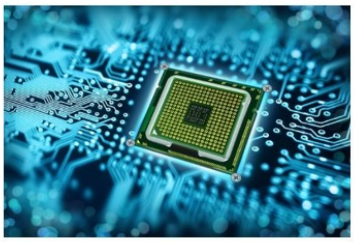
ENSI CAEN

GÉNIE PHYSIQUE ET SYSTÈMES EMBARQUES

1ère année

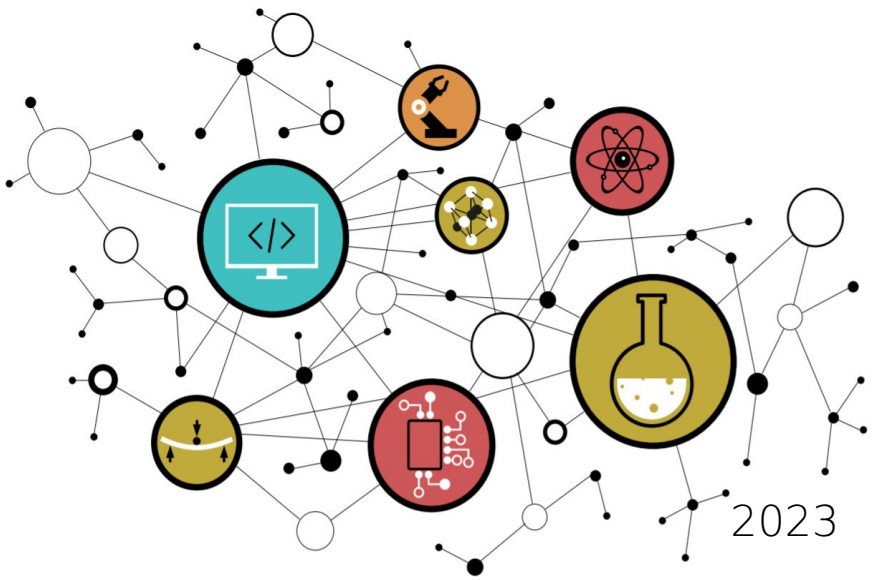
SYSTÈMES EMBARQUES

SUPPORT DE TRAVAUX PRATIQUES





2023

CONTACTS

Établissement

ENSICAEN
6 boulevard Maréchal Juin
CS 45053
14050 CAEN cedex 04

Équipe pédagogique

Isabelle Lartigau - TP
isabelle.lartigau@ensicaen.fr

Dimitri Boudier - TP
dimitri.boudier@ensicaen.fr

Arnaud Martin - TP
arnaud.martin@ensicaen.fr

hugo descoubes – TP et COURS
hugo.descoubes@ensicaen.fr
+33 (0)2 31 45 27 61

RESSOURCES

http://foad.ensicaen.fr/

Les différentes ressources numériques sont accessibles sur la plateforme pédagogique de l'ENSICAEN (aucune authentification requise, accès libre). Archive de travail mcu.zip. *Ne pas oublier de s'inscrire au cours avant tout dépôt !*

https://foad.ensicaen.fr/course/view.php?id=116

PROGRAMME ET OBJECTIFS

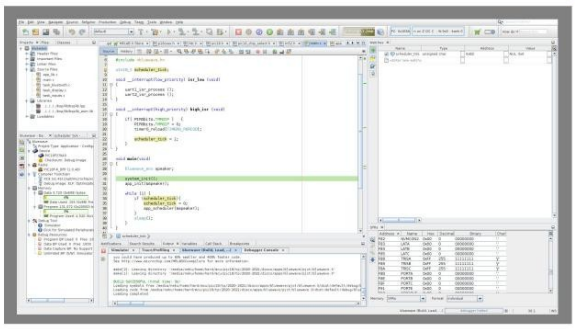
- COURS – *Comprendre le fonctionnement et l'architecture matérielle d'un processeur MCU*

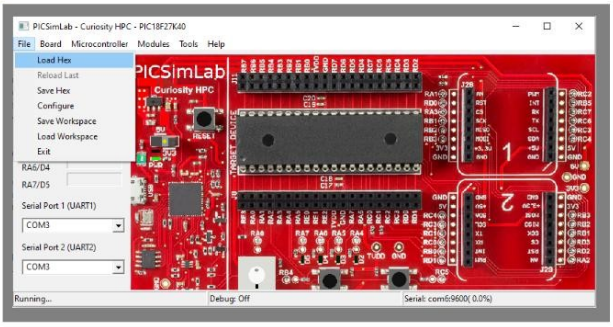
Le cours a pour objectif d'asseoir une bonne compréhension de l'architecture matérielle et du fonctionnement d'un processeur à CPU. Afin d'illustrer et présenter les grands éléments constitutifs de ce type de processeur (CPU, mémoires, périphériques et bus), nous travaillerons sur processeur MCU (Micro Controller Unit ou microcontrôleur). Une fois les grand concepts architecturaux présentés, nous appliquerons nos représentations sur technologie MCU 8bits PIC18 développée par la société Microchip. Nos analyses se feront à l'étage registre du processeur et à l'étage assembleur du modèle de développement logiciel. Une fois l'architecture matérielle assimilée, nous nous intéresserons aux méthodologies de conception et de développement d'application *bare-metal* (sans OS), notamment au concept de *scheduling offline* (stratégie d'ordonnancement). Pour clôturer les phases de cours, nous parcourrons les problématiques liées à des stratégies et techniques de communication rencontrées en Systèmes Embarqués (Liaison série, SPI, I2C, etc).

- TRAVAUX PRATIQUES – *Développer, tester, valider et documenter une solution logicielle embarquée sur MCU. Développement de bibliothèques pilotes (drivers), d'applications de test unitaires et d'une application produit.*

Pour une grande partie des réalisations, nous aurons à développer un *BSP* (Board Support Package) ou *HAL* (Hardware Abstraction Layer) *from scratch* (en partant de rien) et en travaillant à l'étage registre du processeur (plus bas niveau de développement sur machine). En résumé, nous allons tout faire de A à Z. Dans notre cas, le BSP doit être vu comme une collection de fonctions logicielles pilotes (drivers) assurant le contrôle des fonctions matérielles périphériques internes (GPIO, Timer, UART, etc) voire externes (module Bluetooth, afficheur alphanumérique LCD, etc). Notre BSP sera dédié à notre processeur et notre carte. Une migration de technologie ou de solution (processeur et/ou carte) nécessiterait ajustement et redéveloppement. Une fois le BSP développé, testé, validé, documenté et la bibliothèque statique générée, nous développerons une application audio Bluetooth *bare-metal* (sans OS) l'utilisant. L'application implémentera notamment un *scheduler offline*. Nous ne pourrons alors qu'imaginer l'infini potentiel créatif s'ouvrant devant nos yeux !

OUTILS DE DÉVELOPPEMENT





Suivre les indications présentent dans la section OUTILS DE DEVELOPPEMENT sous l'espace moodle associé à l'enseignement afin d'installer les outils. Les outils de développement proposés par Microchip sont libres d'utilisation du moment que nous utilisons les versions dîtes Free ou Lite (outils sans options d'optimisation). Chaque exercice de TP peut être pré-compilé voire testé en simulation à la maison avant l'arrivée en séance. De même, l'installation des outils puis l'utilisation en mode simulation pour une analyse de traduction de programmes C vers ASM PIC18 est sans aucun doute l'une des solutions d'apprentissage et de révision les plus efficaces lorsque nous n'avons pas en possession les plateformes matérielles. Voici ci-dessous la synthèse des outils à installer :

- IDE (Integrated Development Environment) MPLABX v5.50 : https://www.microchip.com/development-tools/pic-and-dspic-downloads-archive

- Toolchain C XC8 v1.45 (Free Mode) : https://www.microchip.com/development-tools/pic-and-dspic-downloads-archive

- Terminal asynchrone de communication TeraTerm (dernière version) : https://ttssh2.osdn.jp/index.html.en

- Drivers VCP (Virtual COM Port) pour chip USB to UART de FTDI (dernière version) : http://www.ftdichip.com/Drivers/VCP.htm

- Simulateur PICSimLab pour Windows 64bits : https://foad.ensicaen.fr/mod/resource/view.php?id=24874

- Emulateur Null Modem com0com (dernière version) : https://sourceforge.net/projects/com0com/files/com0com/3.0.0.0/com0com-3.0.0.0-i386-and-x64-signed.zip/download

ÉVALUATIONS DES COMPETENCES

L'enseignement comportera 2 évaluations distinctes. Une évaluation sur table et une évaluation pratique. Voici le détail des points sur lesquels vous serez évalués :

ÉVALUATION SUR TABLE - 2h30 : *Feuille manuscrite A4 recto/verso*

- CONNAÎTRE – 6pts : Questions de culture générale pouvant traiter sur tout point aborde en séance de cours présentiel ou présent dans le support de travail. *Connaissances fondamentales et culture scientifique de l'ingénieur électronicien*

- COMPRENDRE – 10pts : Exercice de traduction d'un programme C vers un équivalent en assembleur PIC18. Niveau d'exigence proche de l'exercice réalisé en cours. *Comprendre et maîtriser le travail d'un processeur numérique et des outils de compilation*

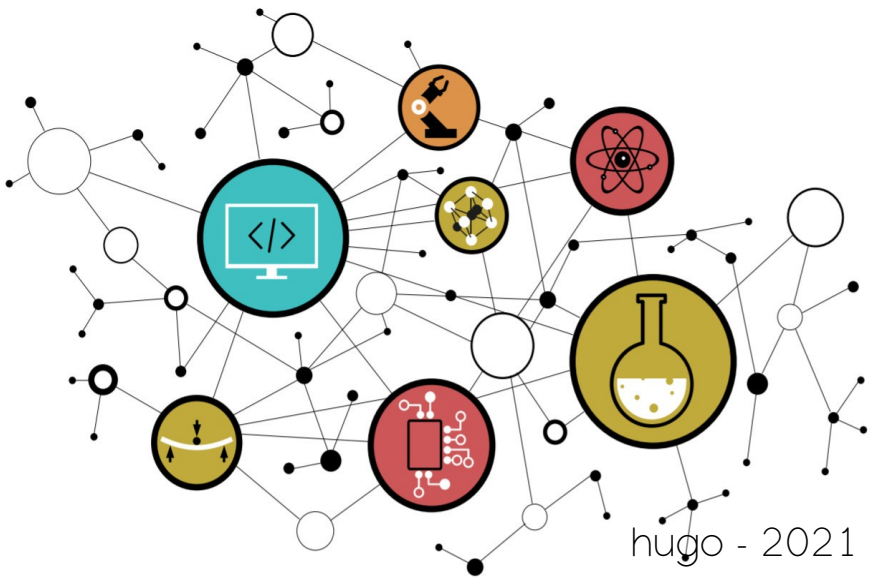
- ANALYSER – 4pts : Analyse d'un programme réalisant une application simple sur une architecture processeur non découverte en enseignement. *Adaptabilité de l'ingénieur aux concepts étudiés sur de nouvelles technologies*

ÉVALUATION PRATIQUE - 1h30 : *Tous documents autorisés*

- DEVELOPPER – 20pts : Réalisation d'un projet simple sur matériel réel pouvant traiter sur tout point abordé durant les séance de Travaux Pratiques. Se référer à son référent de TP pour les questions relatives à cette évaluation. *Faculté de l'ingénieur à répliquer ses compétences opérationnelles sur un cahier des charges nouveau*

TRAVAUX PRATIQUES

PRÉLUDE



hugo - 2021

SOMMAIRE

La trame de TP minimale que nous considérons comme être les compétences minimales à acquérir afin d'accéder aux métiers de base du domaine suit le séquencement suivant : chapitres 1, 2, 3, 4, 5 et 8. Le reste de la trame ne sera pas évalué et représente une extension au jeu de compétences minimales relatif au domaine en cours d'étude (* devant chapitres facultatifs voire complémentaires). Libre à vous d'aller plus loin selon votre temps disponible et votre volonté de mieux comprendre et maîtriser ce domaine !

5. MODULE AUDIO BLUETOOTH EXTERNE

- 5.1. Introduction : *Bluetooth*
- 5.2. Configuration du module Audio Bluetooth RN52

* 6. MODULE DE COMMUNICATION I2C ET AFFICHEUR LCD

* 7. MODULE DE CONVERSION ADC

8. CONCEPTION D'UNE APPLICATION ET ORDONNANCEMENT

- 8.1. Introduction : *Application, ordonnancement et philosophie Unix*
- 8.2. Conception et ordonnancement de l'application
- 8.3. Cahier des charges du POC (Proof Of Concept)
- 8.4. Développement du POC (Proof Of Concept)
- 8.5. Evolutions et améliorations

* 9. DOCUMENTATION TECHNIQUE ET LIVRABLES

- 9.1. Introduction : *Livrables et documentation technique*
- 9.2. Doxygen et documentation d'une bibliothèque
- 9.3. Documentation technique

SEQUENCEMENT PÉDAGOGIQUE

Le plan et le séquencement de la trame de Travaux Pratiques correspondent au chemin proposé afin d'atteindre les objectifs pédagogiques fixés. Ces objectifs ont été choisis au regard des attentes et exigences demandées par les marchés de l'industrie du logiciel et des couches basses des systèmes : systèmes embarqués, systèmes temps réel, développement de systèmes d'exploitation, développement de bibliothèques spécialisées, développement de chaînes de compilation, attaque et sécurité des systèmes, etc, tous des métiers dans divers domaines actuellement exercés par certains de nos anciens élèves. Il s'agit d'un séquencement conseillé qui n'a en aucune façon volonté à être imposé (étudiant comme enseignant encadrant). Pour se dérouler sous les meilleurs hospices, il serait cependant préférable dès le début de l'enseignement de rythmer 1 à 2 heures de travail personnel à la maison en dehors des séances en présentiels avec enseignant. Voici une proposition de séquencement (2h par créneau de TP). Lire à minima l'introduction de chaque TP avant de venir en séance :

2. Module de broche GPIO et assembleur PIC18

- En session de TP : *2.5 / 2.6 (TP n°1) 2.7 / 2.8 (TP n°2) 2.9 (TP n°3)*
- Hors session de TP : *Lire le document prélude et introduction au TP (avant TP n°1 . 2.6 (avant TP n°2) proposition pour 2.9 (avant TP n°3)*

3. Module de comptage Timer et gestion des interruptions

- En session de TP : *3.11 / 3.12 / 3.13 (TP n°4 et n°5)*
- Hors session de TP : *Lire introduction au TP n°3 et proposition pour 3.11 (avant TP n°4)*

4. Module de communication UART et liaison série

- En session de TP : *4.5 / 4.7 (TP n°6) 4.8 / 4.9 (TP n°7) 4.10 / 4.11 (TP n°8) 4.11/ 4.12 (TP n°9) 4.13/ 4.14 (TP n°10)*
- Hors session de TP : *Lire introduction au TP n°4 et proposition pour 4.5 (avant TP n°6)*

5. Module Audio Bluetooth externe

- En session de TP : *5.2 (TP n°11 et n°12)*
- Hors session de TP : *Lire introduction au TP n°5 et proposition pour 5.2 (avant TP n°11)*

8. Conception d'une Application et ordonnancement

- En session de TP : *8.2 / 8.3 / 8.4 / 8.5 / etc (TP n°13 jusqu'à la fin)*
- Hors session de TP : *Lire introduction au TP n°8 et proposition pour 8.2 (avant TP n°13)*

9. Documentation et livrables

- Hors session de TP : *Lire introduction au TP n° 9 et 9.2/ 9.3*

PRÉLUDE

1. PRÉLUDE

1.1. Présentation des Systèmes Embarqués

Un système embarqué peut-être vu comme le système (ensemble physique, électronique et informatique) embarqué dans le produit (souris, clavier, montre, voiture, fusée spatiale, carte bancaire, etc la liste est très longue). A l'image des produits, la conception d'un système embarqué peut offrir de multiples facettes. Il peut être communicant (téléphone mobile, Ebooks, montre connectée, IOT, etc) en utilisant divers protocoles de communication sans fil ou filaire (WIFI, Bluetooth, 2/3/4G, LORA, Ethernet, USB, etc), autonome sur réserve par stockage d'énergie électrique (tablette, souris, voiture, etc), portable dans la main (carte bancaire, clé de voiture, etc), faiblement encombrant (lecteur MP3, carte monétique, etc) comme très encombrant (fusée spatiale, avion, voiture, etc). Un système embarqué peut être vu comme l'ensemble des sous composantes ou sous systèmes suivants. Ce ''tout'' forme un système embarqué complet de stockage, d'échange et de traitement de l'information :

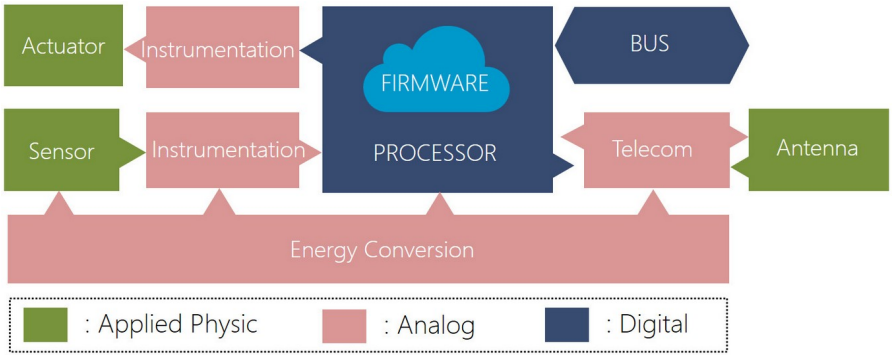
- *Système Physique (hardware/matériel)* : Interfaces avec l'environnement extérieur au produit (capteurs, actionneurs et antennes). Ponts entre l'environnement physique et le périmètre d'action du produit répondant à un besoin (capteurs de température, pression, humidité, luminosité, bouton poussoir, buzzer, LED, antennes radio, WIFI, Bluetooth, etc)

- *Système Électronique Analogique (hardware/matériel)* : Conditionnement et mise en forme des signaux de mesure et de contrôle (chaîne d'instrumentation) ainsi que des chaînes de communication (wire/filaire et wireless/sans fil).

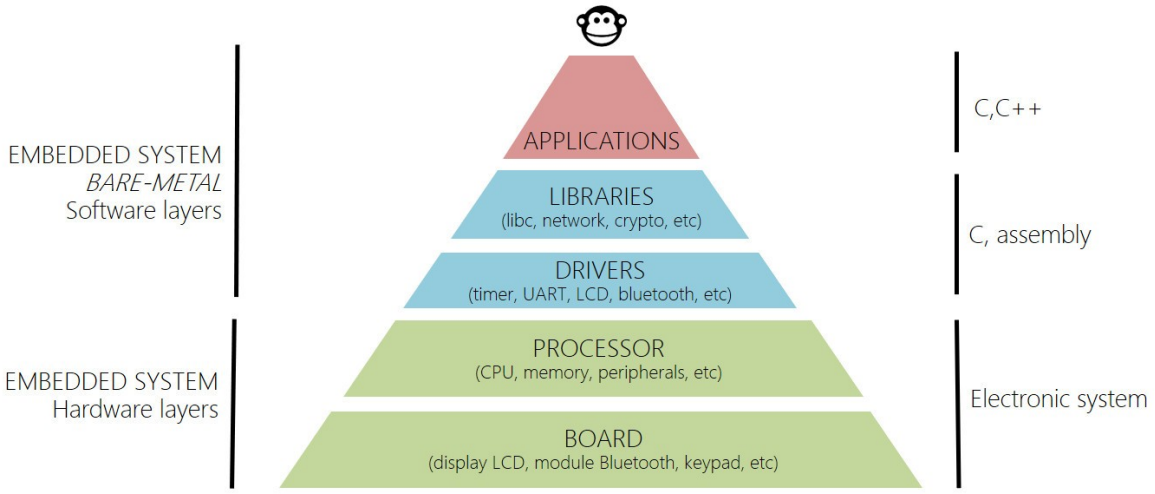
- *Système Électronique de Puissance (hardware/matériel)* : Mise en forme et dimensionnement des entrées électriques d'énergie (redresseur, hacheurs flyback, forward, etc). Stockage de l'énergie électrique sur système embarqué autonome (batterie).

- *Système Électronique Numérique (hardware/matériel)* : Stockage numérique des données (mémoire donnée) et des programmes (mémoire programme). Traitement des données par exécution du programme puis mise en forme de l'information (processeur). Interfaces numériques de communication et d'échange avec l'extérieur (fonctions périphériques)

- *Système Informatique (software/logiciel et firmware/micrologiciel)* : Couche système (système d'exploitation du matériel), couche bibliothèque (utilitaires et fonctionnalités pour les applications) et couche applicative (missions de supervision du système appliquées à des besoins spécifiques voire problématiques de calcul)

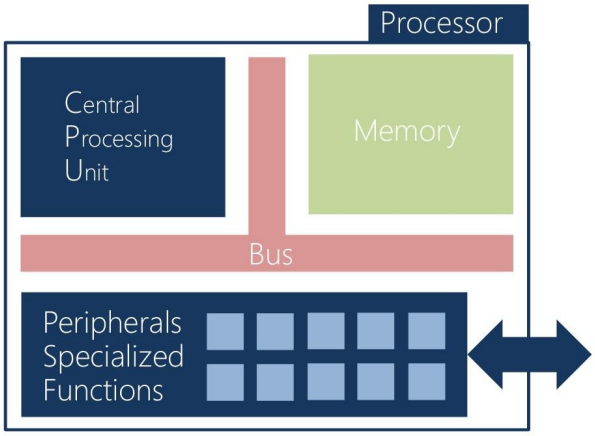


Au cœur de la très grande majorité des produits électroniques actuels se trouvent enfouis un (souris, montre, carte monétique, etc) voire plusieurs processeurs (voiture, avion, etc). Un système embarqué, peut d'ailleurs interagir avec plusieurs sous systèmes embarqués (un ordinateur et une souris par exemple). Différentes familles de processeurs cohabitent sur le marché (MCU, GPP/MPU, AP/MPU, DSP, MPPA, FPGA, GPU, etc), chacune répondant à des besoins et exigences (application/supervision ou calcul/algorithmique, consommation, coût, encombrement, performance, etc). Durant cet enseignement, nous nous intéresserons à l'architecture processeur la plus répandue en volume sur le marché, celle des MCU (Micro Controller Unit ou microcontrôleur).



Un système numérique de traitement de l'information peut souvent être représenté en couches matérielles comme logicielles. Ce modèle représente grossièrement les dépendances des différentes fonctionnalités matérielles et logicielles entre elles (application, bibliothèques, pilotes, périphériques internes et externes, etc) et donc le chemin de l'information dans le système (montante/entrante ou descendante/sortante). L'objectif premier restant de développer une application logicielle, répondant à un besoin et supervisant le système matériel dédié. Contrairement à un ordinateur cherchant la généricité (système d'exploitation multi-applicatifs, interfaces utilisateur génériques et standards, etc), un système embarqué est développé pour répondre spécifiquement et de façon optimale à un besoin. Une souris n'est pas une montre !

