 10h pour les DS18S20). La mémoire « *scratchpad* » contient le registre de température de 2 octets qui mémorise la sortie numérique du capteur de température. De plus, ce bloc fournit l'accès aux registres 1 octet TH et TL des seuils (T*riggers*) d'alarmes haute et basse. Ces registres sont des mémoires non volatiles (EEPROM) et conservent leur information même lorsque l'alimentation est coupée. Le DS18S20 utilise le bus exclusif *DALLAS* 1-Wire qui permet une communication sur un seul fil. La ligne de contrôle nécessite une (forte) résistance pull-up puisque tous les dispositifs sont reliés au bus par une sortie 3 états ou une sortie drain ouvert (la broche DQ dans le cas du DS18S20). Dans ce système de bus, le microprocesseur (dispositif maître) identifie et adresse les composants sur le bus en utilisant le numéro de série unique à 64 bits [29].

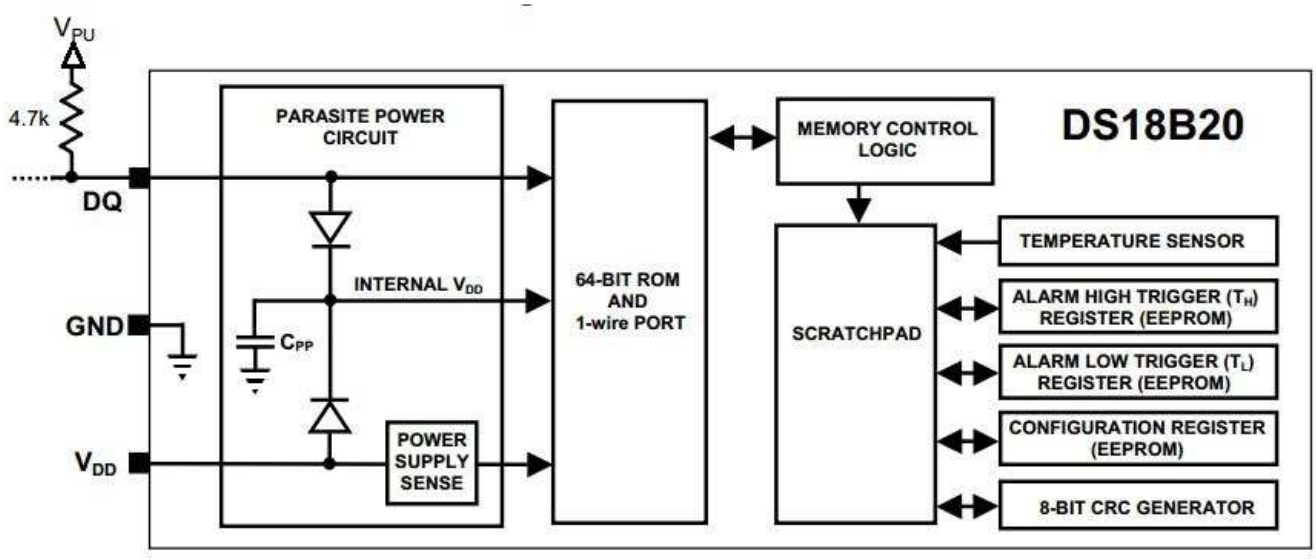


Figure 4.20: Schéma bloc du DS18B20

c) L'encodeur rotatif en quadrature

Un encodeur rotatif en quadrature est un dispositif qui génère un nombre défini de séquences de pulsations lorsqu'il fait un tour de révolution. Dans notre cas, l'encodeur génère 64 pulsations par révolution par canal ; en effet il possède deux canaux A et B qui génèrent deux mêmes séquences de pulsations mais déphasées de 90 dégrées comme le montre la fig 4.21.

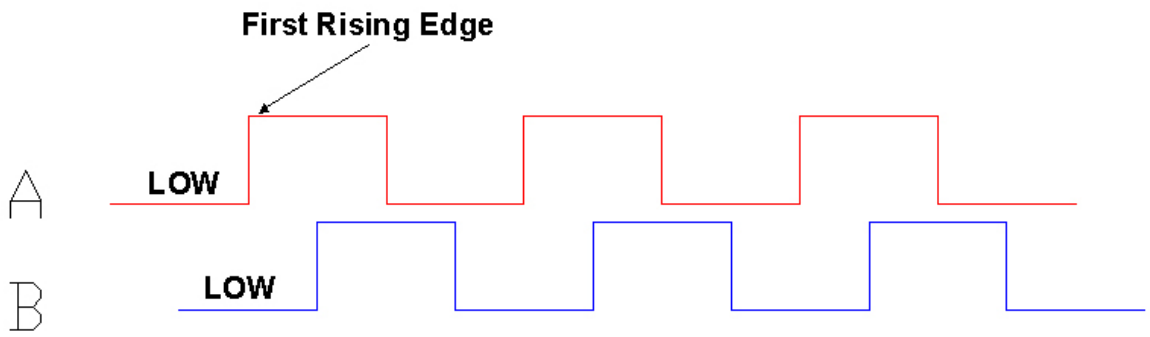


Figure 4.21: Signal de sortie des canaux A et B de l'encodeur [30]