Nous allons tout au long de cet enseignement nous efforcer de comprendre puis maîtriser au mieux les différentes couches de ce modèle. Les solutions matérielles de prototypage ayant été spécifiées (MCU 8bits PIC18F27K40 et carte curiosity HPC de Microchip), une analyse attentive des documentations techniques (datasheet) sera nécessaire afin de développer les couches pilotes (drivers) puis applicative répondant à nos besoins. L'application terminale de l'enseignement visant à concevoir et développer un application audio Bluetooth. Cependant, une grande partie de notre travail consistera à développer un BSP spécifique (Board Support Package, bibliothèque de fonctions pilotes dédiées à notre carte et MCU).



Un MCU (Micro Controller Unit) ou microcontrôleur est une famille de processeur numérique présente sur le marché de l'électronique. Le premier MCU produit et commercialisé date de 1971 par Texas Instruments. Cette famille de processeur possède notamment comme particularité d'intégrer sur une même puce de silicium (on chip) l'ensemble des éléments (CPU, bus, mémoires et périphériques) faisant d'elle un processeur complet sur puce, contrairement aux ordinateurs où les composants silicium électroniques (GPP/MPU, mémoires vive DRAM et de masse HDD/SSD, chipset, ec) sont distincts et portés sur une carte mère (circuit imprimé).

Les MCU sont dédiés et spécialisés aux applications exigeant un faible encombrement spatial (processeur complet intégré sur puce), une faible consommation énergétique (souvent mono CPU à qq10-100MHz) et un faible coût financier (entre de qq0,1€ à qq1€ l'unité). Il s'agit des processeurs les plus fabriqués en volumes dans le monde chaque année avec près de 25 milliards d'unités en 2020 (source IC Insigths). Il s'agit de composants numériques de stockage,d'échange et de traitement de l'information. Rappelons les rôles de chaque entité d'un processeur architecturé autour d'un CPU :

- *CPU (Central Processing Unit) – Traiter l'information :* Composant chargé de récupérer séquentiellement par copie (fetch) une à une les instructions du programme binaire exécutable (firmware) présent en mémoire. Une fois récupérée, chaque instruction est décodée (decode), exécutée (execute) puis le résultat sauvé dans la machine (writeback). Hormis lorsqu'il est forcé en veille, un CPU réalisera sans arrêt les étapes séquentielles suivantes Fetch/Decode/Execute/WriteBack. Le pipeline matériel d'un CPU correspond à sa capacité à réaliser ces traitements en parallèle. Il existe plusieurs modèles architecturaux de fonctionnement de CPU (RISC/CISC, Harvard/VonNeumann/Hybrid, Scalar/Superscalar/VLIW, etc). Tous seront découverts en formation dans la suite du cursus.

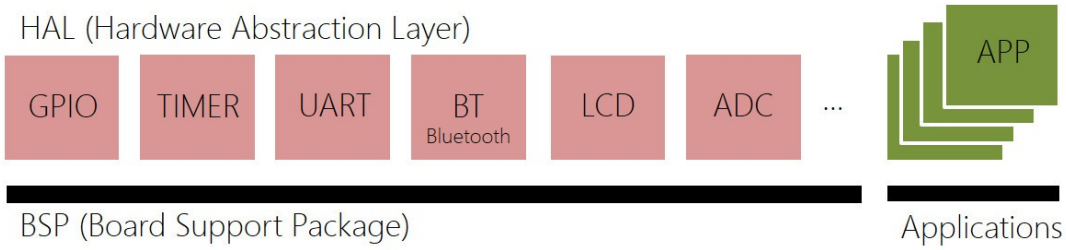
- *Mémoire – Stocker l'information :* La mémoire est chargée de stocker l'information. L'information peut être de deux natures différentes dans la machine, instructions du programme (code) ou données (data)

  - *Mémoire programme :* mémoire non-volatile (persistante), elle sera souvent nommée historiquement mémoire Flash sur MCU. Elle sera le plus souvent accessible en lecture seule par le CPU à l'usage (ReadOnly). Les technologies les plus répandues sont les mémoires flash NOR et flash NAND.

  - *Mémoire donnée :* mémoire volatile le plus souvent de technologie SRAM sur MCU. Elle sera accessible en Lecture et écriture par le CPU à l'usage (Read/Write)

- *Périphériques – Échanger/Convertir/Traiter l'information :* Les fonctions matérielles spécialisées périphériques à l'ensemble CPU/Mémoire, plus communément nommées ''périphériques'', jouent généralement l'un des trois rôles suivants dans le système : *communiquer ou échanger l'information* (par protocole de communication), *convertir l'information* (du domaine analogique vers numérique) ou *traiter l'information* (fonction de traitement matérielle spécialisée)

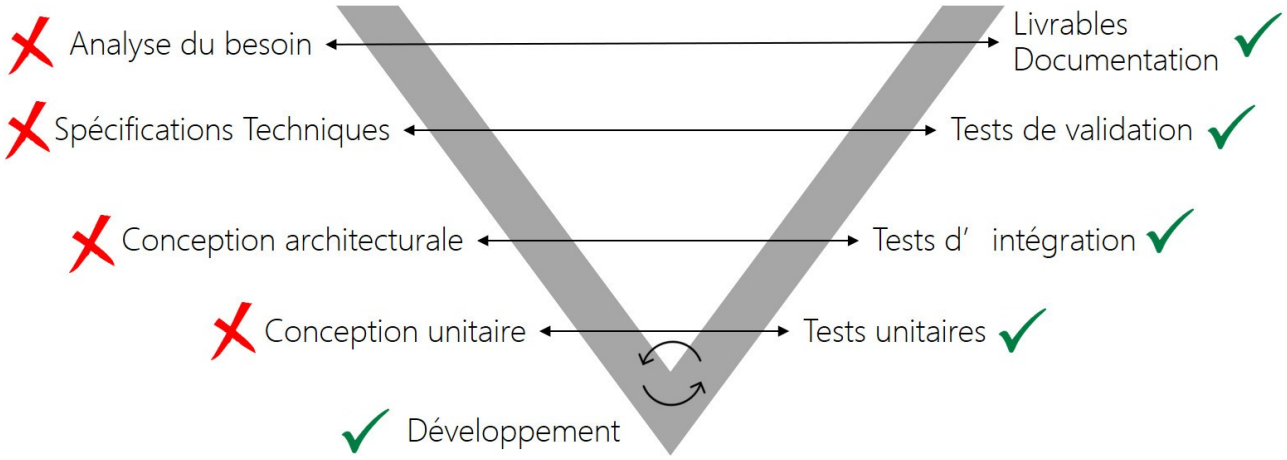
1.2. Objectifs pédagogiques



Les objectifs de cet enseignement sont multiples. Pour une grande partie des réalisations, nous aurons à développer un *BSP (Board Support Package)* ou *HAL (Hardware Abstraction Layer) from scratch* (en partant de rien) et en travaillant à l'étage registre du processeur (plus bas niveau de développement sur machine). En résumé, nous allons tout développer de A à Z. Dans notre cas, le BSP doit être vu comme une collection de fonctions logicielles pilotes (drivers) assurant le contrôle des fonctions matérielles périphériques internes (GPIO, Timer, UART, I2C, etc) voire externes (LED, switch, module Audio Bluetooth, potentiomètre, afficheur alphanumérique LCD, etc).

Le BSP sera dédié à notre processeur de travail (MCU 8bits PIC18F27K40 Microchip) et notre carte (Curiosity HPC Microchip). Une migration de technologie processeur et/ou carte nécessiterait des ajustements et de nouveaux développements. Une fois le BSP développé, testé, validé, documenté et la bibliothèque statique générée, nous développerons une application audio Bluetooth *bare-metal*. Dans le domaine de l'embarqué, *Baremetal* signifie nu sans système d'exploitation (OS ou Operating System ni RTOS ou Real Time OS). L'application implémentera un *scheduler offline*, ce point sera vu plus en détail dans la suite de la trame. En fin d'enseignement, nous ne pourrons alors qu'imaginer l'infini potentiel créatif s'ouvrant devant nos yeux !

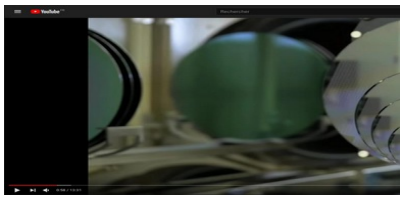
La chronologie de la trame de TP est construite autour d'un workflow typique rencontré en industrie autour de ce type de développements. Avant de développer tout applicatif, nous aurons à valider unitairement toutes les fonctionnalités et interfaces de l'application (matérielles et logicielles). Nous pourrons ensuite nous focaliser sur les phases d'intégration successives. Même si des méthodologies agiles de gestion de projet seront utilisées lors de projets en équipe en 2ième et 3ième année en majeure, celui découvert ici présente un cycle en V (V et agilité peuvent cohabiter). De même, il ne serait pas raisonnable ni pertinent de demander à un élève ingénieur en formation 1ière année de réaliser la conception d'un projet de cette taille sans avoir le recul suffisant sur le domaine. Les spécifications techniques et les conceptions ont donc été réalisées (matériel et logiciel, architecturales et unitaires), vous aurez donc à vous focaliser sur les phases de développement, de test unitaire, de test d'intégration et de validation. La conception sera découverte en majeure durant la 2ième année et la 3ième année. Ce point nécessite une très bonne assise des compétences de 1ière année.

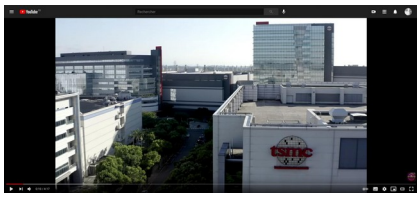


1.3. Quelques ressources internet

Voici quelques ressources en ligne pouvant vous aider à une meilleure contextualisation du domaine, des acteurs, compréhension des processus de fabrication et des outils que nous utiliserons durant cet enseignement.

How microchips are made – Infineon (13mn) and TSMC foundry world leader (4mn)

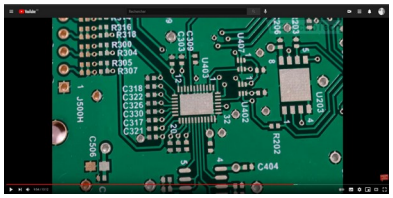




https://www.youtube.com/watch?v=bor0qLifjz4

https://www.youtube.com/watch?v=Hb1WDxSoSec

How PCB and MotherBoards are made – PCBWay (13mn) and GigaByte (2mn)



https://www.youtube.com/watch?v=_GVk_hEMjzs&t=637s

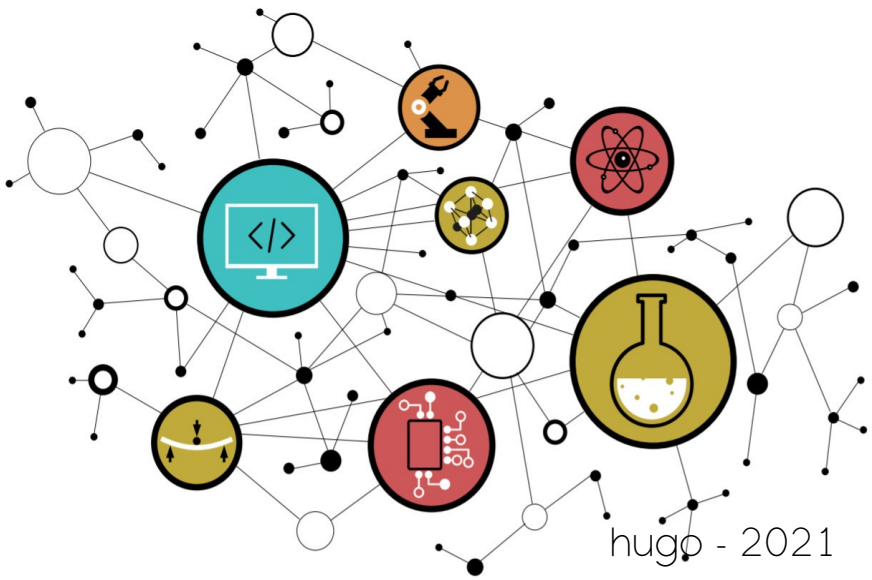
https://www.youtube.com/watch?v=bR-DOeAm-PQ

Microchip Company – Microchip (5mn)

https://www.youtube.com/watch?v=-N20QMIgh4Q

TRAVAUX PRATIQUES

MODULE DE BROCHE GPIO
ET ASSEMBLEUR PIC18

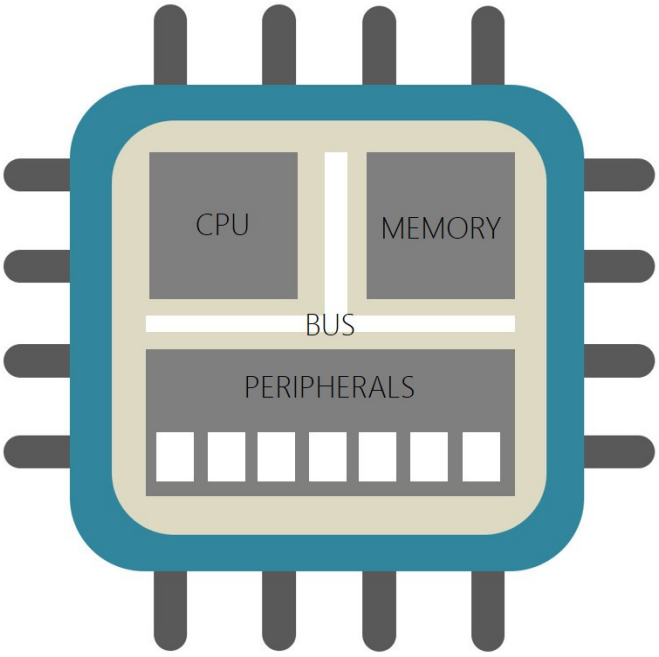


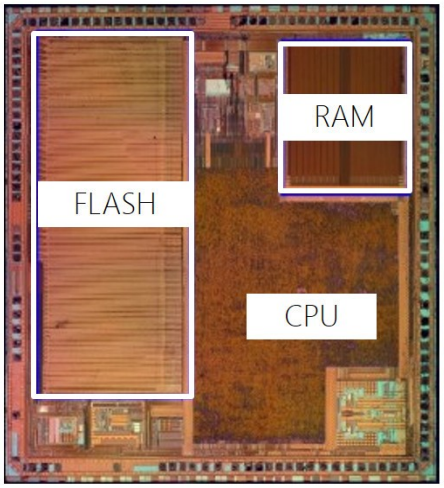
hugo - 2021

SOMMAIRE

MODULE DE BROCHE GPIO ET ASSEMBLEUR PIC18

2. MODULE DE BROCHE GPIO ET ASSEMBLEUR PIC18





Die MCU 32bits Microchip
MCU PIC32MX340F512
MIPS32 M4K Core 5-stage Pipeline
Memory 512Ko Flash / 32Ko SRAM

Au fil des introductions de chaque document de TP, nous allons vous présenter les architectures et les fonctionnements génériques de périphériques standards (GPIO, Timer et UART) présents sur la grande majorité des MCU du marché (Micro Controller Unit ou Microcontrôleur). Pour chaque périphérique, une illustration technologique sur PIC18 (solution MCU 8bits propriétaire Microchip) sera également proposée ainsi qu'un exemple de configuration bas niveau en assembleur PIC18.

Rappelons que sur processeur MCU, la mémoire (program Flash et data SRAM) et le CPU (Central Processing Unit) sont respectivement chargés de stocker l'information (programme et données) puis de la traiter. La mémoire morte Flash non-volatile stocke de façon persistante le programme (ou code) et la mémoire vive SRAM volatile stocke à l'exécution les données en cours de manipulation par le CPU. D'une implémentation technologique à une autre, l'architecture et donc l'empreinte silicium de ces deux éléments fondamentaux pendront plus ou moins de place sur le *die* (puce silicium). A titre indicatif, l'exemple ci-dessus montre les contraintes d'intégration d'un MCU 32bits Microchip du marché. Les services proposés par un processeur seront toujours le fruit de compromis technologiques liés à l'intégration sur silicium. D'un point de vu étymologique, l'ensemble CPU/Mémoire représente déjà à lui seul un processeur (stockage et traitement de l'information). Une fois l'information stockée puis traitée, l'application sera chargée de l'échanger vers l'extérieur du système. Les périphériques entrent alors en jeu.

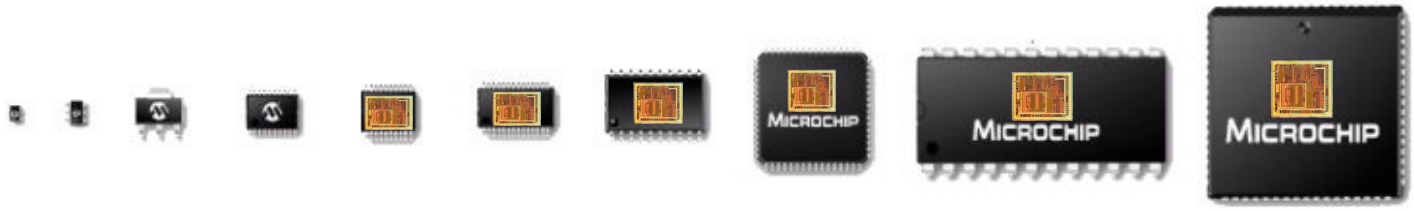
Un périphérique est un service matériel proposé par la machine. Les fonctions matérielles spécialisées périphériques à l'ensemble CPU/Mémoire, plus communément nommées ''périphériques'', jouent généralement l'un des trois rôles suivants dans le système :

- *Communiquer :* Fonction spécialisée d'échange et de partage d'information de machine à machine implémentant un protocole de communication souvent normalisé voire standard (UART, SPI, I2C, USB, Ethernet, CAN, etc). Un protocole de communication assure une encapsulation de l'information ou charge strictement utile (payload).avant transmission ou réception par trames de communication.

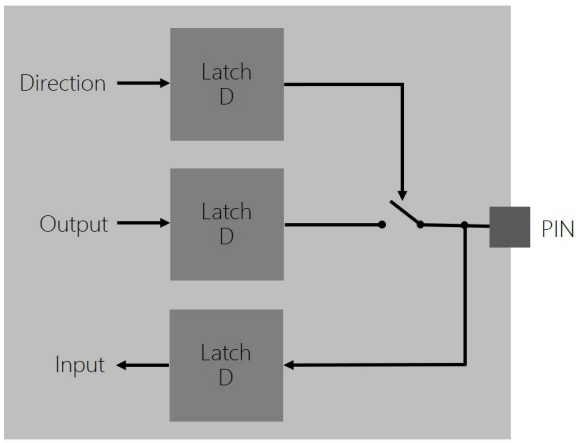
- *Convertir :* Fonction spécialisée de conversion (GPIO, ADC, DAC, PWM, etc) entre les domaines de l'électronique analogique (signaux physiques continus) et numérique (signaux logiques discrets)

- *Traiter (voire accélérer) :* Fonction spécialisée interne (Timer, DMA, Crypto, FFT, etc) de traitement (comptage, copie, calcul, etc). Ceci permet d'aider le CPU à se dédier à la supervision du système par exécution du programme applicatif voire dans certains cas à exécuter du calcul algorithmique.

2.1. Module GPIO ou General Purpose Input Output

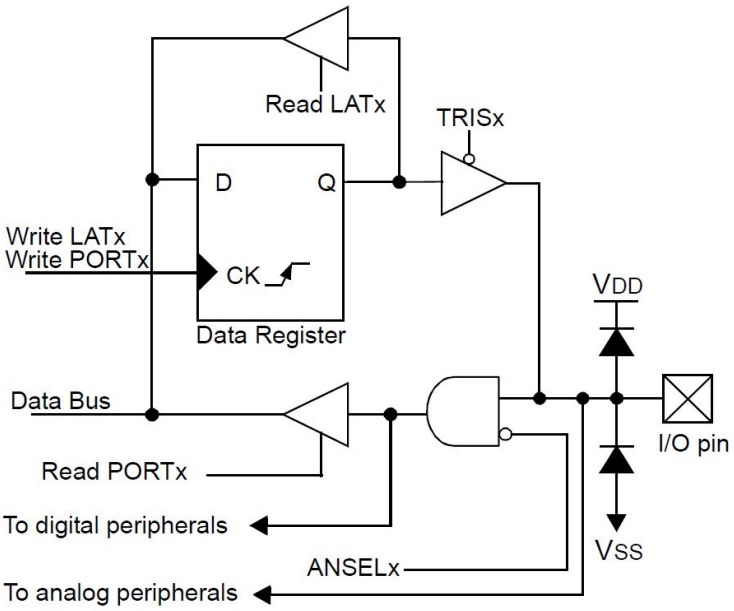


Une broche (ou pin) est une interface physique présente et visible à l'œil nu sur le boîtier d'un MCU ou microcontrôleur. Il est à noter qu'une même puce de silicium (die), un MCU par exemple, peut être encapsulée dans différentes technologies de boîtier (DIP, SOIC, BGA, QFN, etc). Chaque technologie offre son lot d'avantages et d'inconvénients (coût, encombrement, facilité de maintenance, procédé d'intégration, etc). Une broche assure une interface physique avec le module GPIO présent quant à lui sur la puce de silicium constituant le processeur MCU (cf. schéma fonctionnel ci-dessous). Sur tout processeur du marché (MCU, DSP, MPPA, FPGA, etc), une broche d'entrée/sortie généraliste ou GPIO (General Purpose Input Output) offrira toujours un service de configuration de la direction (entrée ou sortie) de type TOR (Tout Ou Rien, par exemple 0V ou Vcc) puis un service d'utilisation en lecture ou en écriture. L'objectif étant de pouvoir lire ou écrire l'état d'une broche. Ceci peut notamment servir d'interface avec des fonctions périphériques externes au processeur. Ces fonctions matérielles externes seront portées sur le PCB (Printed Circuit Board ou circuit imprimé). Par exemple, les GPIO's peuvent permettre d'interfacer des LED ou différents actionneurs (afficheur, servomoteur, contacteur, etc), mais également à lire l'état de boutons poussoirs, d'interrupteurs voire de différents capteurs du marché (capteur électromécanique de fin de course, capteurs optiques, etc).

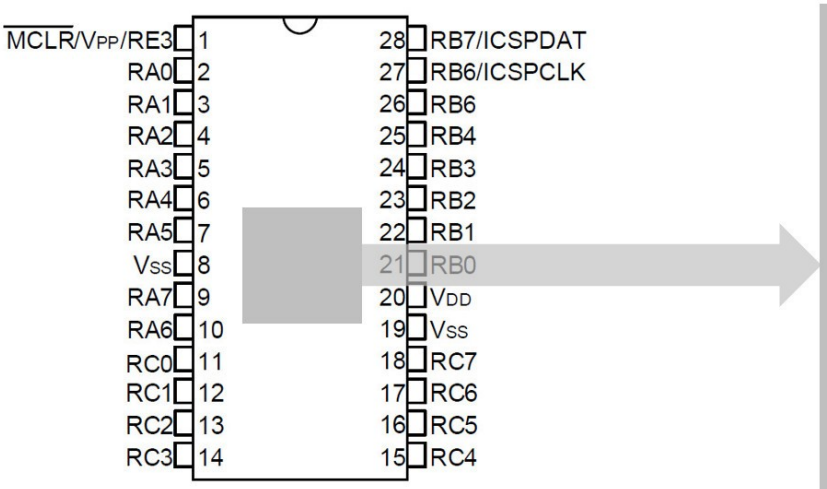


Un module GPIO associé à une broche est une fonction matérielle interne à un MCU. Il est constitué sur silicium d'un circuit logique. La figure ci-dessus ne présente par exemple que le schéma symbolique pour la gestion d'une seule broche. Chacun des états logiques de configuration et d'utilisation d'une broche sera sauvegardé par des bascules (bascule D le plus souvent) assurant la mémorisation logique de ces mêmes états. Nous nommons un ensemble de bascules un registre. Pour un MCU, un PORT (de broches) est un ensemble de broches gérées par registre. De façon générale sur un processeur, un registre peut s'agir de registre de configuration (à écrire), d'utilisation (à lire et écrire) voire d'état (à lire). D'une implémentation technologique à une autre les registres auront des fonctions, des rôles, des noms et des tailles différentes (exemples des technologies MCU Microchip PIC18, MCU STMicroelectronics STM32, MCU Texas Instruments MSP430, etc). Ces aspects sont liés à la technologie utilisée. Une lecture et analyse des documentations techniques du constructeur est donc indispensable dans ce domaine.

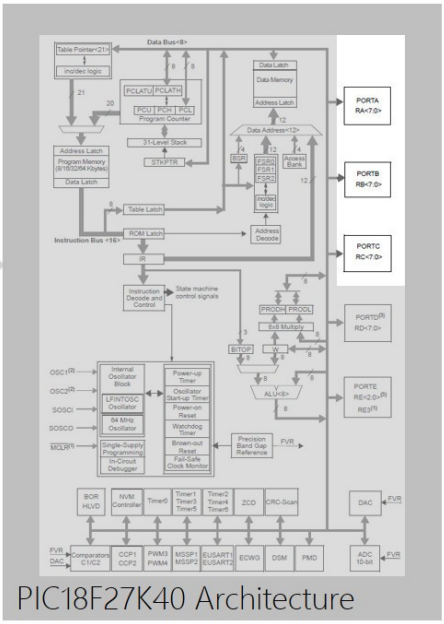
2.2. Module GPIO sur MCU PIC18



Le schéma précédent est extrait de la documentation technique ou datasheet d'un MCU PIC18F27K40 proposé par Microchip. Il présente plus en détail l'architecture matérielle réelle cachée derrière chaque GPIO et broche. Un PORT d'entrée sortie (PORTA, PORTB et PORTC sur PIC18F27K40 avec boîtier 28 broches) représente un ensemble de 8 broches ou pins de type GPIO. Les ports sont manipulables par utilisation de registres 8bits (regroupement en tout de 8 GPIO par PORT).



MCU PIC18F27K40
Packages SPDIP, SSOP, SOIC



PIC18F27K40 Architecture

Cependant, en fonction des besoins dans l'application, ces broches peuvent être configurées et utilisées pour d'autres usage. Les broches sont alors physiquement routées en interne vers d'autres fonctions matérielles périphériques (Timer, UART, ADC, SPI, I2C, etc). Tout ceci doit explicitement être programmé par le développeur et est documenté dans la datasheet du composant.

2.3. Configuration des GPIO sur PIC18

Sur PIC18, les registres de configuration en entrée/sortie des ports sont nommés TRISx (x=A,B, C sur PIC18F27K40) pour Tri-state (3 états : haut ou 1, bas ou 0, et haute impédance ou Z). Les registres d'écriture sont nommés LATx (pour Latch car associé à une bascule D latch) et les registres de lecture PORTx. En résumé, voici les usages des 3 familles de registres 8bits pour la gestion des GPIO sur MCU PIC18 :

- TRISx : Configuration (0=Output et 1=Input) de la direction des broches du PORT x (x=A,B ou C)
- LATx : Écriture (0=0v et 1=Vcc) de l'état logique des broches du PORT x (x=A,B ou C)
- PORTx : Lecture (0=0v et 1=Vcc) de l'état logique des broches du PORT x (x=A,B ou C)

Ouvrir la datasheet des processeurs PIC18Fx7K40 et parcourir le chapitre 15 relatif aux GPIO (I/O Ports) afin d'observer la configuration des registres (mcu/tp.doc/datasheets). La séquence assembleur PIC18 ci-dessous présente des exemples de configuration et de gestion des ports en assembleur PIC18.

```
; all PORTA pins are inputs

MOVLW       0xFF
MOVWF       TRISA


; all PORTB pins are outputs

MOVLW       0x00
MOVWF       TRISB


; pin RC3 is an output

BCF       TRISC, 3


; pin RC0 is an input

BSF       TRISC, 0
```

```
; read all PORTA pin and save value in W (Work) CPU register

MOVF       PORTA, 0
or
MOVFF      PORTA, WREG


; pins RB0-RB3 are set to low level and RB4-RB7 to high

MOVLW       0xF0
MOVWF       LATB


; set RC3 to high level

BSF       LATC, 3


; test if RC0 input level is low and perform action

BTFSC       PORTC, 0
<do_this_if_high>
<do_this_if_low>
```

2.4. Assembleur ou langage d'assemblage PIC18

```
main:   movlw    7
        movwf    data_address_in_data_memory
        movwf    TRISA
        goto     main
        return
```

L'assembleur (assembly) ou langage d'assemblage (assembly langage) est le langage de programmation de plus bas niveau sur la machine. Il est la conversion directe lisible par l'homme (mode texte) du programme exécutable par le CPU de la machine (code binaire). Il est de ce fait, le langage le moins universel au monde, car dépendant du jeu d'instructions supporté par le CPU cible. Entre les marchés des ordinateurs et des systèmes embarqués, un grand nombre de fabricants implémentent des modèles d'exécution CPU (RISC ou CISC, Von Neumann ou Harvard, 8-16-32-64bits, entier voire flottant, scalaire voire vectoriel, VLIW ou superscalaire, etc) sur des technologies différentes (x86, x64, ARM, PIC18, RISC-V, C6000, etc). L'assembleur présenté ci-dessus est par exemple de l'assembleur 8bits PIC18 développé par la société Américaine Microchip pour leur propre gamme de MCU 8bits PIC18. Il s'agit d'un assembleur propriétaire contrairement au solutions Open Hardware RISC-V actuellement rencontrées sur le marché. Comme tout langage de programmation, un programme assembleur se lit de haut en bas, à l'image du modèle d'exécution séquentiel d'un CPU. Même si l'assembleur n'est pas universel, certaines représentations conceptuelles liées au langage peuvent néanmoins être généralisées :

```
label:    instruction    opérandes
```

- Label : un label ou étiquette, est une référence symbolique (simple chaîne de caractères) représentant l'adresse mémoire logique (emplacement) de la première instruction suivant celui-ci. Les labels sont remplacés par les adresses logiques voire physiques à l'édition des liens. Un label se termine par : afin de le différencier d'éventuelles directives d'assemblage voire d'instructions.

- Instruction : une instruction est un traitement élémentaire à réaliser par le CPU. Par exemple, charger (load) une donnée depuis la mémoire vers le CPU ou sauver (store) une donnée depuis le CPU vers la mémoire, réaliser une opération arithmétique ou logique, déplacer une donnée de registre à registre, réaliser un saut dans le programme, etc. L'ensemble des instructions exécutables par un CPU est nommé jeu d'instructions (ISA ou Instruction Set Architecture).

- Opérandes : les opérandes, lorsque l'instruction en utilise, sont les données ou les emplacements de données (registres ou adresses en mémoire principale) manipulées par l'instruction. Nous distinguons les opérandes sources, utilisées comme entrées avant l'exécution de l'instruction, de l'opérande de destination pour sauver le résultat. Les méthodes d'accès aux données en assembleur sont nommées des modes d'adressage :

    - Mode d'adressage registre : l'opérande est un registre CPU dans lequel est sauvé une donnée. *Ce mode d'adressage n'existe pas sur PIC18 car le CPU PIC18 n'intègre qu'un seul registre de travail, le registre W (WORK). Tous les autres registres du processeur MCU sont accessibles par adresse unique associée à chaque registre.*

    - Mode d'adressage immédiat : l'opérande est une constante dont la valeur sera sauvée dans le code binaire de l'instruction. *Par exemple, les instructions movlw 0 ci-dessus. MOVLW signifie MOV (déplacer) L (littéral ou immédiat, donc une constante) dans le registre CPU nommé W (WORK)*

    - Mode d'adressage direct (accès en mémoire donnée) : l'opérande est directement l'adresse de la case mémoire de la donnée. *Par exemple, les instructions movwf data_address_in_data_memory et movwf TRISA ci-dessus.*

    - Mode d'adressage indirect (accès en mémoire donnée) : l'opérande est l'adresse de la case mémoire de la donnée stockée indirectement dans un registre CPU.

2.5. My first MPLABX project from scratch

Ce premier exercice a pour objectif la prise en main des outils de développement, matériel comme logiciel. Nous réaliserons également des analyses bas niveau assembleur des programmes développés. Nous profiterons de ce premier TP d'analyse pour réaliser nos premiers développements de *drivers* (fonctions pilotes de périphériques) en découvrant les mécanismes de gestion des GPIO.

- Si ce n'est pas déjà fait, lire le document nommé *mcu-tp-1-prelude.pdf*
- Si ce n'est pas déjà fait, télécharger l'archive de travail *mcu.zip* sur la plateforme moodle. L'extraire à la racine de votre lecteur réseau école ou dans un répertoire dédié sur votre machine personnelle. Par exemple, ne pas l'extraire dans le répertoire téléchargement. Ordonnez dors et déjà vos développements, cela fait partie du métier d'ingénieur. Tous les développements et créations de projets de l'ensemble des travaux pratiques durant ce semestre se feront dans le répertoire *mcu/tp/disco* (disco comme discovery).

> Je répète une fois de plus le point précédemment cité car je le vois apparaître bien trop souvent chaque année. Aucun développement, ni aucun projet MPLABX n'a à être créé à l'extérieur du répertoire *mcu/tp/disco* durant cette trame de TP. Soyez vigilant sur ce point durant la création de vos projets sous l'IDE MPLABX. Ceci permet d'éviter une dispersion de vos développements durant votre processus de création et facilite le partage de projet logiciel dans une optique de travail collaboratif en équipe. Ce qui sera le cas en entreprise. Votre expérience future vous montrera que ce point peut faire gagner ou perdre des semaines voire des mois de développement sur des projets collaboratifs en ingénierie.

Durant ce premier exercice décorrélé du reste de la trame de TP, nous allons juste créer un programme simple *from scratch* (en partant de zéro) et manipuler quelques GPIO (broches). L'objectif étant juste de comprendre l'architecture minimal d'un programme C sur nos outils de développement (IDE MPLABX et toolchain C XC8) et notre architecture processeur (MCU 8bits PIC18F27K40).

- Créer un fichier *main.c* dans le répertoire suivant *disco/apps/fromscratch/main.c. Sous Windows :*
  - *Clic droit → Nouveau → Document texte → créer le fichier main.txt.*
  - *Aller dans l'onglet Affichage de la fenêtre courant de l'explorateur de fichiers → Afficher / Masquer → [x] Extensions de noms de fichiers*
  - *Modifier l'extension du fichier main.txt en main.c*
- Créer un projet MPLABX nommé *fromscratch* dans le répertoire *disco/apps/fromscratch* . Inclure le fichier *disco/apps/fromscratch/main.c* puis éditer un programme C minimal (cf. ci-dessous). S'aider des tutoriels dans *mcu/tp/doc/tutos*
  - *Compiler le projet : fenêtre Projects > clic droit sur le nom du projet > Clean and Build*

```
main(){}
```

- Modifier le programme de façon à allumer puis éteindre successivement dans un *while(1)* les trois LED D2, D3 et D4 de la carte *curiosity HPC*. Compiler et analyser l'erreur de sortie !

```
main(){
    TRISA = 0x00 ;
    while (1){
        LATA = 0x70 ;
        LATA = 0x00 ;
    }
}
```

- Résoudre l'erreur de compilation en ajoutant le fichier d'en-tête constructeur pour notre MCU

```
#include <pic18f27k40.h>
```

Nous allons maintenant configurer le cadencement de l'horloge de travail du processeur (horloge de référence). Chez Microchip sur processeur PIC, nous parlons de bits de configuration.

- Ouvrir la documentation technique du PIC18F27K40 (répertoire *disco/bsp/doc/datasheets*) et analyser le schéma structurel matériel du sous système d'horloge intégré sur la puce silicium du MCU : *Figure 4-1.Simplified PIC MCU Clock Source Block Diagram*

- Sous MPLABX, adapter la configuration du CPU de façon à travailler à vitesse maximale (64MHz) en utilisant le résonateur interne au MCU (technologie RC ou MEMS) et non externe sur carte (Quartz)

  - *Production → Set Configuration Bits → Modifier le champ CONFIG1L :*
    FEXTOSC = OFF et RSTOS = HFINTOSC_64MHz
  - *Cliquer sur Generate Source Code to Output*
  - *Copier/coller le retour de la console en en-tête de votre fichier source (cf. ci-dessous).*
  - *Compiler le projet : fenêtre Projects > clic droit sur le nom du projet > Clean and Build*

```
/* copy and paste bits configuration here ! */

#include <pic18f27k40.h>

...
```

Test sur carte physique Curiosity HPC

- Ouvrir les propriétés du projet : *fenêtre Projects > clic droit sur le nom du projet > Properties*
- Sélectionner la carte Curiosity HPC : *Hardware Tool > Microchip Kits > Curiosity > Apply > OK*
- Compiler et exécuter le programme : *fenêtre Project > clic droit sur le nom du projet > Run*

Test sur simulateur MPLABX IDE (sans carte de développement)

- Ouvrir les propriétés du projet : *fenêtre Projects > clic droit sur le nom du projet > Properties*
- Sélectionner le simulateur : *Hardware Tool > Simulator > Apply > OK*
- Compiler et exécuter le programme : *fenêtre Projects > clic droit sur le nom du projet > Debug*
- Arrêter la simulation en cliquant sur le bouton carré rouge STOP (ou [Shift] + [F5])

Test sur simulateur de carte PICSimLab

- Sous MPLABX, compiler le programme : *fenêtre Projects > clic droit sur le nom du projet > Clean and Build*
- Sous MPLABX, observer après compilation le dossier où a été sauvé le fichier .hex de sortie:

```
BUILD SUCCESSFUL
Loading code from /<your_computer_path>/dist/default/production/<your_project_name>.hex
```

- Ouvrir le simulateur de carte PICSIMLab : *File > Load Hex > charger le fichier .hex généré. Ne pas hésiter à utiliser le module oscilloscope pour analyser la sortie : Modules > Oscilloscope*

Vous venez de créer, de configurer et d'exécuter en partant de rien votre premier projet sur processeur PIC18. Vous avez ajouté le fichier d'en-tête constructeur afin de pouvoir utiliser les noms des registres du processeur dans votre programme. Puis, vous avez configuré la vitesse de travail du CPU afin de maîtriser la vitesse d'exécution des instructions par le CPU. Pour conclure, vous avez réalisé 3 techniques de test disponibles pour cette trame d'enseignement

2.6. Analyse en assembleur et debug

- Compiler puis charger le programme en mode *debug* dans le MCU cible sur carte curiosity HPC ou sur simulateur MPLABX. S'aider des tutoriels dans *mcu/tp/doc/tutos*
  - *Fenêtre Projects > clic droit sur le nom du projet > Debug*
- Ouvrir et déplacer dans l'environnement graphique de l'IDE une fenêtre permettant d'observer le code binaire désassemblé (conversion binaire vers assembleur PIC18 du firmware par l'IDE) :
  - *Window → PIC/target Memory Views → Program Memory*
- Parcourir la totalité de la mémoire flash (mémoire programme) et retranscrire le code binaire du programme assembleur générés ci-dessous en respectant le *mapping* (adresses) ou modèle mémoire choisi par le *linker* (éditeur de liens) pour placer les instructions du programme en mémoire.

Address	Binary Opcode	Label	Disassembly Listing

- Analyser et comprendre le code généré
- Ajouter une instruction assembleur *nop* (no operation) dans la boucle *while* du *main*. Compiler puis charger le nouveau programme en mode *debug*. Analyser le programme assembleur généré. Mettre un point d'arrêt sur le *nop*

```
asm("nop");
```

  - Point d'arrêt : *Double clic dans la fenêtre d'édition au numéro de ligne désiré*
  - Exécuter le programme pas à pas : *icône Step Into ou touche F7*
- En analysant la console de sortie de l'IDE (Build, Load,...) après compilation, préciser dans quel répertoire sur votre ordinateur a été rangé le *firmware* exécutable de sortie après édition des liens

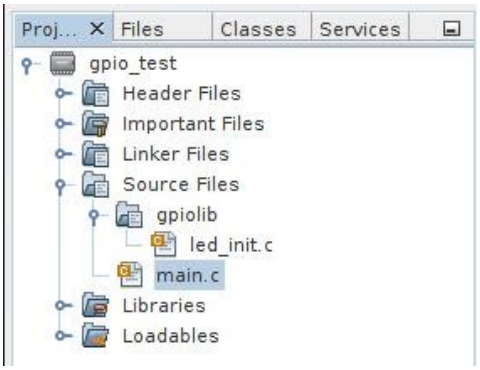
Voilà, ce premier exercice est maintenant terminé. Il n'avait pour objectif que de permettre une meilleure compréhension de l'architecture minimale d'un programme sous IDE MPLABX, d'utiliser les outils de debug pour tester vos programmes dans la suite de la trame et de comprendre le fonctionnement à l'étage assembleur d'un programme C élémentaire. A partir de maintenant nous allons travailler dans l'arborescence de fichiers créée spécifiquement pour notre BSP (Board Support Packag pour carte Curiosity HPC et MCU PIC18F27K40 – *mcu/tp/disco/bsp*). La conception du BSP étant déjà réalisée vous aurez à vous concentrer sur les phases de développement, test unitaire et test d'intégration de celui-ci.

2.7. BSP et fonctions pilotes C/ASM

- Créer un projet MPLABX nommé *gpio_test* dans le répertoire *disco/bsp/gpio/test/pjct*. Inclure le fichier *bsp/gpio/test/main.c* et s'assurer de la bonne compilation du projet. S'aider des tutoriels dans *mcu/tp/doc/tutos*

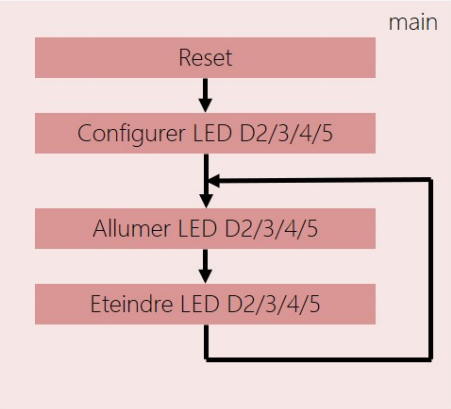
- Sur quelles broches du MCU sont physiquement connectées les LED D2, D3, D4 et D5 ? *S'aider de la documentation utilisateur (user guide) de la carte Curiosity HPC de Microchip présente dans mcu/tp/doc/datasheets*

- Créer un répertoire logique *gpiolib* dans le répertoire Sources Files du projet sous l'IDE (clic droit *New Logical Folder*). A réaliser et renommer pour chaque exercice. Ajouter le fichier *src/led_init.c* au projet et vérifier la bonne compilation de celui-ci.



- Modifier le fichier *led_init.c* afin de configurer les broches souhaitées en sortie. Modifier également le fichier d'en-tête *include/gpio.h* pour l'activation ou la désactivation des LED (développement de macro en langage C). *Cet exercice n'a pour objectif que de vous montrer différentes manières en langage C pour réaliser un même traitement.*

- Développer une application de test implémentant le diagramme de séquence suivant :

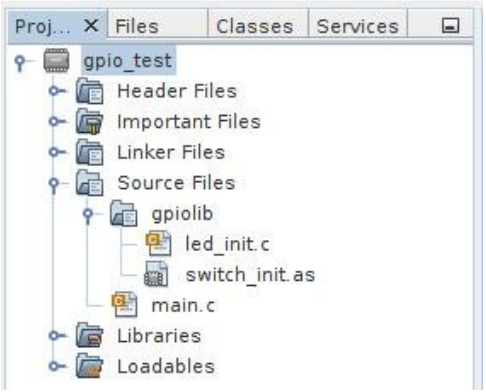


- Analyser le programme binaire généré. Observer à l'oscilloscope le signal en sortie du MCU envoyé à la LED D2. *Quelle solution (fonction ou macro) vous semble la plus efficace et pourquoi (exemple de question de recruteur en entretien d'embauche) ?*

2.8. Gestion des boutons poussoirs

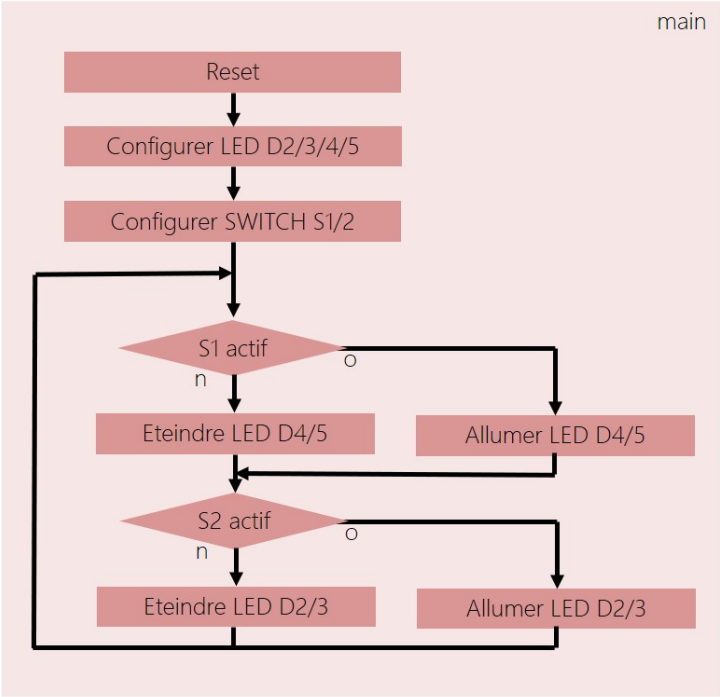
- Sur quelles broches du MCU sont physiquement connectés les boutons poussoirs S1 et S2 ? *S'aider de la documentation utilisateur (user guide) de la carte Curiosity HPC de Microchip présente dans mcu/tp/doc/datasheets*

- Ajouter le fichier assembleur *src/switch_init.as* au projet et vérifier la bonne compilation de celui-ci.