L'encodeur permet l'acquisition de la vitesse, la position et l'angle lorsqu'on l'attache à l'arbre du moteur. Pour la vitesse il suffit de compter le nombre de pulsations à un intervalle de temps donné et de déduire la vitesse. La nécessité d'avoir deux canaux est le fait de dire au microprocesseur quand est ce qu'il va compter. En effet, à cause du déphasage, les pulsations du canal A arrivent au microprocesseur avant celles du canal B. Alors, on peut utiliser les pulsations du canal A comme des événements qui déclenchent une interruption et ainsi, on peut compter le nombre de pulsations du canal B. A chaque changement d'état du canal A de l'état *LOW* à l'état *HIGH*, le microcontrôleur compte la pulsation du canal B. Cette technique est très intéressante car lorsque le moteur tourne dans le sens inverse, ce sont les pulsations du canal B qui serviront de déclencheur d'interruption et celles du canal A sera comptées. Le microcontrôleur STM32F407ZGT6 possède une interface encodeur.

IV.5. IMPLEMENTATION DANS UN RTOS

Comme on a vu dans le chapitre précédent, un RTOS possède plusieurs attributs et fonctionnalités. Parmi ces fonctionnalités, ce qui nous intéresse c'est la gestion de multitâches c'est-à-dire la création des threads ; en effet les taches d'acquisition, les taches de commande et les taches de traitement de données ont besoins d'être harmoniser afin d'avoir une bonne performance. Dans notre cas, l'analyseur de spectre FFT, la commande du moteur en PWM asservis avec un PID, la communication via le réseau internet et l'acquisition de température implémentent chacun une tâche. Cela a pour effet de gagner du temps et de bien gérer les données car s'il s'avère qu'on n'a pas implémenté un RTOS, les tâches seraient obligées de s'exécuter de manière séquentielle mais pas simultanément. Par conséquent, il y aura une grande perte de temps et de performance car chaque tâche aurait été obligée d'attendre les autres avant de commencer. De plus, il existe des tâches qui nécessitent beaucoup plus de temps comme le cas de l'analyseur de spectre.

Les fonctions pour créer ces 4 tâches sont les suivantes :

```
/* Init task */
 xTaskCreate(Main_task, (int8_t *)"Main", configMINIMAL_STACK_SIZE * 2, NULL,MAIN_TASK_PRIO, NULL);
 xTaskCreate(Motor_task, (int8_t *)"Motor", configMINIMAL_STACK_SIZE * 2, NULL,MAIN_TASK_PRIO, NULL);
        xTaskCreate(Temperature_task, (int8_t *)"Temperature", configMINIMAL_STACK_SIZE * 2, NULL,MAIN_TASK_PRIO, NULL);
        xTaskCreate(FFT_task, (int8_t *)"FFT", configMINIMAL_STACK_SIZE * 2, NULL,MAIN_TASK_PRIO, NULL);

 /* Start scheduler */
 vTaskStartScheduler();
```

La communication via internet se fait dans la tâche Main_task et comme la logique le veut, les autres tâches devraient communiquer leurs données à cette tâche, puisqu'elle est responsable d'acheminer les données vers internet. Donc, l'utilisation des sémaphores ou des mutex , qui sont des fonctionnalités du RTOS responsables de la communication inter-Thread, est nécessaire. Cependant, dans notre cas, nous utilisons déjà un accès direct en mémoire (DMA) pour accéder aux données. D'où, il n'est plus nécessaire d'utiliser ces autres fonctionnalités du RTOS. Un extrait de code principal se trouve dans le main.c en annexe 5.

Le RTOS aussi est le responsable de la temporisation. En effet, il fournit une base de temps de retard en milliseconde avec la fonction *OsDelay(x)* qui est utilisée dans plusieurs tâches. Par exemple, dans l'acquisition de la température par le DS18B20, l'accès au registre de données températures nécessite 750 ms pour une résolution de 12 bits. Donc, il faut retarder la fonction d'accès à ce registre de 750 ms avant de faire un nouvel accès pour pouvoir récupérer une nouvelle valeur de température.

Les tâches qui ne traitent pas de données ou qui rencontrent des erreurs doivent être supprimées après avoir été exécuté car elles gaspillent beaucoup de ressources. La fonction responsable de cette suppression est la fonction *vTaskDelete (NULL*).

L'avantage aussi d'utiliser un RTOS est de pouvoir instancier plusieurs fois une même tâche. C'est très utile dans notre cas, car notre appareil est un serveur et il est possible qu'il doit accepter et créer plusieurs connections venant de plusieurs clients.

IV.6. RESULTATS

Le système a répondu à notre attente. L'acquisition de la température et de la vitesse fonctionnent en temps réel. Les données sont bien acheminées via internet vers le client en temps réel. L'analyseur de spectre fonctionne car on peut voir le spectre du signal sinusoïdal qu'on a généré pour le test. On peut constater les erreurs (la périodicité du spectre, l'existence du lobe) qu'on a prévu lors de la mise en place de l'analyseur de spectre comme le montre la fig 4.22. La vitesse du moteur suit bien la référence lorsqu'on change celle-ci. Les différentes tâches s'exécutent bien simultanément.

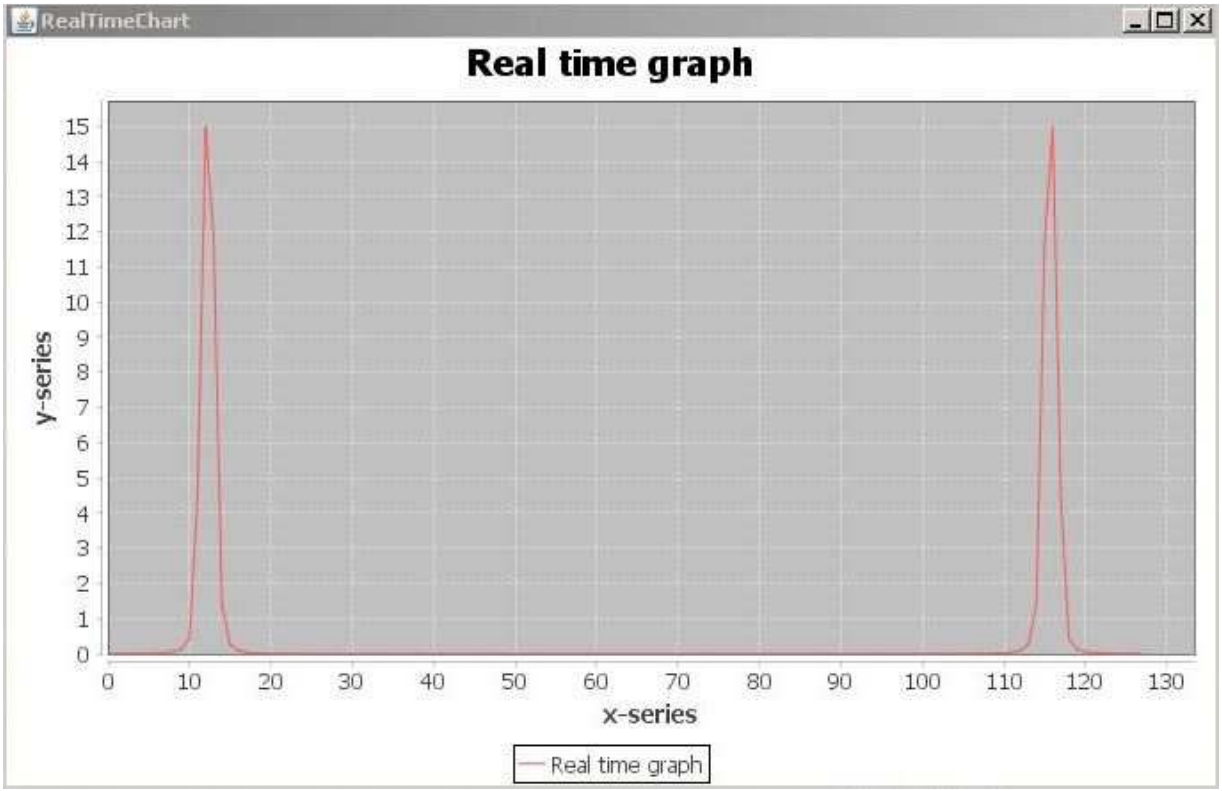


Figure 4.22 : Analyseur de spectre sur l'interface homme machine client

On constate aussi que la totalité du code source du programme ne consomme que 21 % de la mémoire et des ressources du microcontrôleur ; ce qui justifie très bien la puissance du microcontrôleur.

CONCLUSION

L'étude effectuée sur les microprocesseurs ARM a montré les différences entre chacune des technologies ARM selon leurs caractéristiques, leurs facultés et leurs architectures. On a pu en tirer leurs exploits technologiques. Et cette étude a pour but de bien les exploiter et aussi de bien choisir quels microprocesseurs ARM satisferaient les besoins d'un système.

Parmi les technologies ARM, le processeur ARM Cortex-M3/M4, conçu pour les microcontrôleurs, possède une architecture interne complexe et spécifique. L'étude du fonctionnement interne et la connaissance des technologies internes de ce microcontrôleur ont permis de le cerner. Cette étude a été nécessaire afin de mieux saisir les techniques pour bien programmer l'appareil que ce soit dans un contexte de langage machine ou de langage d'un peu plus haut niveau.

Le langage C semble un langage qui convient aux microcontrôleurs car il fournit à la fois des API de bas niveau et de haut niveau. De plus l'étude sur les outils, incluant les aides à la programmation, et l'étude sur les étapes de programmation basées autour du langage C/C++ montre la possibilité aux développeurs de bien se déployer de manière efficace dans un environnement convivial et ergonomique.

Toutes ces études ont mené à la réalisation d'un petit système serveur en temps réel implémenté sur le microcontrôleur ARM Cortex-M4. Que ce soit l'asservissement ou le traitement de signal ou la communication via internet ou l'acquisition de données ou la génération de signal ou la commande de moteur, toutes ces tâches sont très complexes en termes de calcul. Et pourtant, le serveur a su gérer tout cela en temps réel et simultanément. Cela montre la puissance du microcontrôleur ARM Cortex-M4 aussi bien dans les petites applications embarquées que dans les plus grandes.