- Modifier le fichier assembleur *switch_init.as* afin de configurer les broches souhaitées en entrée. Modifier également le fichier d'en-tête *include/gpio.h* pour la lecture des états logiques sur les GPIO précédemment configurées (développement de macro en langage C). *Cet exercice n'a pour objectif que de vous montrer différentes manières en langage C pour réaliser un même traitement. Il vous faudra également configurer le registre ANSELB. Quel est son rôle ?*

- Développer une application de test implémentant le diagramme de séquence suivant :



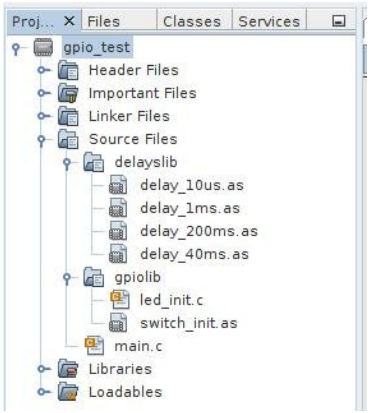
- Valider le bon fonctionnement de votre application

2.9. Délais logiciel

Dans une application, nous cherchons tant que possible à éviter les temporisations et délais logiciels. Il s'agit de boucles de décrémentation voire d'incrémentation exécutées par le CPU correspondant à des fonctions d'attentes. Nous préférons dédier le CPU à la supervision du système (application) ou aux opérations de calcul (algorithmique). Le reste du temps, le système doit être au repos (veille) pour des considérations énergétiques. Nous verrons la mise en veille du CPU dans le prochain exercice sur les interruptions et les modules périphériques Timers. Néanmoins, dans certains cas, comme les phases d'initialisation de certains périphériques au démarrage, ces fonctions de délais ou temporisations logicielles peuvent être utiles. Nous en verrons des applications par la suite.

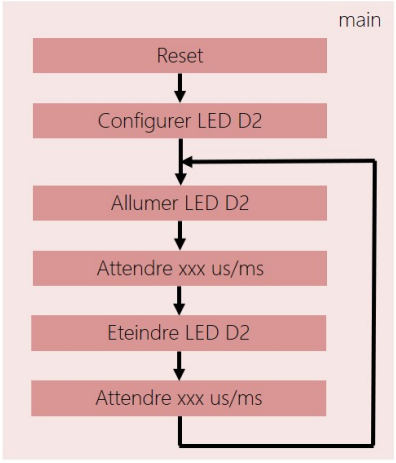
- Quelle est la fréquence de travail du CPU ? Quel fichier d'en-tête de notre BSP devrait-on modifier pour jouer sur ce paramètre (le chercher et spécifier son nom) ?

- Créer un répertoire logique *delayslib*. Ajouter les fichiers *bsp/common/*delay_*xxx.as* au projet et vérifier la bonne compilation du projet.



- Modifier le fichier *delay_10us.as* afin de réaliser une temporisation logicielle en assembleur de 10us. *S'aider de documentation du MCU PIC18F27K40 (chapitre 36 – Instruction Set Summary) et notamment de l'instruction decfsz.*

- Développer une application de test implémentant le diagramme de séquence suivant :



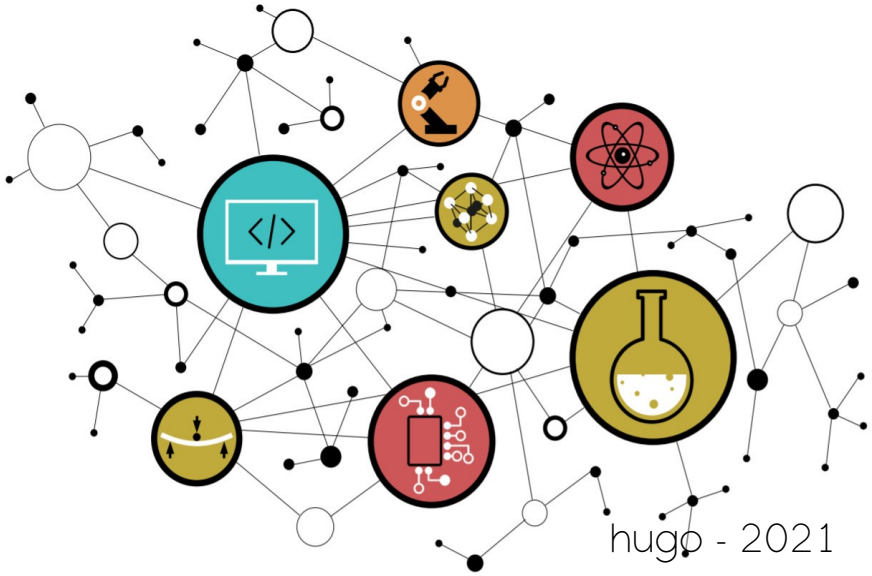
- Valider les durées de temporisation à l'oscilloscope puis réitérer le travail pour chaque fonction d'attente *delay_1ms*, *delay_40ms* et *delay_200ms*

- Quel est le pourcentage de charge CPU (durée de travail du CPU par unité de temps) ?


notes

TRAVAUX PRATIQUES

MODULE DE COMPTAGE TIMER ET GESTION DES INTERRUPTIONS



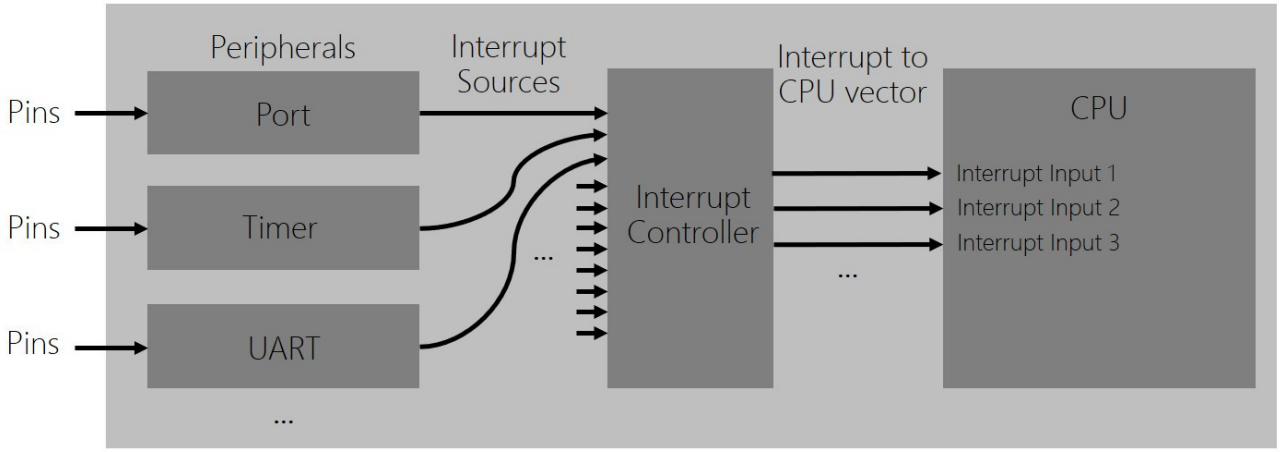
hugo - 2021

SOMMAIRE

MODULE DE COMPTAGE TIMER ET GESTION DES INTERRUPTIONS

3. MODULE DE COMPTAGE TIMER ET INTERRUPTIONS

3.1. Interruption matérielle



Le concept d'interruption matérielle est essentiel sur tout processeur conçu autour d'un CPU. Il s'agit de la capacité des fonctions périphériques à interrompre le CPU en cours de traitement pour lui affecter une nouvelle mission potentielle. Une interruption matérielle sera toujours rattachée à une fonction matérielle périphérique et correspond à l'occurrence d'un événement physique dans le système. Elle est toujours envoyée par un périphérique vers le CPU. Une fois configuré pour une mission dédiée, un périphérique est un composant indépendant et autonome dans le processeur lui-même. Chaque périphérique possède donc un rôle et une tâche spécifique dans le système (communiquer, convertir ou traiter).

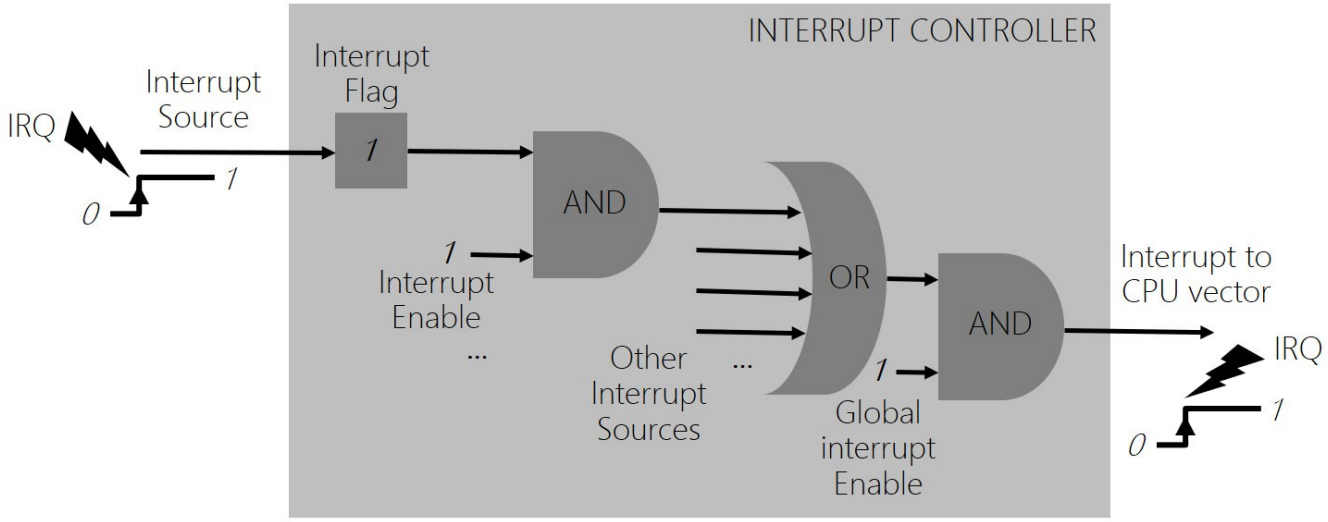
Un module périphérique Timer est par exemple conçu autour d'un compteur ou décompteur numérique. Sa mission est de compter à la place du CPU. Les modules UART, SPI, I2C ou USB sont par exemple des périphériques de communication permettant d'échanger de l'information avec l'extérieur du processeur. Communication de machine à machine. Lorsqu'un événement physique relatif à la tâche d'un périphérique se produit (fin de comptage pour un Timer, réception ou fin d'émission d'une donnée pour un UART, etc), celui-ci va tenter d'interrompre le CPU pour le prévenir de cet événement. Une interruption est alors envoyée par le périphérique vers le CPU. Une interruption est un simple signal physique électrique booléen tout ou rien. A la réception de l'interruption, le CPU sera soit en cours d'exécution d'une instruction (*fetch/decode/execute/writeback*), soit en veille (*inactif*). Pour que le CPU soit sensible et donc voit l'interruption, le développeur aura à configurer le contrôleur d'interruption du processeur. Nous allons découvrir tout ceci dans les pages qui suivent.

Une interruption est donc un signal physique partant d'un périphérique et arrivant (sous conditions) jusqu'au CPU. Les entrées d'interruption d'un CPU sont le plus souvent classées par niveau de priorité. Il est à noter qu'une ''Belle'' application, ne travaillera et ne réalisera des actions qu'au moment opportun, lorsqu'un traitement est strictement nécessaire. Le reste du temps, le système de supervision devra rester au repos (veille). La mise en veille CPU correspond à sa capacité à ne pas exécuter d'instruction (CPU inactif). Alors, seuls les périphériques restent à l'écoute des événements extérieurs (voire intérieurs) au système. Ils servent donc d'interfaces entre le système embarqué et son environnement physique extérieur.

3.2. Source et requête d'interruption IRQ

Dans la majorité des cas, il existe au moins autant de sources d'interruption que de périphériques dans le processeur. Une source d'interruption est un signal physique unidirectionnel (conducteur sur le die) allant d'un périphérique au contrôleur d'interruption. Une interruption (par abus de langage) ou requête d'interruption ou IRQ (Interrupt Request) correspond au passage d'un état logique inactif à actif sur une source d'interruption. Un périphérique envoie alors une requête au CPU et demande à interrompre son traitement en cours afin de lui affecter une nouvelle mission. Faut-il encore que le CPU y soit sensible !

3.3. Logique de démasquage d'interruption



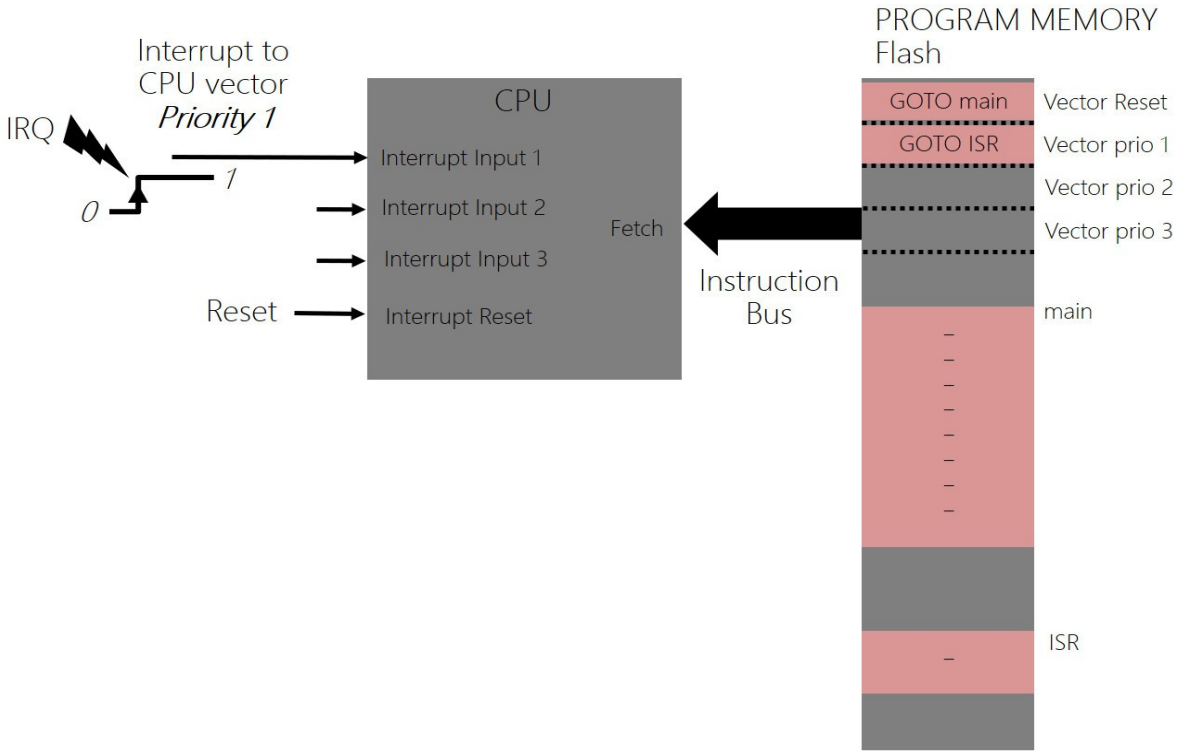
Il est à noter que par défaut à la mise sous tension, généralement le CPU n'est sensible à aucune source d'interruption, hormis celle du reset (bouton poussoir externe voire reset logiciel). Même si des périphériques devaient envoyer des IRQ (Interrupt Request), ils ne pourraient interrompre l'exécution du programme en cours. En effet, le développeur en charge du développement devra démasquer les sources d'interruption une à une en fonction des besoins spécifiques de l'application. Il s'agit d'une logique combinatoire de démasquage à réaliser afin de configurer le contrôleur d'interruption (cf. Interrupt Controller ci-dessus).

Comme pour tous les périphériques, cette configuration se fera par écriture dans des registres. Il y aura en général à minima la validation de la source d'interruption et la validation globale des interruptions pour l'ensemble du processeur. De même, les requêtes d'interruption ou IRQ seront mémorisées (cf. interrupt flag ci-dessus – simple bascule D) et auront à être acquittées explicitement par mise à zéro dans l'application par le développeur. Ces acquittements se feront dans les fonctions d'interruption (ou ISR) et seront présentés par la suite.

Si une IRQ parvient à traverser le contrôleur d'interruption, elle forcera le CPU à arrêter les traitements en cours et à exécuter un code spécifique (vecteur d'interruption). La commutation est matériellement câblée dans le CPU. Il est souvent associé un niveau de priorité (voire de sous priorités) à un vecteur d'interruption. Ceci permet à l'ingénieur de rendre plus au mois prioritaires des périphériques (Timer, UART, USB, Ethernet, etc) et donc les tâches à réaliser par l'application. Le problème se produira lorsque plusieurs périphériques enverront des requêtes simultanément.

3.4. Vecteur d'interruption

Un vecteur d'interruption est une ''petite'' zone en mémoire programme. Un vecteur d'interruption est donc chargé de mémoriser ''quelques'' instructions binaires. Ces instructions permettent une redirection vers la fonction d'interruption ou ISR (Interrupt Service/Software Routine). D'où le nom de vecteur. Les ISR sont quant à elles des fonctions logicielles accessibles depuis l'application et présentent dans le programme en cours de développement. Il est potentiellement possible d'avoir autant d'ISR que de vecteurs d'interruption dans une application. Rappelons que généralement il est associé un niveau de priorité à un vecteur d'interruption et donc aux ISR liées. Ces priorités sont également configurables par registres.



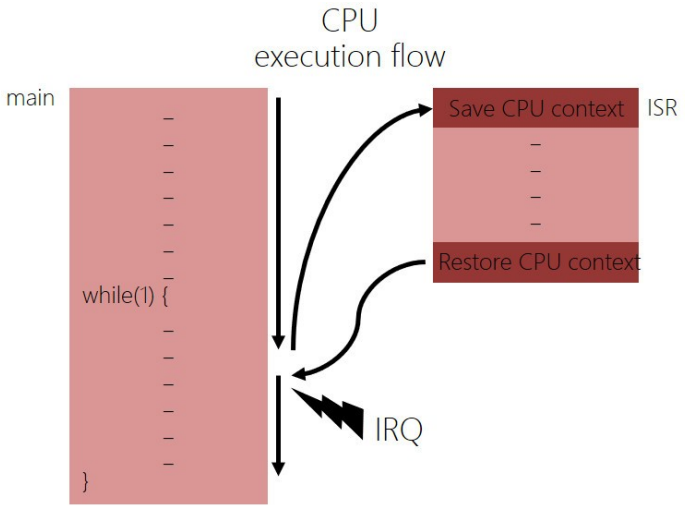
Attention, le schéma ci-dessus représente deux concepts radicalement différents (cf. version numérique du document) :

- Architecture matérielle (hardware) en gris pour les fonctions électroniques matérielles (CPU et mémoire programme seulement) et les flèches noires pour les bus et conducteurs physiques
- Micrologiciel (firmware) en rose correspondant au code binaire utile généré suite au processus de compilation et d'édition des liens sur ordinateur

Le firmware est donc mémorisé en mémoire programme non-volatile, souvent nommée mémoire flash sur MCU. Il s'agit de l'application logicielle embarquée dans le système matériel. Il s'agit de la mission du système embarqué répondant à un besoin. Sans système d'exploitation (scheduler), si aucun périphérique ne cherche à interrompre le CPU, alors celui-ci n'exécutera que du code de fonctions appelées depuis la fonction main. Dans l'exemple ci-dessus, dès que le CPU capte l'IRQ sur son entrée d'interruption de priorité 1, il commence à aller chercher puis exécuter le code présent dans le vecteur d'interruption de priorité 1 (simple instruction GOTO ISR). Puis par redirection le CPU ira exécuter le code de la fonction ISR associée. Dès que l'ISR est terminée, il reprend l'exécution de l'application exactement à l'instruction à laquelle il avait été interrompu par l'IRQ. Nous nommons ce phénomène commutation de contexte (sauvegarde et restauration). Il sera illustré sur PIC18 par la suite dans cet enseignement. De même, les vecteurs d'interruption sont généralement situés aux adresses les plus basses de la mémoire programme et sont sur certaines architectures translatables à d'autres adresses mémoire. L'ensemble des vecteurs d'interruption est nommée table des vecteurs d'interruption.

3.5. Fonction d'interruption ISR

Une ISR (Interrupt Service/Software Routine) ou fonction d'interruption est un fonction logicielle telle que vous avez pu en rencontrer en langage C dans toute application. A ceci près, qu'elle ne doit jamais être appelée de façon explicite depuis l'application. Il s'agit de programmation événementielle. Les ISR se réveilleront de façon asynchrone (non prédictible lorsqu'il s'agit de périphériques de communication) sur événement matériel en suivant la logique précédemment présentée (IRQ, démasquage puis vecteur d'interruption). Une ISR possédant le plus haut niveau de privilège d'exécution sur un processeur à CPU (MCU, AP, GPP, DSP, MPPA, etc), il faut toujours penser à passer le moins de temps possible à l'intérieur en déportant les traitements longs dans les fonctions appelées depuis le main (code applicatif). Sans système d'exploitation, utiliser alors des variables globales pour les échanges.



En respectant le code couleur précédemment utilisé (cf. version numérique pdf), les concepts maintenant présentés sont représentatif du logiciel développé par le développeur et du firmware embarqué. Sans système d'exploitation, les développements sont alors nommés *Baremetal,* soit nu directement sur le processeur sans système logiciel exploitant la machine (scheduler pour le CPU, gestionnaire mémoire par MMU/MPU pour la mémoire, drivers ou pilotes pour les périphériques, etc). L'application doit alors impérativement implémenter un while(1). Nous verrons en travaux pratiques comment développer une application en implémentant un *scheduler offline*.