Comme perspective pour l'amélioration du système, on pourrait utiliser des capteurs de qualités (précis et temps accès rapide). Avec la puissance du microcontrôleur, on pourrait aussi crypter la communication via internet avec des algorithmes de cryptage avancé pour des raisons de sécurité. Et enfin, l'analyseur de spectre pourrait être utilisé avec des signaux réels provenant de l'extérieur au lieu du signal test.

ANNEXE 1 : ARM

I.1. TABLEAU DE L'EVOLUTION DES PROCESSEURS ARM DEPUIS 1994

Tableau XV : Date de parution des processeurs ARM [31]

Year	Classic cores					Cortex cores			
	ARM7	ARM8	ARM9	ARM10	ARM11	Microcontroller	Real-time	Application (32-bit)	Application (64-bit)
1994	ARM7DI								
1995	ARM710a								
1996		ARM810							
1997	ARM720T ARM740T								
1998	ARM7TDMI ARM710T		ARM9TDMI ARM940T						
1999			ARM9E-S ARM966E-S						
2000			ARM920T ARM922T ARM946E-S	ARM1020T					
2001	ARM7TDMI-S ARM7EJ-S		ARM9EJ-S ARM926EJ-S	ARM1020E ARM1022E					
2002				ARM1026EJ-S	ARM1136J(F)-S				
2003			ARM968E-S		ARM1156T2(F)-S ARM1176JZ(F)-S				
2004						Cortex-M3			
2005					ARM11MPCore			Cortex-A8	
2006			ARM996HS						
2007						Cortex-M1		Cortex-A9	
2008									
2009						Cortex-M0		Cortex-A5	
2010						Cortex-M4		Cortex-A15	
2011							Cortex-R4 Cortex-R5 Cortex-R7	Cortex-A7	
2012						Cortex-M0+			Cortex-A53 Cortex-A57
2013								Cortex-A12	
2014						Cortex-M7		Cortex-A17	
2015									Cortex-A72

I.2. LISTE DE LA FAMILLE DES PROCESSEURS ARM ET LEURS ARCHITECTURES

Les familles des processeurs ARM sont listées par les tableaux qui suivent :

Tableau XVI: Familles, Architectures et noms des processeurs ARM (1) [31]

ARM family	ARM1	ARM2	ARM3	ARM6	ARM7
ARM architecture	ARMv1	ARMv2 ARMv2a	ARMv2a	ARMv3	ARMv3
ARM core	ARM1	ARM2 ARM250	ARM3	ARM60 ARM600 ARM610	ARM700 ARM710 ARM710a

Tableau XVII : Familles, Architectures et noms des processeurs ARM (2) [31]

ARM family	ARM7T	ARM7EJ	ARM8	ARM9T	ARM9E	ARM10E
ARM architecture	ARMv4T	ARMv5TEJ	ARMv4	ARMv4T	ARMv5TE ARMv5TEJ	ARMv5TE ARMv5TEJ
ARM core	ARM7TDMI(-S) ARM710T ARM720T ARM740T	ARM7EJ-S	ARM810	ARM9TDMI ARM920 ARM922 ARM940	ARM946E-S ARM966E-S ARM968E-S ARM996HS ARM926EJ-S	ARM1020E ARM1022E ARM1026EJ-S

Tableau XVIII : Familles, Architectures et noms des processeurs ARM (3) [31]

ARM family	ARM11	SecurCore	Cortex-M	Cortex-R	Cortex-A (32 bits)	Cortex-A (64 bits)
ARM architecture	ARMv6 ARMv6T2 ARMv6Z ARMv6K	ARMv6-M ARMv4T ARMv7-M	ARMv6-M ARMv7-M ARMv7E-M	ARMv7-R	ARMv7-A	ARMv8-A
ARM core	ARM1136J(F)-S ARM1156T2(F)-S ARM1176JZ(F)-S ARM11MPCore	SC000 SC100 SC300	Cortex-M0 Cortex-M0+ Cortex-M1 Cortex-M3 Cortex-M4 Cortex-M7	Cortex-R4 Cortex-R5 Cortex-R7	Cortex-A5 Cortex-A7 Cortex-A8 Cortex-A9 Cortex-A12 Cortex-A15 Cortex-A17	Cortex-A53 Cortex-A57 Cortex-A72

ANNEXE 2 : DESCRIPTIONS DES REGISTRES DU CORTEX-M3

II.1. LES COMBINAISONS POSSIBLES DES CHAMPS DE BITS DU REGISTRE DE CONTROL

Les différentes combinaisons de nPRIV et SPSEL

- nPRIV (0) et SPSEL (0): Applications simples, l'application entière est en cours d'exécution dans le niveau d'accès privilégié. Seulement une pile est utilisée par le programme principal et par les gestionnaires d'interruption. Seule la pile principale (MSP) est utilisée.
- nPRIV (0) et SPSEL (1): Applications avec un OS embarqué, avec l'exécution de la tâche actuelle fonctionnant en mode *Thread* privilégié. La PSP est sélectionnée dans la tâche en cours, et le MSP est utilisé par le noyau de l'OS et par les gestionnaires d'exceptions.
- nPRIV (1) et SPSEL (0) : mode *Thread* avec le niveau d'accès non privilégié et utilisant MSP. Cette combinaison peut être observée en *Handler mode*, mais elle est moins susceptible d'être utilisée pour les tâches de l'utilisateur parce que dans les OS embarqués, la pile pour les tâches d'application est séparée de la pile utilisée par le noyau de l'OS et par le gestionnaire d'exceptions.
- nPRIV (1) et SPSEL (1): Applications avec un OS embarqué, avec l'exécution de la tâche actuelle fonctionnant en mode *Thread* non-privilégié. La PSP est sélectionnée dans la tâche en cours, et le MSP est utilisé par le noyau de l'OS et par les gestionnaires d'exceptions [8].

II.2. DESCRIPTIONS DES CHAMPS DE BIT DU REGISTRE D'ETAT ET DE CONTROLE DE VIRGULE FLOTTANTE

Tableau XIX : Registre FPSCR [9]

Bit	Description
N	Negative flag (update by floating point comparison operations)
Z	Zero flag (update by floating point comparison operations)
C	Carry/borrow flag (update by floating point comparison operations)
V	Overflow flag (update by floating point comparison operations)
AHP	Alternate half-precision control bit: 0 – IEEE half-precision format (default) 1 – Alternative half-precision format
DN	Default NaN (Not a Number) mode control bit: 0 – NaN operands propagate through to the output of a floating point operation (default) 1 – Any operation involving one or more NaN(s) returns the default NaN
FZ	Flush-to-zero model control bit: 0 – Flush-to-zero mode disabled (default). (IEEE 754 standard compliant) 1 – Flush-to-zero mode enabled
RMode	Rounding Mode Control field. The specified rounding mode is used by almost all floating-point instructions: 00 – Round to Nearest (RN) mode (default) 01 – Round towards Plus Infinity (RP) mode 10 – Round towards Minus Infinity (RM) mode 11 – Round towards Zero (RZ) mode
IDC	Input Denormal cumulative exception bit. Set to 1 when floating point exception occurred, clear by writing 0 to this bit. (Result not within normalized value range; see section 13.1.2.)
IXC	Inexact cumulative exception bit. Set to 1 when floating point exception occurred, clear by writing 0 to this bit.
UFC	Underflow cumulative exception bit. Set to 1 when floating point exception occurred, clear by writing 0 to this bit.
OFC	Overflow cumulative exception bit. Set to 1 when floating point exception occurred, clear by writing 0 to this bit.
DZC	Division by Zero cumulative exception bit. Set to 1 when floating point exception occurred, clear by writing 0 to this bit.
IOC	Invalid Operation cumulative exception bit. Set to 1 when floating point exception occurred, clear by writing 0 to this bit.

ANNEXE 3 : DETAILS TECHNIQUES DES MATERIELS UTILISES

III.1. FICHE TECHNIQUE DU MICROCONTROLEUR STM32F407ZGT6

Le datasheet du microcontrôleur contient les informations suivantes :

- *Core: ARM® 32-bit Cortex®-M4 CPU with FPU, Adaptive real-time accelerator (ART Accelerator™) allowing 0-wait state execution from Flash memory, frequency up to 168 MHz, memory protection unit, 210 DMIPS/1.25 DMIPS/MHz (Dhrystone 2.1), and DSP instructions*
- *Memories*
    - *Up to 1 Mbyte of Flash memory*
    - *Up to 192+4 Kbytes of SRAM including 64-Kbyte of CCM (core coupled memory) data RAM*
    - *Flexible static memory controller supporting Compact Flash, SRAM, PSRAM, NOR and NAND memories*
- *LCD parallel interface, 8080/6800 modes*
- *Clock, reset and supply management*
    - *1.8 V to 3.6 V application supply and I/O*
    - *4-to-26 MHz crystal oscillator*
    - *Internal 16 MHz factory-trimmed RC (1% accuracy)*
    - *32 kHz oscillator for RTC with calibration*
    - *Internal 32 kHz RC with calibration*
    - *Sleep, Stop and Standby modes*
    - *V_{BAT} supply for RTC, 20×32 bit backup registers + optional 4 KB backup SRAM*
- *3×12-bit, 2.4 MSPS A/D converters: up to 24 channels and 7.2 MSPS in triple interleaved mode*
- *2×12-bit D/A converters*
- *General-purpose DMA: 16-stream DMA controller with FIFO and burst support*
- *Up to 17 timers: up to twelve 16-bit and two 32-bit timers up to 168 MHz, each with up to 4 IC/OC/PWM or pulse counter and quadrature (incremental) encoder input*
- *Debug mode*
    - *Serial wire debug (SWD) & JTAG interfaces*
    - *Cortex-M4 Embedded Trace Macrocell™*
- *Up to 140 I/O ports with interrupt capability*
    - *Up to 136 fast I/O up to 84 MHz*
    - *Up to 138 5 V-tolerant I/O*
- *Up to 15 communication interfaces*
    - *Up to 3 × I^2C interfaces*