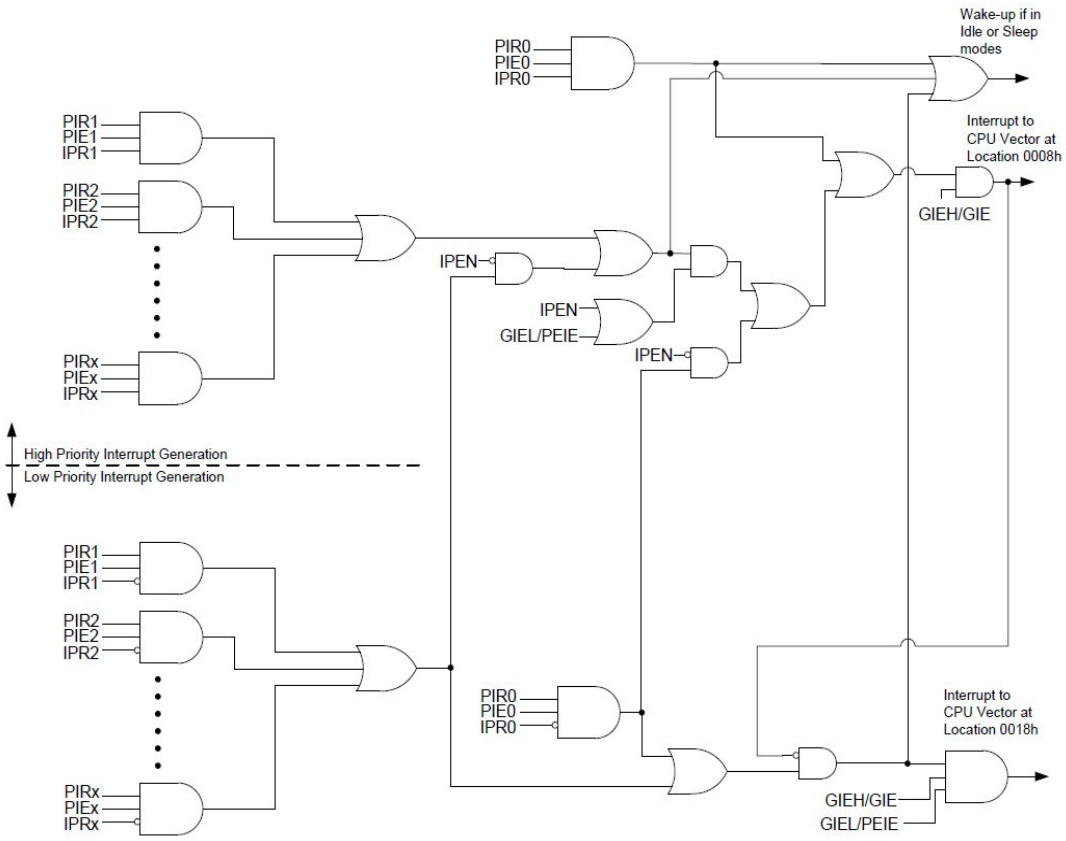
Les ISR étant spécifiques à une architecture, leur déclaration sera différente d'un processeur à un autre et donc d'une chaîne de compilation à une autre. Le plus souvent, un qualificateur de fonction lui est associé. Le compilateur a besoin de ce qualificateur afin d'implémenter les sauvegardes et restaurations de contexte en en-tête et pied de fonction. L'exemple ci-dessous illustre une déclaration d'ISR de priorité haute sur PIC18 en langage C sur toolchain XC8. Pour information, il y a seulement deux niveaux de priorité (haute et basse) et donc deux vecteurs d'interruption sur PIC18 (en plus de celui du reset).

```
void main (void)
{
   _
   while(1) {
      _
      _
   }
}
```

```
void __interrupt(high_priority) ISR (void)
{
   _
   _
   _
}
```

Vous constaterez que le code C ainsi que le schéma ci-dessus utilisent deux illustrations différentes pour présenter un même concept (texte vs graphique). L'une est une implémentation technologique en langage C et l'autre une représentation conceptuelle sous forme de dessin. La majorité des exercices demandés dans la trame de travaux pratiques seront représentés graphiquement de façon générique. Il sera à votre charge de réaliser les implémentations technologiques C ou assembleur sur PIC18 sous environnement de développement MPLABX et chaîne de compilation XC8.

3.6. Gestion des interruptions sur PIC18



Les MCU 8bits PIC18 de Microchip ont tous en commun le même CPU, les mêmes bus, et donc le même jeu d'instructions assembleur (ISA) ainsi que la même chaîne de compilation (XC8). Il existe néanmoins un très large choix de PIC18 différents au catalogue de Microchip. Il sont tous différenciés par leurs ressources mémoire (Flash et SRAM) ainsi que par leur jeu de périphériques (UART, SPI, I2C, USB, CAN, etc). Un PIC18 intègre en général un large jeu de périphériques et chaque périphérique possède une voire plusieurs sources d'interruption. Il existe donc un certain nombre de registres de configuration pour les interruptions. Nous pouvons classer tous ces registres 8bits de configuration des interruptions sur PIC18 en 4 sous familles :

- INTCON : *Configuration globale pour l'ensemble du système (bits GIEH, GIEL, IPEN, etc)*
- PIEx (x=0 à 7) : *Configuration des bits de validation ou démasquage (interrupt enable)*
- PIRx (x=0 à 7) : *Lecture et acquittement des bits d'interruption IRQ (interrupt flag)*
- IPRx (x=0 à 7) : *Configuration des bits de priorité (ISR priorité basse ou haute)*

De même, pour chaque périphérique, 3 bits seront alors à configurer (xxx dépend du périphérique à configurer). Ces bits correspondent à des champs de bits dans les registres précédemment cités.

- xxxIE : *Interrupt Enable (validation de l'interruption afin de rendre le CPU sensible à l'IRQ)*
- xxxIF : *Interrupt Flag (mémorisation de l'IRQ pour acquittement dans l'ISR)*
- xxxIP : *Interrupt Priority (configuration du vecteur d'interruption de priorité basse ou haute)*

L'exemple suivant présente une configuration d'interruption en assembleur pour le Timer0 :

```
; clear flag, enable and set low priority interrupt

BCF        PIR0, TMR0IF
BSF        PIE0, TMR0IE
BCF        IPR0, TMR0IP
```

```
; global interrupt enable for all MCU system

BSF        INTCON, IPEN
BSF        INTCON, GIEL
BSF        INTCON, GIEH
```

Program memory mapping of PIC18F27K40

Observons ci-dessous un extrait de datasheet présentant l'organisation de la mémoire programme d'un PIC18F27K40, soit le MCU utilisé en TP. Nous pouvons également constater les emplacements et tailles des 3 vecteurs d'interruption des MCU PIC18 :

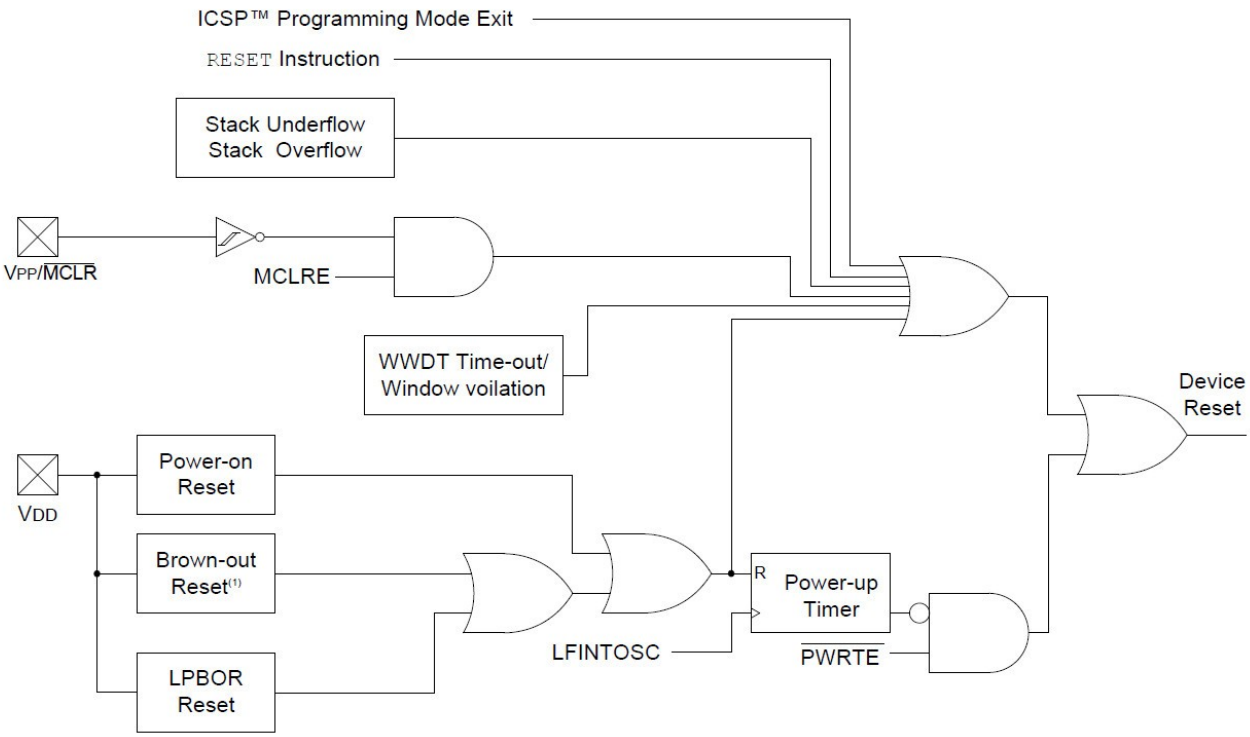
- Adresse 0x000000 : *reset*
- Adresse 0x000008 : *vecteur de priorité haute*
- Adresse 0x000018 : *vecteur de priorité basse*

Ces emplacements sont les mêmes pour tous les PIC18 de Microchip du marché. Chaque vecteur d'interruption ne propose que quelques octets de stockage. Pour information, un MCU PIC18F27K40 possède 128ko de mémoire programme Flash (ci-dessous 64KW soit 64KWords - 1Word=2octets sur PIC18) et 1ko de mémoire donnée non-volatile EEPROM. Ces deux technologies de mémoire sont non-volatiles et à ne pas confondre avec la mémoire donnée volatile de technologie SRAM (4Ko sur PIC18F27K40)

<table>
<tr><th rowspan="2">Address</th><th colspan="5">Device</th></tr>
<tr><th>PIC18(L)Fx4K40</th><th>PIC18(L)F25/45K40</th><th>PIC18(L)F65K40</th><th>PIC18(L)Fx6K40</th><th>PIC18(L)Fx7K40</th></tr>
<tr><td>Note 1</td><td colspan="5">Stack (31 Levels)</td></tr>
<tr><td>00 0000h</td><td colspan="5">Reset Vecor</td></tr>
<tr><td>...</td><td colspan="5">...</td></tr>
<tr><td>00 0008h</td><td colspan="5">Interrupt Vecor High</td></tr>
<tr><td>...</td><td colspan="5">...</td></tr>
<tr><td>00 0018h</td><td colspan="5">Interrupt Vecor Low</td></tr>
<tr><td>...</td><td colspan="5">...</td></tr>
<tr><td>00 001Ah to 00 3FFFh</td><td>Program Flash Memory (8 KW)</td><td>Program Flash Memory (16 KW)</td><td>Program Flash Memory (16 KW)</td><td>Program Flash Memory (32 KW)</td><td rowspan="3">Program Flash Memory (64 KW)</td></tr>
<tr><td>00 4000h to 00 7FFFh</td><td rowspan="4">Not Present(2)</td><td rowspan="4">Not Present(2)</td><td rowspan="4">Not Present(2)</td><td rowspan="3">Not Present(2)</td></tr>
<tr><td>00 8000h to 00 FFFFh</td></tr>
<tr><td>01 0000h to 01 FFFFh</td></tr>
<tr><td>02 0000h to 1F FFFFh</td><td>Not Present(2)</td></tr>
<tr><td>20 0000h to 20 000Fh</td><td colspan="5">User IDs (8 Words)(3)</td></tr>
<tr><td>20 0010h to 2F FFFFh</td><td colspan="5">Reserved</td></tr>
<tr><td>30 0000h to 30 000Bh</td><td colspan="5">Configuration Words (6 Words)(3)</td></tr>
<tr><td>30 000Ch to 30 FFFFh</td><td colspan="5">Reserved</td></tr>
<tr><td>31 0000h to 31 00FFh</td><td colspan="2">Data EEPROM (256 Bytes)</td><td colspan="3" rowspan="2">Data EEPROM (1024 Bytes)</td></tr>
<tr><td>31 0100h to 31 01FFh</td><td colspan="2">Unimplemented</td></tr>
<tr><td>30 000Ch to 30 FFFFh</td><td colspan="5">Reserved</td></tr>
<tr><td>3F FFFCh to 3F FFDh</td><td colspan="5">Revision ID (1 Word)(4)</td></tr>
<tr><td>3F FFFEh to 3F FFFFh</td><td colspan="5">Device ID (1 Word)(4)</td></tr>
</table>

PIC18F27K40

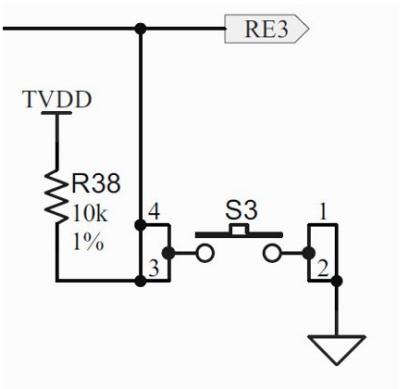
3.7. Gestion du RESET sur PIC18

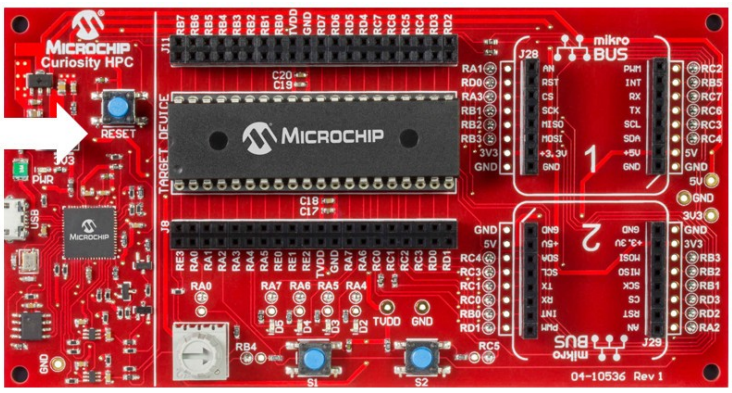


Nous pouvons observer ci-dessus la logique de démasquage du RESET sur PIC18F27K40. Par exemple, si un niveau logique bas est appliqué sur la broche externe RE3 nommée Vpp/$\overline{\text{MCLR}}$ (Master Clear) et si le bit de configuration MCLRE (MCLR Enable) est actif (fait par défaut à la mise sous tension), alors un RESET processeur est forcé.

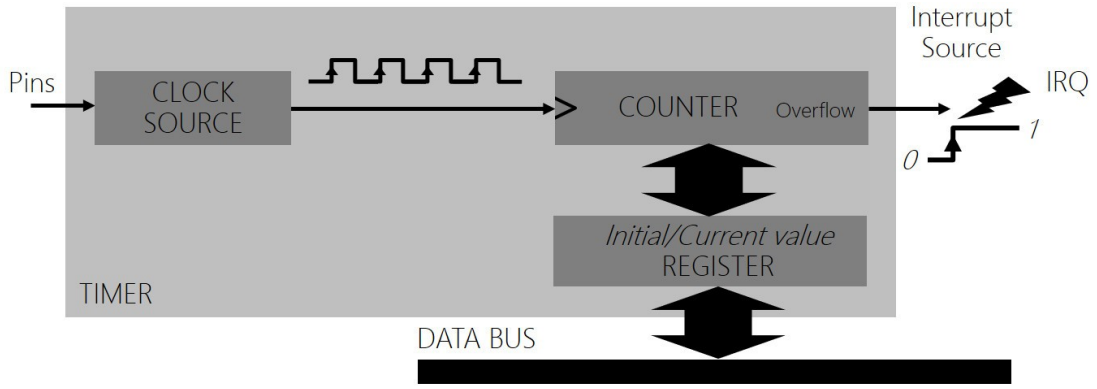
Le CPU exécute alors le code présent dans le vecteur d'interruption du RESET à l'adresse 0x000000 de la mémoire programme. En général, celui-ci appel la fonction main (en assembleur) ou la fonction de startup (en langage C) utilisée par défaut par la chaîne de compilation.

Observons par exemple ci-dessous le schéma électrique de câblage du bouton poussoir du RESET sur la carte Curiosity HPC de Microchip utilisée en TP. La broche Vpp/$\overline{\text{MCLR}}$ est la broche RE3 sur le boîtier du processeur. Il est courant sur un processeur qu'une broche ait différents noms et donc différents rôles possibles. Cela dépend de certaines fonctionnalités additionnelles associées à la plupart des broches. Chaque rôle associé à une broche aura à être configuré par le développeur en fonction des besoins spécifiques de chaque application. Ces aspects seront abordés dans la suite des exercices. Toute fonctionnalité matérielle du MCU est bien entendu documentée dans la documentation technique du processeur.





3.8. Module périphérique de comptage Timer



Un Timer sera toujours conçu autour d'un compteur ou décompteur numérique. Il est donc dédié aux opérations de comptage et le plus souvent à la gestion de bases de temps dans les applications (acquisitions de mesures physiques à intervalles de temps régulier, tâches périodiques, etc). Quelque soit la technologie du Timer, nous pouvons jouer sur 3 éléments afin de configurer une référence temporelle :

- Fréquence/Période de comptage (horloge de référence du Timer – fonction matérielle Clock Source ci-dessus)
- Valeur initiale de comptage (fonction matérielle Initial/Current Value Register ci-dessus)
- Nombre de bits utilisés par le compteur 8-16-32-64 bits avant débordement (fonction matérielle Counter et overflow ci-dessus)

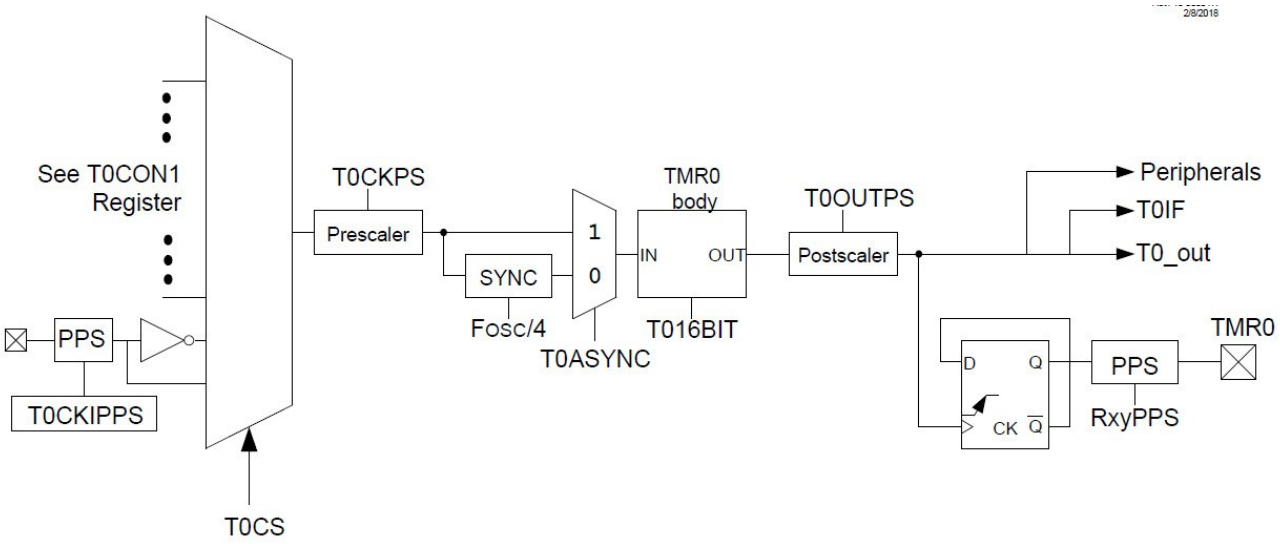
L'ensemble de ces trois éléments constituent en général un tout nommé Timer. Néanmoins, les Timers les plus élémentaires rencontrés sur processeur ne possèdent même pas de référence initiale de comptage ni d'horloge de référence configurable. Ils démarrent en partant de 0 à la mise sous tension et comptent à la fréquence de travail du CPU. Il sont souvent nommés TSC (Time Stamp Counter) et sont présents dans chaque CPU d'un grand nombre de processeurs du marché (Intel, AMD, TI C6000, ARM, etc). Ils sont souvent utilisés pour réaliser de la mesure de temps d'exécution de programme. Néanmoins, un Timer applicatif est plus riche en services matériels et fonctionnalités (comparateur numérique, rechargement automatique, module PWM, etc). Leur configuration peut sur certaines technologies être même relativement ardue.

Observons ci-dessous la configuration d'un Timer 8bits générique et calculons une valeur initiale de comptage à recharger après débordement afin d'obtenir une base de temps. Après comptage puis mise au niveau haut du bit de débordement (bit d'overflow), le Timer reprend le comptage à 0 par défaut (sans chargement de la valeur initiale). Par exemple sur 8bits ($255_{10} + 1_{10} = 1111\ 1111_2 + 1_2 = 1\ 0000\ 0000_2 = 256_{10}$ soit un résultat sur 9bits), le bit de débordement est le $9^{ième}$ bit actif. Exemple de calcul d'une valeur initiale de comptage sur Timer 8bits :

- Timer 8bis, comptage de 0 à 255 soit de 0x00 à 0xFF
- Hypothèse d'une horloge de référence de période 1ms
- Quelle serait la valeur initiale de comptage à charger au compteur afin d'obtenir un débordement après 100ms ?
- Réponse : *155 ou 0x9B (valeur à charger dans le registre initial de comptage)*
- Pourquoi : *Le Timer aura à compter 100 cycles de référence (100 x 1ms = 100ms) avant débordement au $256^{ième}$ cycle. Nous devons donc débuter le comptage à la valeur 155*

Sur un Timer, le signal à la source de l'interruption ou IRQ n'est autre que le bit de débordement (overflow flag). Ce signal électrique est alors envoyé au CPU (cf. chapitre Interruption).

3.9. Module périphérique Timer0 sur PIC18



Le MCU PIC18F27K40 utilisé à l'école intègre 4 Timers 16bits (Timer0, Timer1, Timer3 et Timer5) ainsi que 3 Timers 8bits (Timers2, Timer4 et Timer6). Il intègre donc 7 Timers pouvant être utilisés pour différents besoins dans une application.