    - *Up to 4 USART /2 UART*
    - *Up to 3 SPI (42 Mbits/s), 2 with muxed full-duplex I2S to achieve audio class accuracy via internal audio PLL or external clock*
    - *2 × CAN interfaces (2.0B Active)*
    - *SDIO interface*
- *Advanced connectivity*
    - *USB 2.0 full-speed device/host/OTG controller with on-chip PHY*
    - *USB 2.0 high-speed/full-speed device/host/OTG controller with dedicated DMA, on-chip full-speed PHY.*
    - *10/100 Ethernet MAC with dedicated DMA: supports IEEE 1588v2 hardware, MII/RMII*
- *8- to 14-bit parallel camera interface up to 54 Mbytes/s*
- *True random number generator*
- *CRC calculation unit*
- *96-bit unique ID*
- *RTC: subsecond accuracy, hardware calendar* [32]

III.2. FICHE TECHNIQUE DU MODULE PHY KS8721BL

Le datasheet du module ethernet PHY contient les informations suivantes :

- *Single chip 100BASE-TX/100BASE-FX/10BASE-T physical layer solution*
- *2.5V CMOS design; 2.5/3.3V tolerance on I/O*
- *3.3V single power supply with built-in voltage regulator; Power consumption <340mW (including output driver current)*
- *Fully compliant to IEEE 802.3u standard*
- *Supports MII and Reduced MII (RMII)*
- *Supports 10BASE-T, 100BASE-TX, and 100BASE-FX with Far_End_Fault Detection*
- *Supports power-down and power-saving modes*
- *Configurable through MII serial management ports or via external control pins*
- *Supports auto-negotiation and manual selection for 10/100Mbps speed and full-/half-duplex modes*
- *On-chip, built-in, analog front-end filtering for both 100BASE-TX and 10BASE-T*
- *LED outputs for link, activity, full-/half-duplex, collision, and speed*
- *KS8721BL is a drop-in replacement for the KS8721BT in the same footprint*
- *Commercial Temperature Range: 0°C to +70°C* [33].

III.3. SCHEMA DU PLATINE OLIMEX

Figure A.1: Olimex E407

III.4. FICHE TECHNIQUE DU THERMOMETRE NUMIERIQUE DS18B20

Le datasheet du thermomètre numérique contient les informations suivantes :

- *Unique 1-Wire interface requires only one port pin for communication*
- *Each device has a unique 64-bit serial code stored in an onboard ROM*
- *Multidrop capability simplifies distributed temperature sensing applications*
- *Requires no external components*
- *Can be powered from data line. Power supply range is 3.0V to 5.5V*
- *Measures temperatures from –55°C to +125°C (–67°F to +257°F)*
- *0.5°C accuracy from –10°C to +85°C*
- *Thermometer resolution is user-selectable from 9 to 12 bits*
- *Converts temperature to 12-bit digital word in 750ms (max.)*
- *User-definable nonvolatile alarm settings*
- *Alarm search command identifies and addresses devices whose temperature is outside of programmed limits (temperature alarm condition)*

- *Available in 8-pin SOIC (150mil), 8-pin SOP, and 3-pin TSOP-92 packages*
- *Software compatible with the DS1822*
- *Applications include thermostatic controls, industrial systems, consumer products, thermometers, or any thermally sensitive system* [34]

III.5. FICHE TECHNIQUE DU DRIVER L298N

Figure A.2: L298N

Le datasheet du driver L298N contient les informations suivantes :

DUAL FULL-BRIDGE DRIVER :

- *OPERATING SUPPLY VOLTAGE UP TO 46 V*
- *TOTAL DC CURRENT UP TO 4 V*
- *LOW SATURATION VOLTAGE*
- *OVERTEMPERATURE PROTECTION*
- *LOGICAL "0" INPUT VOLTAGE UP TO 1.5 V*
- *HIGH NOISE IMMUNITY [35]*

ANNEXE 4 : MODELISATION DU MOTEUR A COURANT CONTINU

IV.1. Représentation théorique du moteur à courant continu

a) Modélisation de la partie électrique

La partie électrique du moteur peut être représentée par la fig A.3 suivante :

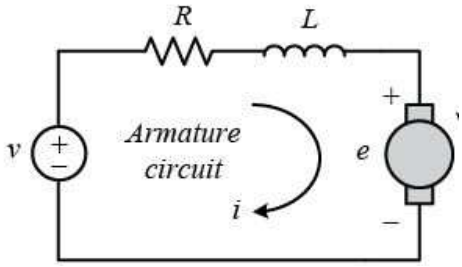


Figure A.3: Modèle électrique du moteur à courant continu

En appliquant la loi de Kirchhoff dans l'armature du circuit, on obtient l'équation (1) :

$$v(t) = v_L(t) + v_R(t) + e(t) = L\frac{di}{dt} + Ri + e(t) \quad (1)$$

Avec $v_e(t)$ la force électromotrice qui est proportionnelle avec la vitesse angulaire $\dot{\theta}(t)$ de rotation de l'arbre du moteur par la constante de force électromotrice K_e, d'où l'équation (2):

$$e(t) = K_e\dot{\theta}(t) \quad (2)$$

b) Modélisation de la partie mécanique

La partie mécanique du moteur peut être représentée par le schéma sur la fig A.4:

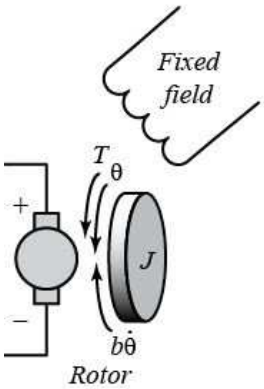


Figure A.4: Modèle mécanique du moteur à courant continu

Le modèle mécanique consiste à représenté le rotor par un volant d'inertie J soumis à :

- un <u>couple utile</u> $T\ ou\ T_u$ tel que $T_u = T_m - T_p$.
  Avec T_m le moment du couple électromagnétique généré par le moteur. En général, T_m est proportionnel au courant de l'armature et à la force produit par le champ magnétique. Etant donné que le champ magnétique est supposé constant, le moment du couple électromagnétique T_m est seulement proportionnel au courant de l'armature par la constante du couple du moteur K_t, d'où l'équation (3):
  $$T_m = K_t i \quad (3)$$
  Et T_p le moment du couple de pertes. T_p est vu comme une perturbation et il est négligeable devant le moment du couple électromagnétique T_m. Toutefois, le calcul de T_p se fait lorsque le moteur tourne à vide car elle est constante [28]. Mais pour la suite du travail, il ne sera pas pris en compte.
- <u>un couple de frottement</u> T_f proportionnel à la vitesse de rotation du rotor :
  $$T_f = b\dot{\theta}(t) \quad (4)$$
  où b est le coefficient de frottement visqueux du moteur.

En appliquant le principe fondamental de la dynamique de la seconde loi de Newton appliqué à un solide en rotation, et en remplaçant par (3) et (4), on peut écrire l'équation (5) :

$$J\ddot{\theta}(t) = T_u - T_f = T_m - T_f = K_t i(t) - b\dot{\theta}(t) \quad (5)$$

IV.2. <u>La fonction de transfert du moteur</u>

On considère que $v(t)$ soit l'entrée et $\omega(t)$ la sortie du système et que les conditions initiales sont nulles.

Après transformation dans le domaine de Laplace, les équations électriques (1) et (2) deviennent :

- $V(p) = V_L(p) + V(p) + E(p) = LpI(p) + RI(p) + E(p)$ (6)
- $E(p) = K_e\Omega(p)$ (7)

Après transformation dans le domaine de Laplace, l'équation mécanique (5) devient :

$$Jp\Omega(p) = K_t I(p) - b\Omega(p) \quad (8)$$

D'où en arrangeant (6), (7), et (8), on obtient :

$$H(p) = \frac{\Omega(p)}{V(p)} = \frac{K_t}{JLp^2 + (JR + Lb)p + Rb + K_e K_t}$$

ANNEXE 5 : EXTRAIT DE CODE EN LANGAGE C

Extrait du code main.c

```
void Main_task(void * pvParameters)
    {
        ETH_BSP_Config();                       // configure ethernet (GPIOs, clocks, MAC, DMA)
        LwIP_Init();                            // Initilaize the LwIP stack
        tcpecho_init();                         //Initialize tcp  server

     #ifdef USE_DHCP                            //   l'utilisation du #ifdef pour le DHCP [36]
        xTaskCreate(LwIP_DHCP_task, (int8_t *)"DHCP", configMINIMAL_STACK_SIZE * 2, NULL
       , DHCP_TASK_PRIO, NULL);                 //Start DHCP CLIENT
     #endif
        for( ;; )                               // infinite loop
            {
              vTaskDelete(NULL);                // Delete the task
            }
    }
void Temperature_task(void * pvParameters)
    {
         float temp;                            // temporary variable
         TM_DELAY_Init();                       // Initialize delay
         ds18b20_init();                        //initialize ds18b20 and one-wire communication
         while (1)    //infinite loop
                {
                temp = ds18b20_read();      //temperature reading
                osDelay(750);                   // delay for 12 bits resolution
                 }
    }
 void OsDelay(uint32_t nCount)
    {
        vTaskDelay(nCount);                     //     delay
     }
 #ifdef  USE_FULL_ASSERT                        // if errors occurs
  void assert_failed(uint8_t* file, uint32_t line)
    {
        exit(0);                                // stop application
        while (1);
      }
 #endif
```

REFERENCES BIBLIOGRAPHIQUES

[1] : Le processeur ARM et les SoCs, https://www.igel.com/fr/telecharger/livres-blancs.html, mars 2015

[2]: FPGA, http://www.xilinx.com/fpga/, mars 2015

[3]: James A Langbridge, Professional Embedded ARM Development, Wrox Wiley brand, 2014

[4]:MPCORE, http://www.arm.com/products/processors/classic/arm11/arm11-mpcore.php, mars 2015