Le schéma bloc ci-dessus ne présente que la structure interne du Timer0. Les autres Timers du PIC18F27K40 offrent d'autres services matériels complémentaires et sont donc différents. Le Timer0 est configurable en mode 16bits ou 8bits. Seul le mode 16bits est présenté ci-dessus et sera utilisé en TP. Petit exercice, entourer sur le schéma les fonctions matérielles suivantes :

- *Sous module compteur 16bits (COUNTER) ?*
- *Sous module de référence d'horloge (CLOCK SOURCE) ?*
- *Signal d'interruption IRQ envoyé au CPU par le Timer0 ?*

Ouvrir la datasheet des processeurs PIC18Fx7K40 et parcourir le chapitre 18 relatif au Timer0 (Timer0 Module) afin d'observer la configuration des registres. Le Timer0 possède deux registres 8bits de configuration (T0CON0 et T0CON1) et deux registres 8bits de chargement de la valeur initiale de comptage (TMR0L et TMR0H). Analysons le registre 8bits T0CON1 chargé de configurer l'horloge de référence.

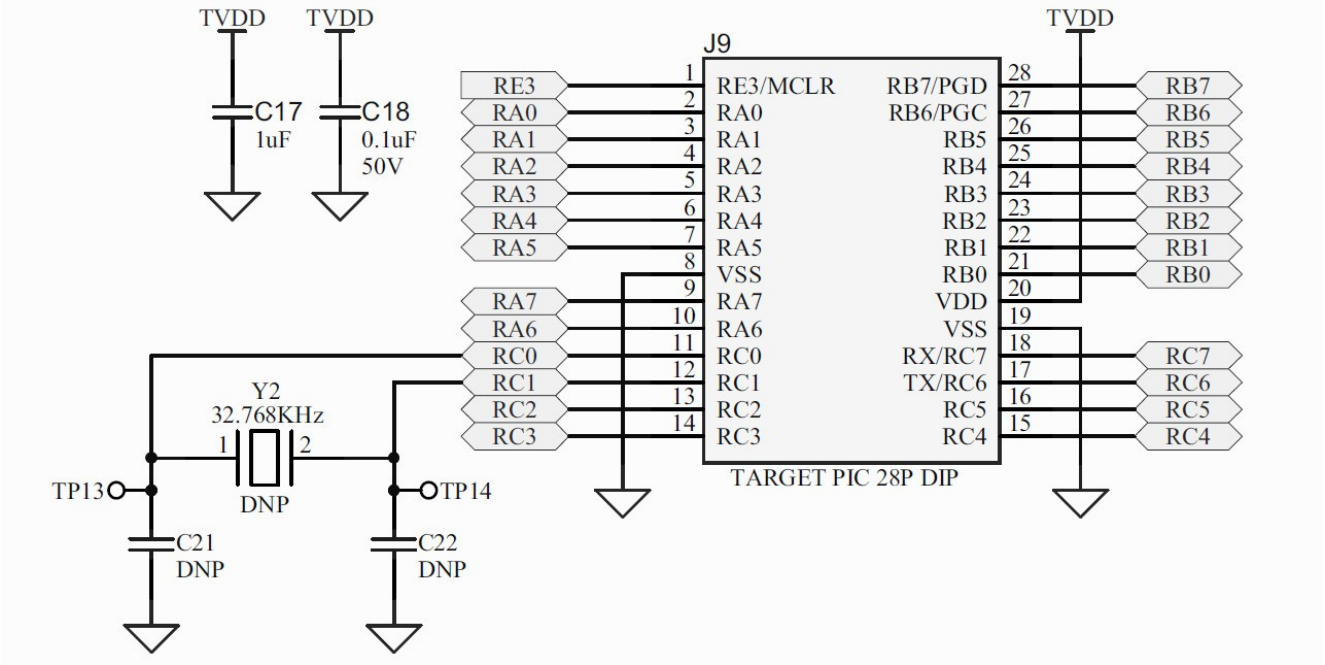
Nous pouvons observer sur le schéma ci-dessus que les champs T0CS (Timer0 Clock Source Select) et T0ASYNC (Timer0 Input Synchronization) permettent de piloter deux multiplexeurs d'aiguillage chargés de router la référence interne ou externe d'horloge. De même, le champ T0CKPS (Timer0 Clock Prescaler Select) permet de configurer un diviseur de fréquence (1:1, 1:2, 1:4, etc, 1:32768) avant d'entrer sur le compteur 16bits.

Name: T0CON1
Offset: 0xFD6

Timer0 Control Register 1

Bit	7	6	5	4	3	2	1	0
	T0CS[2:0]			T0ASYNC	T0CKPS[3:0]			
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

3.10. Configuration du Timer0 sur PIC18



Attention, la configuration proposée n'est pas celle demandée dans les TP. Elle sera donc à adapter. L'exemple suivant présente comment réaliser une base de temps très précise de 1 seconde. Il s'agirait d'une configuration de Timer si nous souhaitions réaliser une montre à quartz ou un réveil par exemple. En effet, les quartz 32.768KHz comme présenté ci-dessus sont typiquement ceux rencontrés dans les montres à Quartz. Les quartz possèdent une très faible dérive en fréquence par rapport à d'autres technologies de résonateurs (RC, MEMS, etc), typiquement proche de +/-10ppm (Particules Par Millions) soit +/- 0,00001% d'erreur et donc de précision.

Cet exercice pourrait d'ailleurs être réalisé sur la carte Curiosity HPC utilisée en TP à l'image de la capture du schéma électrique ci-dessus mais en utilisant néanmoins le Timer3 (broches utilisées par défaut sur la carte Curiosity HPC afin de relier le MCU au Quartz 32.768KHz).

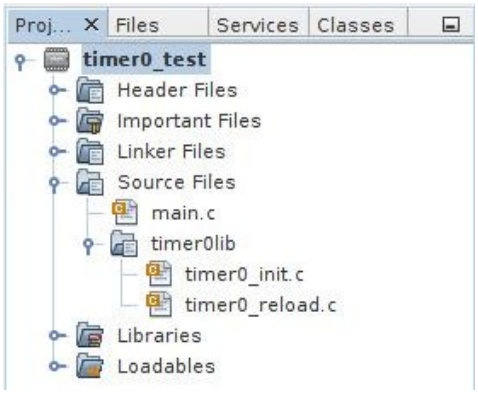
```
; Timer configuration :
; Timer disable, 16bits mode, post-scaler 1:1

MOVLW       0b00010000
MOVWF       T0CON0

; Timer configuration :
; Pins selection RC0 and RC1
; Extern 32.768KHz Quartz resonator
; No CPU synchro, pre-scaler 1:1

MOVLW       0b00110000
MOVWF       T0CON1

; Timer initialization :
; Initial decimal value 32767 (0x7FFF)
; Always write TMR0H before TMR0L

MOVLW       0x7F
MOVWF       TMR0H
MOVLW       0xFF
MOVWF       TMR0L
```

```
; Interrupt configuration :
; Clear flag, enable
; Set low priority interrupt

BCF         PIR0, TMR0IF
BSF         PIE0, TMR0IE
BCF         IPR0, TMR0IP

; Interrupt configuration :
; Global interrupt enable

BSF         INTCON, IPEN
BSF         INTCON, GIEL
BSF         INTCON, GIEH

; Timer enable
; The Timer start to count only now !

BSF         T0CON0, T0EN

; … This program generate an
interruption after 1s !
```

3.11. Configuration du Timer0 et interruption

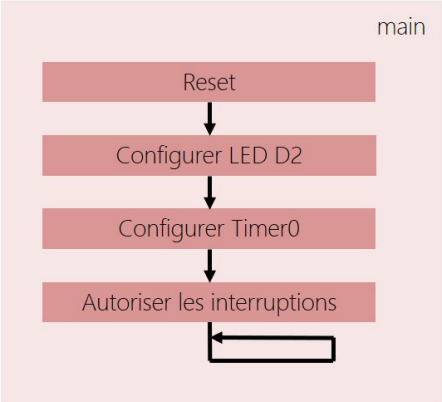
Nous allons nous intéresser au mécanisme de gestion des interruptions et de configuration de Timer. Nous nous efforcerons de gérer une base de temps en soulageant au maximum le CPU. Objectif, ne travailler qu'en cas d'absolue nécessité puis mettre le CPU en veille le reste du temps.

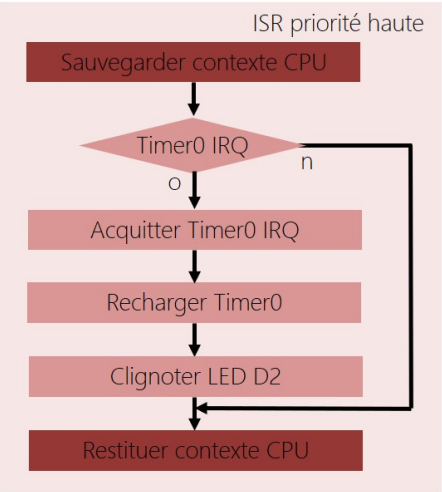
- Créer un projet MPLABX nommé *timer0_test* dans le répertoire *disco/bsp/timer0/test/pjct*. Inclure les fichiers *bsp/timer0/test/main.c, bsp/timer0/src/timer0_initc, bsp/timer0/src/timer0_reload.c* et s'assurer de la bonne compilation du projet. S'aider des tutoriels dans *mcu/tp/doc/tutos*



- Modifier les sources du projet afin de configurer le Timer0 pour qu'il puisse lever une interruption de priorité haute toutes les 20ms (horloge CPU Fosc à 64MHz donc Fosc/4 = 12MHz). *S'aider du fichier d'en-tête bsp/timer0/include/timer0.h et de la documentation (cartouche doxygen) de la fonction d'initialisation dans ce même header. Prendre l'habitude de parcourir les fichiers d'en-têtes durant la trame de TP et garder cette habitude pour le reste de votre vie dans le monde du développement logiciel. Les fichiers d'en-tête (headers) servent à générer les documentations techniques des bibliothèques logicielles. Ils précisent l'ensemble de l'API et services disponibles (Application Programming Interface).*

- Développer une application de test implémentant le diagramme de séquence suivant :





- Valider la base de temps à l'oscilloscope (période 40ms, 20ms LED allumée et 20ms LED éteinte)
- Estimer la charge CPU (durée de travail du CPU par unité de temps) ?

3.12. Analyse assembleur et commutation de contexte

- Compiler puis exécuter le programme en mode *debug* sur carte physique ou simulateur MPLABX
- Ouvrir et déplacer dans l'environnement graphique de l'IDE une fenêtre permettant d'observer le code binaire désassemblé
  - Window → PIC Memory Views → Program Memory
- Parcourir la totalité de la mémoire flash, identifier l'emplacement des fonctions (*main, timer0_init, timer0_reload et l'ISR*) et retranscrire en partie le code assembleur de l'ISR (seulement) ci-dessous. L'analyse peut commencer à être subtile, c'est normal.

Label	Disassembly Listing

- Comment se nomment les traitements observés en en-tête et pied d'ISR ? Quels sont leurs rôles ?

- Déclarer l'ISR comme une fonction C classique (sans qualificateur de fonction *interrupt*). Réitérer l'exercice d'analyse et observer le code binaire généré. Quels problèmes pourraient survenir dans d'autres applications si nous devions garder cette implémentation ?

3.13. Mise en veille du CPU

- Appeler l'instruction assembleur *sleep* dans la boucle infinie de la fonction principale. Configurer dans la fonction d'initialisation du Timer0 le champ IDLEN du registre CPUDOZE afin de valider le mode veille du CPU. Vérifier le bon fonctionnement de l'application.

```
asm(''sleep'');
```

- Expliquer à l'aide d'un chronogramme le fonctionnement de l'application. Quand le CPU exécute-t-il le code du main, le code de l'ISR et est-il en veille ?

- Estimer par le calcul la charge CPU (durée de travail du CPU par unité de temps) ?

- Quelles sont vos conclusions entre la fin de l'exercice 2 (avec délais logiciels) et maintenant (avec Timer, interruption et mise en veille), sachant que l'application réalisée d'un point de vu utilisateur est la même (clignotement d'une LED) ?

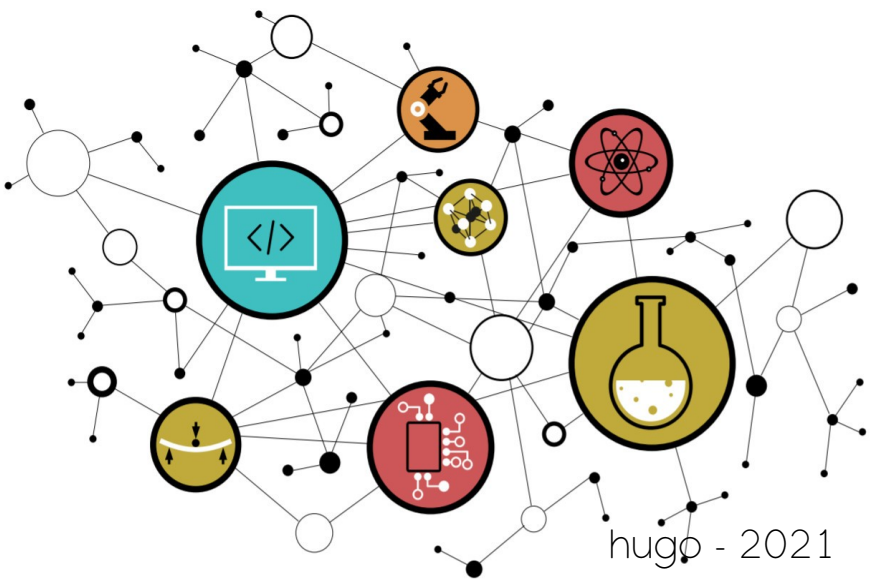
Une mise en veille correspond à une inactivité du CPU. Le CPU arrête donc sa machine séquentielle d'état (Fetch/Decode/Execute/WriteBack) et stoppe donc le processus d'exécution d'instructions. Seul un périphérique, et donc un événement physique interne ou externe, peut réveiller un CPU de sa veille. Dans cet exercice, il s'agit d'une IRQ envoyée par le Timer0 précédemment configuré. Mais par exemple sur votre ordinateur, en cas de mise en veille CPU, il s'agit le plus souvent d'une IRQ venant du contrôleur périphérique USB auquel est relié le clavier ou la souris.

Il est à noter qu'une mise en veille est toujours explicitement (directement ou indirectement) demandée depuis l'application ou le système d'exploitation. C'est donc au développeur d'estimer et de demander au système de se mettre au repos, comme il est de sa responsabilité de prévoir le mécanisme de réveil.


notes

TRAVAUX PRATIQUES

MODULE DE COMMUNICATION UART ET LIAISON SÉRIE

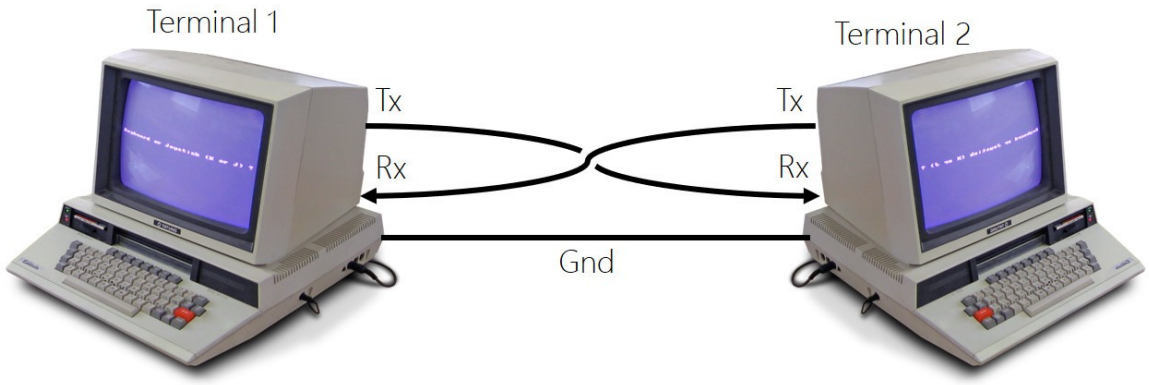


hugo - 2021

SOMMAIRE

MODULE DE COMMUNICATION UART ET LIAISON SÉRIE

4. MODULE DE COMMUNICATION UART



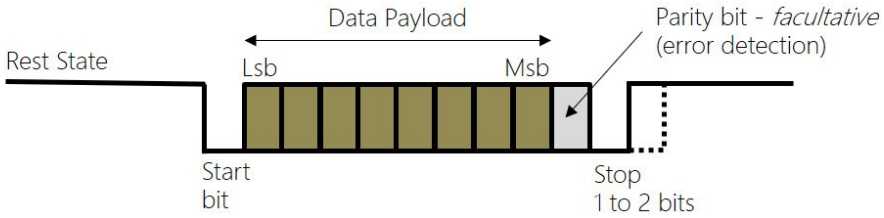
Le protocole de communication d'une liaison série asynchrone et les périphériques UART (Universal Asynchronous Receiver Transmitter) l'exploitant sont rencontrés depuis maintenant longtemps sur processeur numérique et ordinateur. A titre indicatif, la norme RS232 à été spécifiée en 1981 et n'a progressivement disparu sur ordinateur qu'à partir des années 2000 suite à l'arrivée de l'USB (norme USB1.0 Universal Serial Bus en 1996). Les modules de communications UART restent néanmoins encore très rencontrés en Systèmes Embarqués. Tout simplement car ils répondent toujours à un besoin (échanges bas débits de caractères en mode point à point. Il s'agit de communication Full Duplex (communication bidirectionnelle simultanée) utilisant 3 conducteurs physique. Les broches sont souvent nommées :

- Rx : Broche de réception croisée (toujours vu du récepteur). Tx vers Rx.
- Tx : Broche de transmission croisée (toujours vu du transmetteur). Tx vers Rx.
- Gnd : Fil de référence de masse (ground ou masse)

Les modules périphériques UART seront encore probablement utilisés très longtemps et sont bien connus des ingénieurs du domaine. L'UART est un périphérique spécialisé dans l'échange de caractères (données sur 8bits de façon générique) en topologie point à point entre 2 systèmes (peer to peer). Le tout avec des débits plutôt lents (de qq1KBps à qq100KBps) à notre époque (contrôle de procédé, test et prototypage, mesure, contrôle de module de communication, etc). Étant spécialisé dans l'échange de caractères, si un Homme cherche à interagir directement avec un périphérique UART depuis un ordinateur, celui-ci aura à configurer et utiliser un terminal asynchrone de communication (minicom, kermit, PuTTY etc sous GNU\Linux et TeraTerm, PuTTY, etc sous Windows). Rappelons de façon générique qu'un terminal ou une console est dédiée aux communications en mode texte par échange de phrases. Exemple ci-dessous d'interface de configuration sous terminal minicom sur système GNU\Linux.

```
+-----[configuration]------+
| Filenames and paths      |
| File transfer protocols  |
| Serial port setup        |
| Modem and dialing        |
| Screen and keyboard      |
| Save setup as dfl        |
| Save setup as..          |
| Exit                     |
| Exit from Minicom        |
+--------------------------+
```

4.1. Protocole de communication d'une liaison série asynchrone



Ne pas confondre l'UART (périphérique matériel) et le protocole de communication (technique d'échange d'information). Une liaison série asynchrone est l'un des rares derniers protocoles asynchrones de communication rencontré sur le marché. Cela signifie que l'émetteur n'envoie aucune information concernant le débit ou vitesse de transfert, à l'instar des SPI et I2C (conducteur physique d'horloge dédié) ou encore de l'USB et l'Ethernet (champs d'horloge en en-tête des trames assurant la synchronisation d'une PLL à la réception).

Sur une liaison série asynchrone, l'état au repos de la ligne de communication est l'état logique haut (état de repos ou rest state ci-dessus). Une trame débutera toujours par 1 bit de start (état logique bas). Suivent 7 ou 8bits de données utiles ou payload (charge utile), 1 bit de parité facultatif permettant une gestion élémentaire de détection d'erreur. Une trame se termine par 1 ou 2 bits de stop.

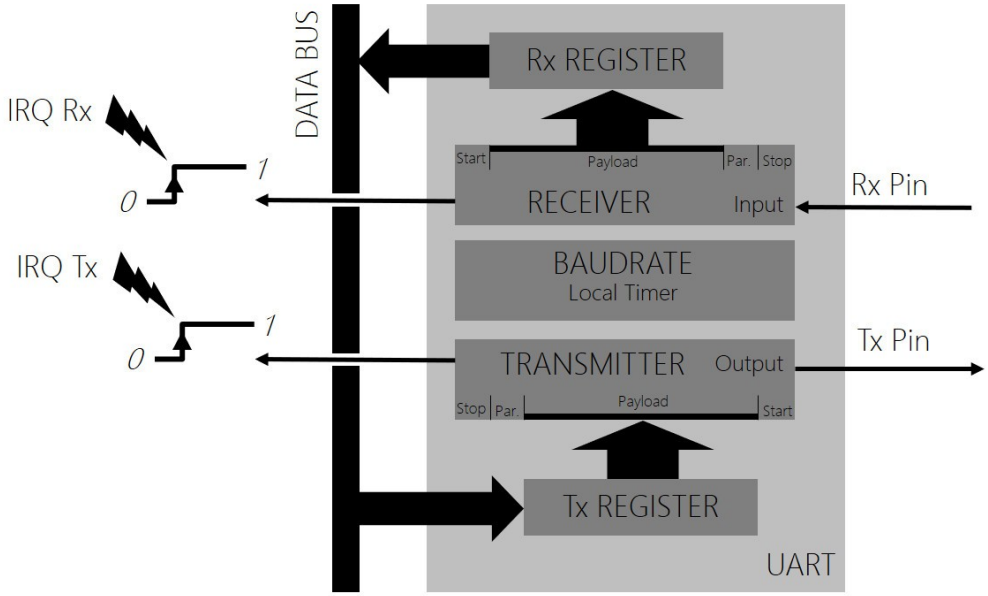
Néanmoins, la liaison série asynchrone est le plus souvent utilisée avec la configuration suivante : 1 bits de start, 8 bits de payload et 1 bit de stop (solution minimale et souvent suffisante pour du test). Une trame série complète d'informations est alors constituée de 10bits dont 8 bits utiles (différence entre débit réel et utile). Par exemple, avec un débit configuré à 9600Bd/s (Bps ou Baud/s ou symbole/s ou bits/s pour une liaison série asynchrone), l'envoi d'un bit dure environ 100us et donc l'envoi d'un caractère environ 1ms.