[5]: Processors ARM Cortex, http://www.arm.com/products/processors/, mars 2015

[6]: Trevor Martin, The Designer's Guide to the Cortex-M Processor Family, Newnes Elsevier, 2013

[7]: Cortex Family, http://www.emcu.it/CortexFamily/CortexFamily.html, mars 2015

[8]: Joseph Yiu, The Definitive Guide to the ARM Cortex-M3, Second Edition, Newnes Elsevier, 2010

[9]: Joseph Yiu, The Definitive Guide to the ARM Cortex-M3 and Cortex-M4, Third Edition, Newnes Elsevier, 2014

[10]: Cortex-M3 processor, http://www.arm.com/products/processors/cortex-m/cortex-m3.php, mars 2015

[11]: Cortex-M0 processor, http://www.arm.com/products/processors/cortex-m/cortex-m0.php, mars 2015

[12]: Cortex-M0+ processor, http://www.arm.com/products/processors/cortex-m/cortex-m0plus.php, mars 2015

[13]: Cortex-M4 processor, http://www.arm.com/products/processors/cortex-m/cortex-m4.php, mars 2015

[14]: Cortex-M7 processor, http://www.arm.com/products/processors/cortex-m/cortex-m7.php, mars 2015

[15] : STM32F7, http://www.industrie-techno.com/arm-devoile-le-cortex-m7-stmicroelectronics-est-en-premiere-ligne-avec-son-stm32-f7.32586, mars 2015

[16] : Cortex-M Series, http://www.arm.com/products/processors/cortex-m/index.php, mars 2015

[17]: Joseph Yiu, The Definitive Guide to the ARM Cortex-M0, Newnes Elsevier, 2011

[18]: CMSIS, http://www.arm.com/products/processors/cortex-m/cortex-microcontroller-software-interface-standard.php, mars 2015

[19]: CMSIS-Core, http://www.keil.com/pack/doc/CMSIS/Core/html/_using_pg.html , mars 2015

[20]: CMSIS-Driver, http://www.keil.com/pack/doc/CMSIS/Driver/html/index.html, mars 2015

[21] : CMSIS-DSP, http://www.keil.com/pack/doc/CMSIS/DSP/html/modules.html, mars 2015

[22] : CMSIS-RTOS, http://www.keil.com/pack/doc/CMSIS/RTOS/html/_using_o_s.html, mars 2015

[23] : Lwip, http://savannah.nongnu.org/projects/lwip/, mars 2015

[24] : Socket Interface, http://docs.oracle.com/cd/E19120-01/open.solaris/817-4415/sockets-85885/index.html, mars 2015

[25] : E531, Traitement Numérique du Signal, cours en 5ème année, Département Electronique, ESPA

[26] : Analyse de Fourier, http://villemin.gerard.free.fr/Wwwgvmm/Analyse/Fourier.htm, mars 2015

[27] : E421, Système asservis Linéaire échantillonnée, cours en 4ème année, Département Electronique, ESPA

[28] : E323, Electrotechnique, cours en 3ème année, Département Electronique, ESPA

[29] : Capteur DS18B20, http://jl.and.free.fr/iut_licence//iut_licence/DS18S20_DS18B20.htm, mars 2015

[30] : Optical quadrature encoder, https://quantumdevices.wordpress.com/2010/02/22/why-use-an-optical-quadrature-encoder-for-a-motor-encoder/, mars 2015

[31]: List of ARM microarchitectures, https://en.wikipedia.org/wiki/List_of_ARM_microarchitectures, mars 2015

[32]: DATASHEET STM32F407ZGT6, http://st.com/, mars 2015

[33]: DATASHEET KS8721BL, http://micrel.com/, mars 2015

[34]: DATASHEET DS18B20, http://dallas.com/, mars 2015

[35]: DATASHEET L298N, http://st.com/, mars 2015

[36]: E331, Programmation en C, cours en 3ème année, Département Electronique, ESPA

Auteur :

ANDRY-NANJA Jerry Stéphane

Titre :

Développement sur architecture ARM Cortex-M3/M4

Nombre de pages : 112
Nombre de figures : 68
Nombre de tableau : 19

Résumé :

Aujourd'hui, l'utilisation de microprocesseur est devenue très importante dans les appareils technologiques. Ce mémoire est consacré sur le développement des microprocesseurs ARM Cortex-M3/M4 dans les systèmes embarqués. Les généralités sur ces types de processeurs et l'étude concernant la technologie interne du microprocesseur ARM Cortex-M3/M4 sont mises en valeur dans cet ouvrage. De plus, les outils de programmations et la manière de programmer sur ces microprocesseurs, basés essentiellement autour du langage C, y sont développés. Grace à ces études, on a pu réaliser un logiciel embarqué sur ce microprocesseur, pour l'acquisition et le traitement de données venant des capteurs et l'acheminement de ces données via internet en temps réel.

Mots-clés :

Microprocesseur, ARM Cortex, Microcontrôleur, RTOS, Embedded System.

Encadreur :

Monsieur ANDRIAMANANTSOA Guy Danielson

Contact :

Tel : 0331163397

Mail : lnkennedy75@gmail.com