Durant une communication de module UART à module UART, physique ou logique, ne jamais oublier de bien configurer récepteur et émetteur au même débit et avec le même protocole de communication (nombre de bits de donnée, nombre de bits de stop, gestion du bit de parité et contrôle de flux). Exemple ci-dessous d'interface de configuration sous terminal minicom sur système GNU\Linux :

- Débits 9600 Bps
- 8 bits de payload
- Pas de bit parité
- 1 bit de stop
- Pas de contrôle de flux matériel ni logiciel

```
+-----------------------------------------------------------------------+
| A -    Serial Device      : /dev/ttyUSB0                              |
| B - Lockfile Location     : /var/lock                                 |
| C -   Callin Program      :                                           |
| D -  Callout Program      :                                           |
| E -    Bps/Par/Bits       : 9600 8N1                                  |
| F - Hardware Flow Control : No                                        |
| G - Software Flow Control : No                                        |
|                                                                       |
|    Change which setting?                                              |
+-----------------------------------------------------------------------+
```

4.2. Module périphérique UART

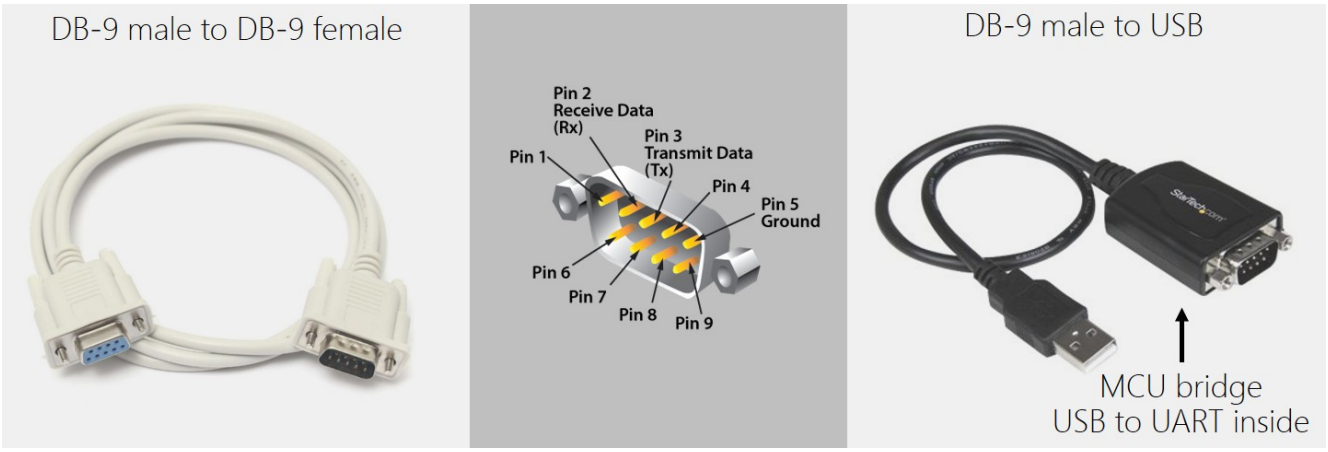


Un périphérique UART (Universal Asynchronous Receiver Transmitter) peut être vu comme deux périphériques dissociés. Un récepteur (receiver) et un transmetteur (transmitter). Néanmoins, les deux implémentent le protocole d'une liaison série asynchrone et leur configuration protocolaire est similaire (protocole et débit). Un Timer local dédié (Baurate ci-dessus) est souvent présent dans un UART afin de configurer le débit de communication.

Comme tout périphérique série de communication, le transmetteur et le récepteur sont conçus autour de registres à décalage. Le récepteur est chargé de récupérer les trames bit à bit dans son registre interne à décalage, de détecter d'éventuelles erreurs de transmission puis d'enlever l'enveloppe protocolaire de la trame pour ne récupérer que les données utiles (payload). A chaque nouvelle trame, la donnée utile est chargée dans un registre de travail (Rx register ci-dessus) afin d'être récupérée par le CPU et une requête d'interruption est envoyée au CPU (IRQ Rx) pour le prévenir de l'arrivée d'une donnée.

Le transmetteur fait quant à lui le travail inverse. L'application demande à envoyer une donnée par écriture dans le registre de transmission (Tx register ci-dessus, opération atomique de qq cycles CPU). Néanmoins, cela ne signifie pas que la donnée ait été transmise. Le transmetteur est alors responsable de charger la donnée utile dans son registre interne à décalage, de rajouter l'enveloppe protocolaire (start, stop voire parité) puis d'envoyer bit à bit les données en respectant le débit configuré. Une fois la donnée envoyée, une requête d'interruption est envoyée au CPU (IRQ Tx) pour le prévenir de la fin de transmission.

4.3. Norme RS232

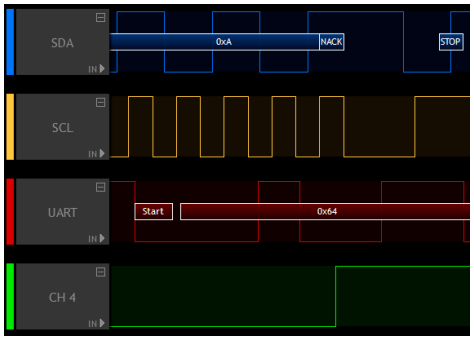


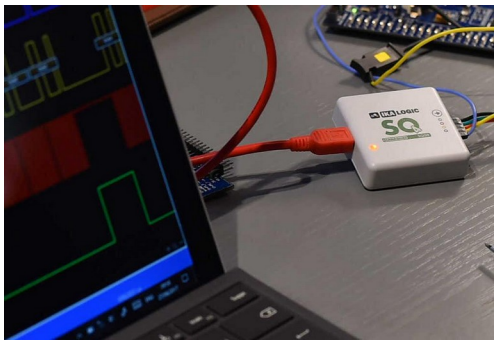
Ne pas confondre la norme RS-232 (nommée aussi EIA 232) avec le protocole d'une liaison série asynchrone et un périphérique UART. Cette norme est apparue en 1981 et a donnée naissance aux interfaces port COM sur ordinateur sous Windows (remplacé par l'USB depuis les années 2000). Cette norme de communication de machine à machine utilise des UART et implémente une liaison série asynchrone, mais tend à standardiser les débits de communication, les connecteurs et câbles associés, les longueurs de câbles, les niveaux de tension sur les conducteurs physiques, etc.

Cette norme standardise donc des contraintes et des limites physiques permettant de faciliter l'interfaçage et la communication de machines entre elles. Observons quelques unes de ces limitations :

- Longueurs de câbles en fonction des débits : 60m (2,4KBd/s), 15m (9,6KBd/s) et 2,6m (56KBd/s)
- Niveaux de tensions (logique inversée) : niveau logique 0 (de +3V à +25V) niveau logique 1 (de -3V à -25V)
- Technique de codage des bits : NRZ (Non Return to Zero)
- Types de connecteurs : DB9 (9 broches), etc

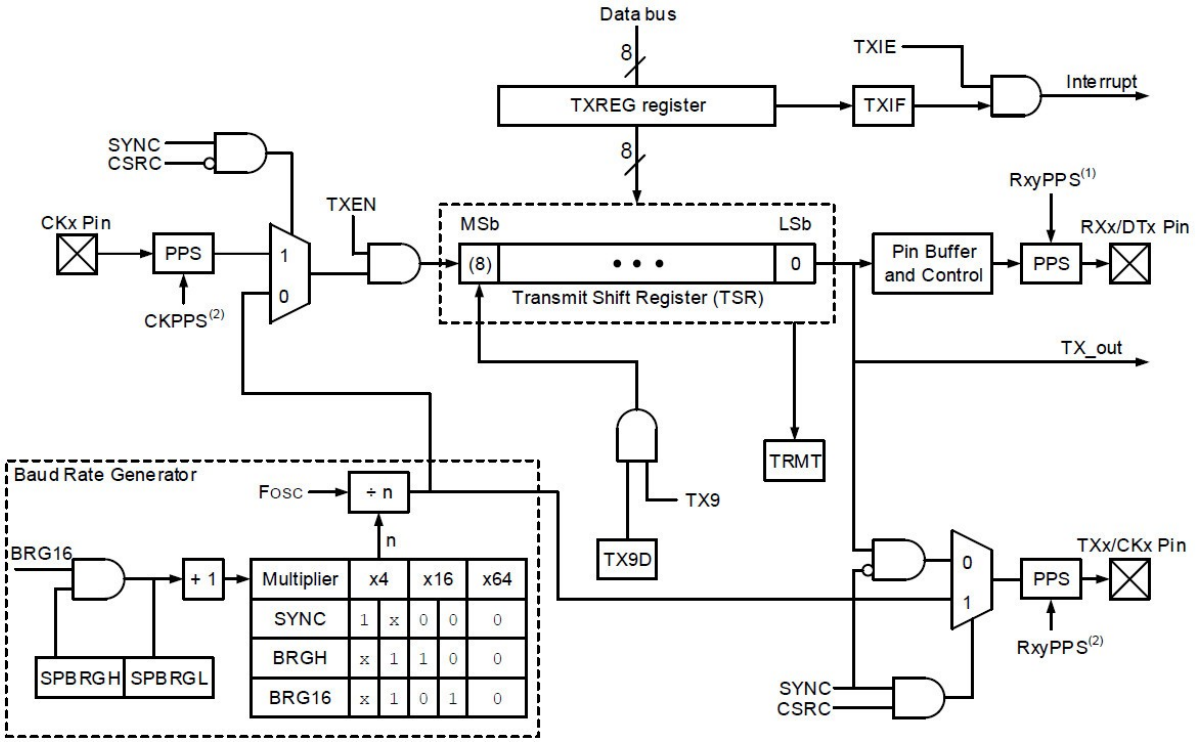
Dans certaines phases de debug et de test, nous pouvons être amenés à utiliser des outils matériels d'analyse des trames circulant sur les bus de communication externes au MCU (broches Tx et Rx pour une liaison série asynchrone). Nous utilisons pas exemple à l'école des solutions conçues en France par la société IKALOGIC ainsi que des options d'analyse proposés avec les oscilloscopes.





4.4. Module périphérique UART sur PIC18

Transmetteur UART liaison série asynchrone

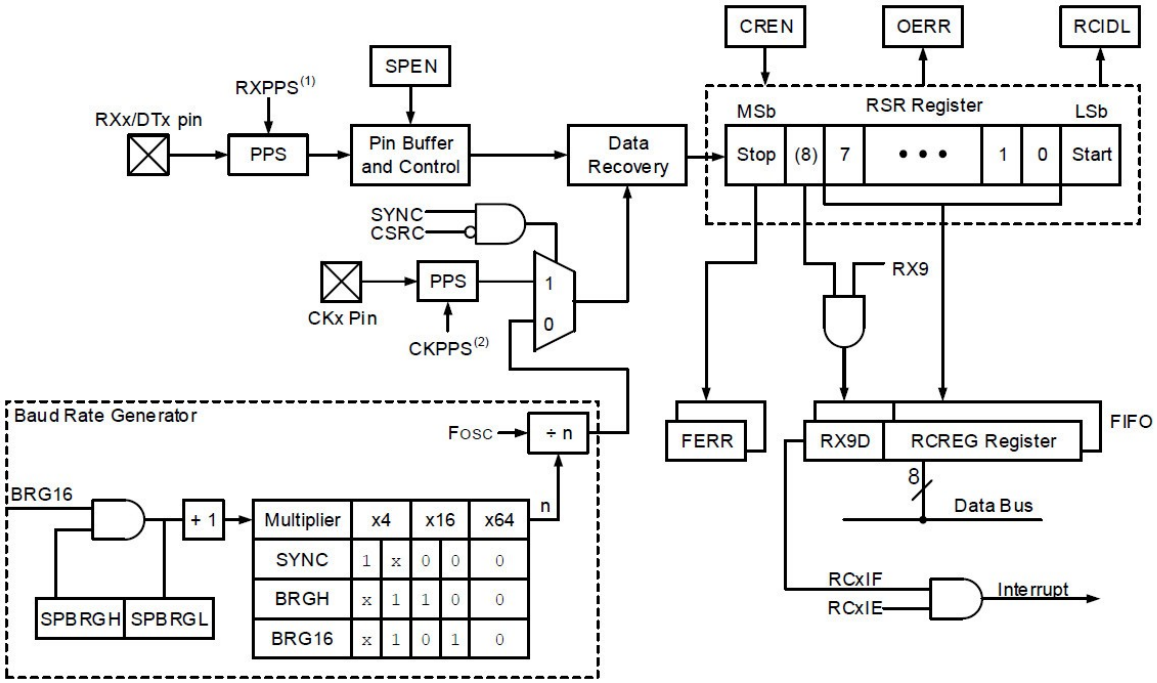


Le MCU PIC18F27K40 utilisé en TP intègre deux UART (UART1 et UART2). Les deux UART seront configurés et utilisés durant cet enseignement. Ces deux modules UART sont identiques. Seuls les noms des registres diffèrent (nommage avec indice 1 ou 2). Le schéma bloc ci-dessus présente la structure du transmetteur de ces UART. Petit exercice, entourer sur le schéma les fonctions matérielles suivantes :

- *Registre à décalage de transmission (envoi bit après bit cadencé sur l'horloge de référence)*
- *Registre de travail (pour écriture de la payload depuis l'application) permettant de charger le registre à décalage de transmission*
- *Ensemble assurant la référence d'horloge et donc le débit de la communication (Baud Rate Generator)*
- *Broche Tx de sortie*
- *Signal d'interruption IRQ envoyé au CPU (après envoie d'une information)*

Ouvrir la datasheet des processeurs PIC18Fx7K40 et parcourir le chapitre 27 relatif aux UART (EUSART Enhanced Universal Synchronous Asynchronous Receiver Transmitter) afin d'observer la configuration des registres. Prenons l'exemple de l'UART1, sachant que l'UART2 possède une stratégie de configuration et d'utilisation similaire. L'UART 1 possède 2 registres de configuration récepteur/transmetteur (RC1STA et TX1STA), 3 registres de configuration du débit à l'image de la configuration d'un Timer (BAUDCON1, SP1BRGH et SP1B1GL) et deux registres de travail pour la gestion des payload (TX1REG et RC1REG). Analysons une partie du registre 8bits TX1STA chargé de configurer le transmetteur. Nous pouvons observer sur le schéma ci-dessus que le champ TXEN (Transmitter Enable) permet d'appliquer la référence d'horloge au registre à décalage de transmission (TSR ou Transmit Shift Register) et autorise donc une transmission. Le champ SYNC (Synchronous) permet potentiellement d'utiliser une référence d'horloge externe, comme de sortir sur broche cette même référence et de transformer la communication asynchrone en communication synchrone.

Récepteur UART liaison série asynchrone



Le schéma bloc ci-dessus présente la structure du récepteur UART sur PIC18. Petit exercice, entourer sur le schéma les fonctions matérielles suivantes :

- *Registre à décalage de réception (récupération bit après bit de la trame de communication)*
- *Registres en FIFO (First In First Out) de travail pour la réception (pour lecture de la payload par l'application)*
- *Ensemble assurant la référence d'horloge et donc le débit de la réception (Baud Rate Generator)*
- *Broche Rx d'entrée*
- *Signal d'interruption IRQ envoyé au CPU (après réception d'une information)*

Ouvrir la datasheet des processeurs PIC18Fx7K40 et parcourir le chapitre 27 relatif aux UART (EUSART Enhanced Universal Synchronous Asynchronous Receiver Transmitter) afin d'observer la configuration des registres. Analysons une partie du registre 8bits RX1STA chargé de configurer le récepteur. Nous pouvons observer sur le schéma ci-dessus que le champ SPEN (Serial Port Enable) permet de valider ou pas la connexion physique de la broche au registre à décalage de réception. Nous pouvons également constater que le récepteur est chargé de détecter les erreurs. Si une erreur de communication est détectée, alors l'un des champs FERR (Frame Error) et OERR (Overrun Error) est activé par le récepteur. Dans tous les cas, si une erreur de communication est détectée ou constatée dans l'application, la meilleure stratégie reste de demander un renvoi à l'émetteur.

Receive Status and Control Register

Bit	7	6	5	4	3	2	1	0
	SPEN	RX9	SREN	CREN	ADDEN	FERR	OERR	RX9D
Access	R/W	R/W	R/W	R/W	R/W	RO	R/HC	R/HC
Reset	0	0	0	0	0	0	0	0

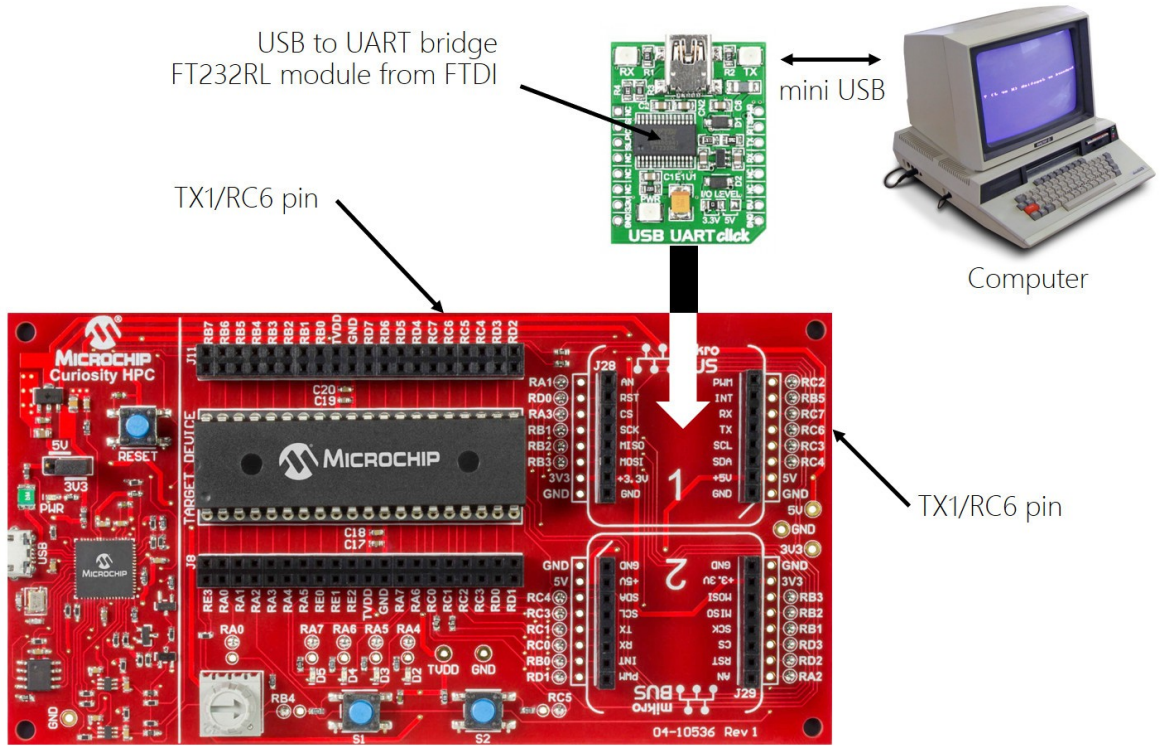
4.5. Configuration du module UART sur PIC18

Attention, le configuration présentée ci-dessous n'est pas celle demandée dans les TP. Elle sera donc à adapter. L'exemple suivant présente une séquence assembleur de code configurant le transmetteur de l'UART1 avec un débit de 115200Bd/s pour un horloge système de 64MHz. Une fois configuré, le caractère 'D' est transmis en respectant le protocole d'une liaison série asynchrone sur la broche TX1/RC6 du processeur. Le bit ou flag TRMT (TSR Register is Empty) est mis à jour par l'UART1 est permet de savoir si des bits restent présents dans le registre à décalage de transmission (registre TSR).

```
; UART1 Transmitter configuration :
; uart1 enable, asynchronous mode
; 8bits data, no parity, high baudrate

MOVLW       0b00100100
MOVWF       TX1STA

; UART1 BaudRate Generator configuration :
; 16bits baudrate mode
; baudrate 115.200KBps (64MHz CPU clock)

MOVLW       0b00001000
MOVWF       BAUDCON1
MOVLW       0x8B
MOVWF       SP1BRG
MOVLW       0x00
MOVWF       SP1BRGH
```

```
; send 'D' ASCII code
; 'D' = 68 = 0x44

       MOLW      0x44
       MOWF      TX1REG

; wait for the end of transmission...

Label:
       BTFSS      TX1STA, TRMT
       GOTO       Label

; … This program send 'D' caracter by UART1
; The transmission end after 86.806 us
; 115200 bit/s = 8,6806 us/bit
; Frame lenght : 10 bits
; 1 bit start + 8 bits payload + 1 stop
```

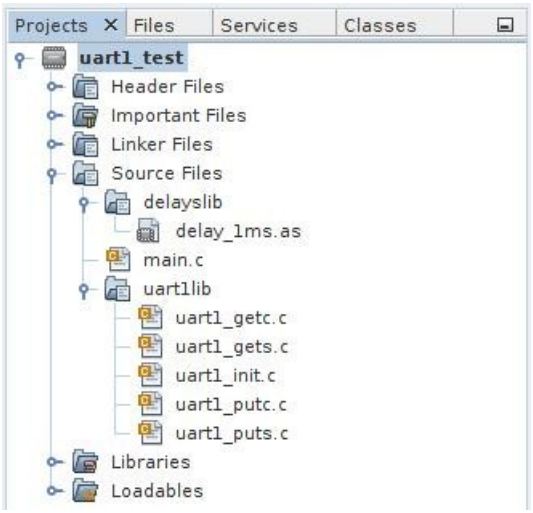
A notre époque, la plupart des ordinateurs modernes ne disposent plus de port COM avec connecteur DB-9 pour liaison série. Nous utilisons généralement des modules passerelles USB vers UART (TTL 0-5V) assurant une transposition de protocole mais ne modifiant pas la donnée échangée. Nous parlons alors de bridge ou de passerelle (exemple de votre box internet, par exemple ADSL vers WIFI). Un bridge UART1 vers UART2 sera d'ailleurs développé en TP. Sur le marché des modules ''USB to UART'', la société FTDI s'est spécialisée sur ce type de solution.



4.6. Module périphérique UART1 en transmission

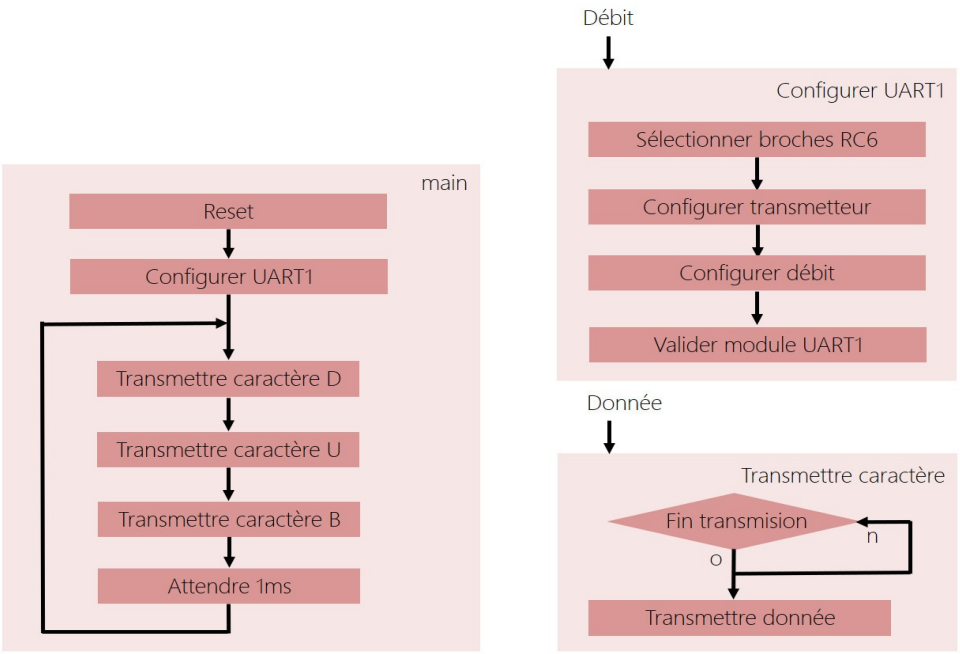
Nous allons maintenant découvrir les techniques d'une communication série asynchrone entre un processeur MCU PIC18F27K40 et un ordinateur équipé d'un terminal asynchrone dédié (TeraTerm, PuTTY, minicom, etc).

- Créer un projet MPLABX nommé *uart1_test* dans le répertoire *disco/bsp/uart1/test/pjct*. Inclure les fichiers *bsp/uart1/test/main.c*, *bsp/common/delay_1ms.c* et *uart1_init.c*, *uart1_putc.c*, *uart1_puts.c*, *uart1_getc.c*, *uart1_gets.c* présents dans le répertoire *bsp/uart1/src/*. S'assurer de la bonne compilation du projet. S'aider des tutoriels dans *mcu/tp/doc/tutos*



- Modifier les sources *uart1_init.c* et *uart1_putc.c* afin de configurer l'UART1 pour que le périphérique puisse implémenter une communication avec un débit de 9600Bd/s et respectant la configuration spécifiée dans le fichier d'en-tête *bsp/uart1/include/uart1.h. S'aider de la documentation (cartouche doxygen) de la fonction d'initialisation dans ce même header. Dans un premier temps, nous ne configurerons que le transmetteur et le générateur de débit, le tout sans gestion d'interruption.*

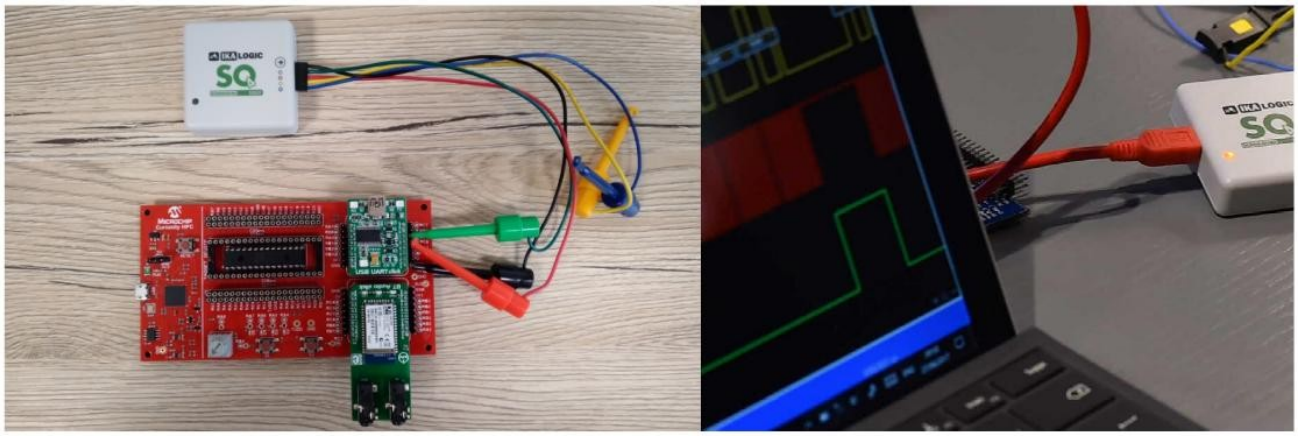
- Développer une application de test implémentant le diagramme de séquence suivant. *Cette technique d'envoi se nomme polling ou scrutation et ne nécessite aucune configuration d'interruption.*



Test sur carte physique Curiosity HPC

- Une fois validé, désélectionner le simulateur et sélectionner à nouveau la carte Curiosity pour les tests à venir sur carte physique
- Ouvrir les propriétés du projet : *fenêtre Projects > clic droit sur le nom du projet > Properties*
- Sélectionner la carte Curiosity HPC : *Hardware Tool > Microchip Kits > Curiosity > Apply > OK*
- Observer à l'oscilloscope le signal envoyé sur la broche RC6 (UART1 Tx) du MCU puis représenter ci-dessous les trames transmises par l'UART1 (caractères 'D' 'U' 'B' et attente de 1ms).

Analyseur Logique IKALOGC SQ50 ou SQ100

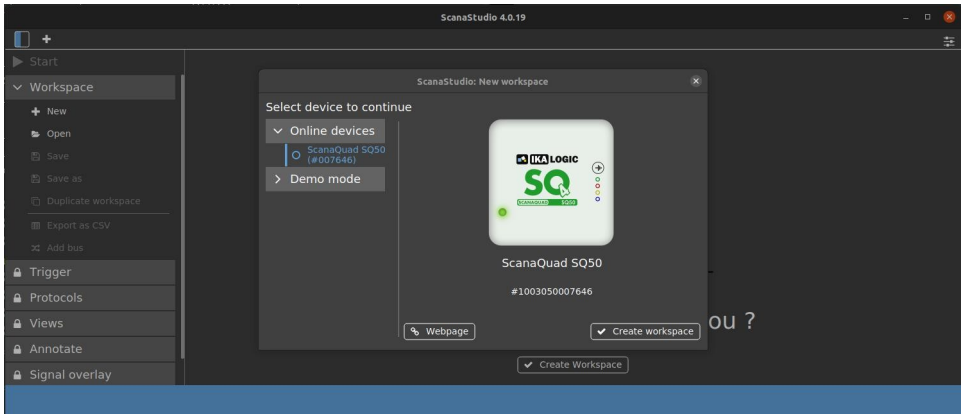


Contrairement à un oscilloscope, un analyseur logique est dédié à la capture et l'analyse de signaux logiques TOR (Tout ou Rien) le plus souvent captés lors de communications entre systèmes d'information (MCU, capteurs, actionneurs, ordinateur, etc). Nous allons ici utiliser un analyseur conçu en France par la société IKALOGIC en utilisant leur application pour ordinateur et tablette nommée SKANASTUDIO.

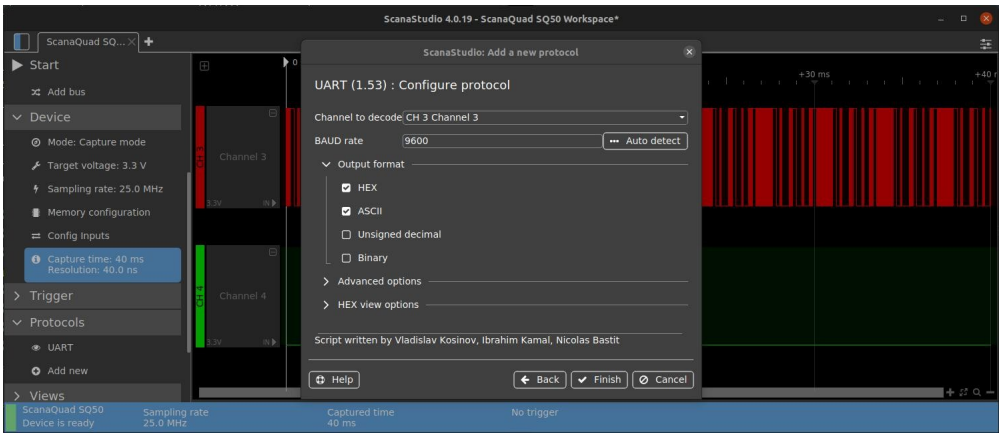
Pour rappel, ne jamais toucher un composant électronique avec les doigts pour des problème d'ESD (Electro-Static Discharge)ou de décharge électrostatique. Toujours manipuler les cartes en les tenant par les côtés/tranches. De même, lorsqu'avec une pince grippe fil vous essayer d'accrocher une broche (cf. sondes SQ50/100), toujours être prudent à ne pas réaliser de court-circuit entre la masse (GND) et l'alimentation (+5V ou +3,3V). Sinon, vous pouvez détruire le régulateur de tension +5V vers +3,3V.

- Connecter la sonde logique IKALOGIC SQ50 ou SQ100 à la carte Curiosity HPC hors tension comme ci-dessus :
  - Fil NOIR sur GND
  - Fil ROUGE sur RC6
  - Fil VERT sur RC7

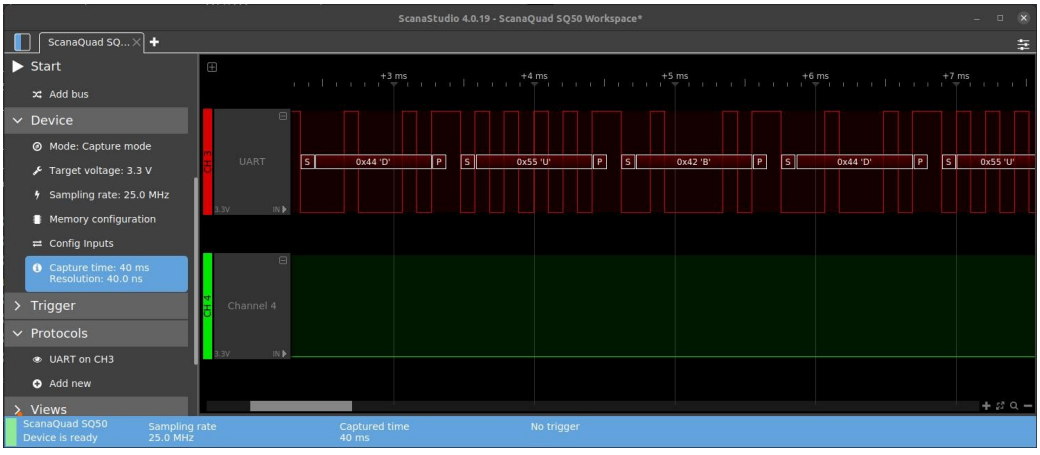
- Exécuter l'application SkanaStudio : *C:\Program Files (x86)\ScanaStudio\ScanaStudio.exe*
- Workspace > *New*
- Online devices > *ScanaQuad SQ50 ou SQ100 > Create workspace*



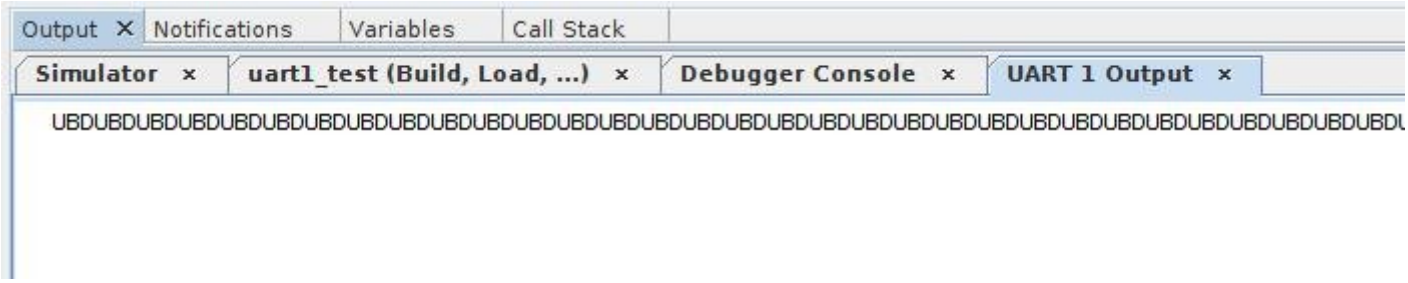
- Protocols > Add new
- Select protocol to continue > UART
  - Channel to decode : *CH 3 Channel 3*
  - BAUD rate : *9600*
  - Finish



- Observer la sortie, redimensionner l'échelle temporelle, renommer le canal, sauvegarder la configuration, etc



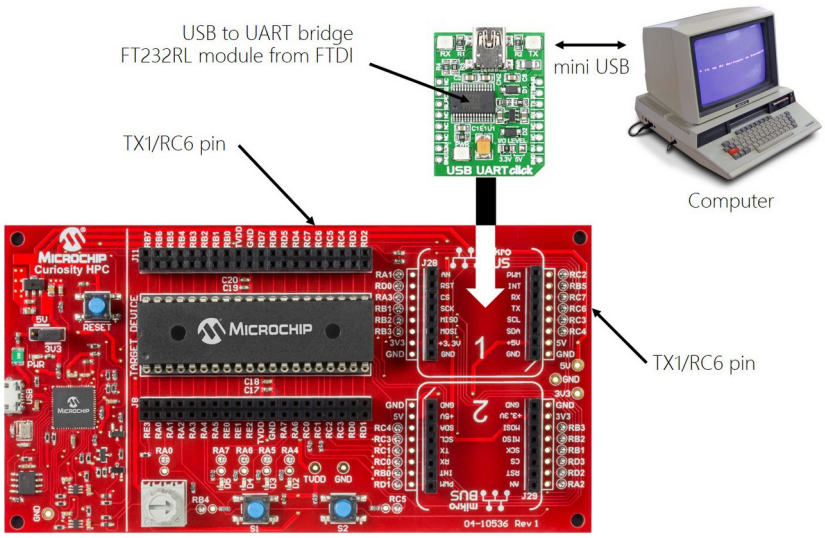
Test sur simulateur MPLABX IDE (sans carte de développement)



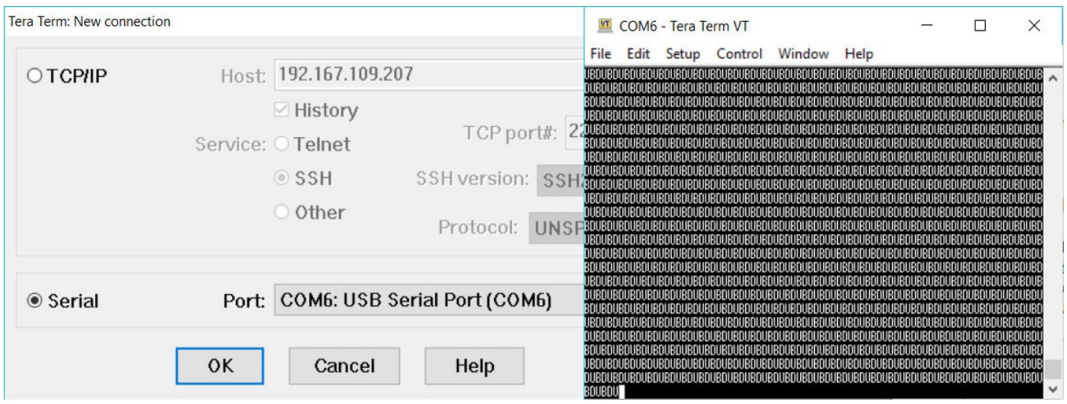
- Ouvrir les propriétés du projet : *fenêtre Projects → Clic droit sur le nom du projet → Properties*
- *Sélectionner le simulateur : Hardware Tool → Simulator → Apply*
- *Conf : [default] → Simulator*
- Activer l'UART1 sur le simultauer de MPLABX : *Option categories → UART1 IO Options → Enable UART1 IO → Apply → OK*
- Compiler et exécuter le programme : *fenêtre Projects → clic droit sur le nom du projet → Debug*
- Arrêter la simulation en cliquant sur le bouton carré rouge STOP (ou [Shift] + [F5])
- Observer la fenêtre de sortie UART 1 Output

4.7. Terminal de communication série sur ordinateur

Nous allons maintenant valider nos développements en déployant une communication série avec un ordinateur. Vérifier que le module *click board* de *Mikroelektronika USB to UART* soit bien physiquement connecté au socket *mikro BUS 1* dédié sur la carte Curiosity HPC. Vérifier également la connexion en USB avec l'ordinateur.



- Ouvrir un terminal de communication série sur ordinateur et valider le bon fonctionnement du programme. Par exemple, utiliser TeraTerm sous Windows :
  - Exécuter l'application : *C:\Program Files (x86)\teraterm\ttermpro.exe*
  - Sélectionner le terminal série : *Serial → sélectionner votre interface de communication → OK*
  - Configurer le port série : *Setup → Serial Port... → valider la configuration de la communication série → OK*
  - *Il ne reste plus qu'à observer les retours dans la console. De même, tout caractère saisi sur cette même console sera envoyé par le module série virtuel depuis l'ordinateur (port COM sous Windows) vers notre carte de développement*



4.8. Transmission de chaînes de caractères

- Modifier le fichier source *uart1_puts.c* puis de valider l'envoi de chaînes de caractères par UART1. Modifier la fonction *main* du programme en conséquent

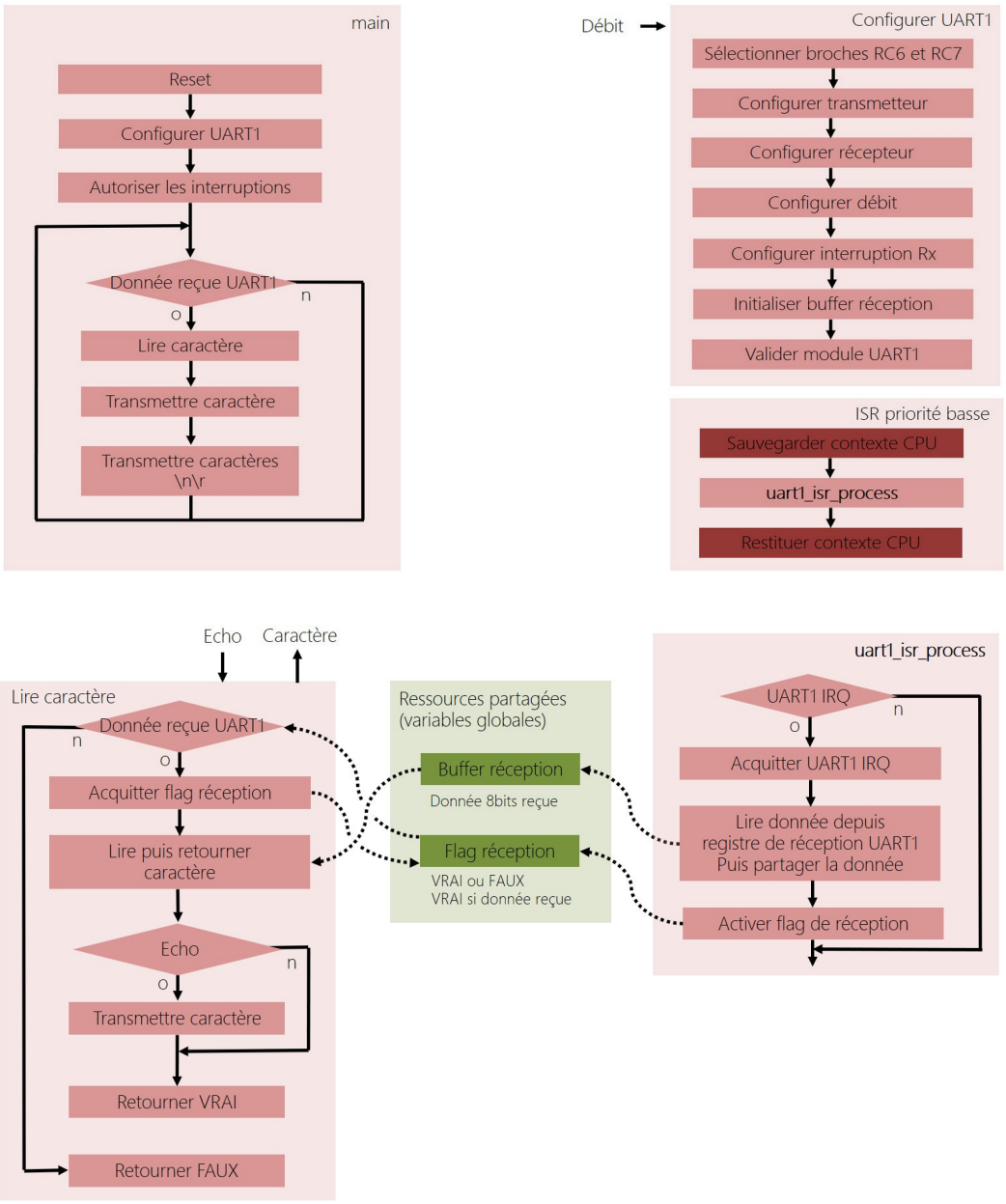
```
uart1_puts("DUB");
```

4.9. Module périphérique UART1 en réception

- Modifier les sources *uart1_init.c* et *uart1_getc.c* afin de configurer l'UART1 pour que le périphérique puisse implémenter la réception de données en respectant la configuration spécifiée dans le fichier d'en-tête *bsp/uart1/include/uart1.h. S'aider de la documentation (cartouche doxygen) de la fonction d'initialisation dans ce même header. Nous aurons à configurer l'interruption en réception en priorité basse ainsi que le récepteur de l'UART1. Pour information, en général sur MCU pour acquitter une interruption en réception pour un UART il nous suffit de lire et donc vider le registre de réception. C'est le cas sur MCU PIC18. Attention, les fonctions du fichier uart1_getc.c doivent probablement être les fonctions les plus complexes à écrire de la trame de TP. Être donc attentif à cette partie de la trame !*

- Quelle est la broche de réception Rx utilisée par défaut par l'UART1 sur carte Curiosity HPC ?

- Développer une application de test implémentant le diagramme de séquence suivant. *Lire les conseils et informations sur la page suivante*



L'utilisation de fonctions d'interruption ou ISR dans une application amène un nouveau paradigme, celui de la programmation événementielle. En effet, nous ne pouvons pas prédire le moment de réveil d'une ISR. Une routine d'interruption est donc entièrement indépendante de la fonction main. Pour la première fois à l'école, nous allons devoir impérativement utiliser des variables globales (ressources partagées) afin d'assurer l'échange d'information entre ISR et les fonctions appelées depuis le main. Nous aurons besoin d'échanger deux informations :

- Variable globale statique mémorisant la donnée reçue sur 8bits (cf. logigramme - Buffer réception)
- Variable globale statique mémorisant l'indicateur booléen (flag tout ou rien) spécifiant si une donnée est reçue (cf. logigramme - flag réception)

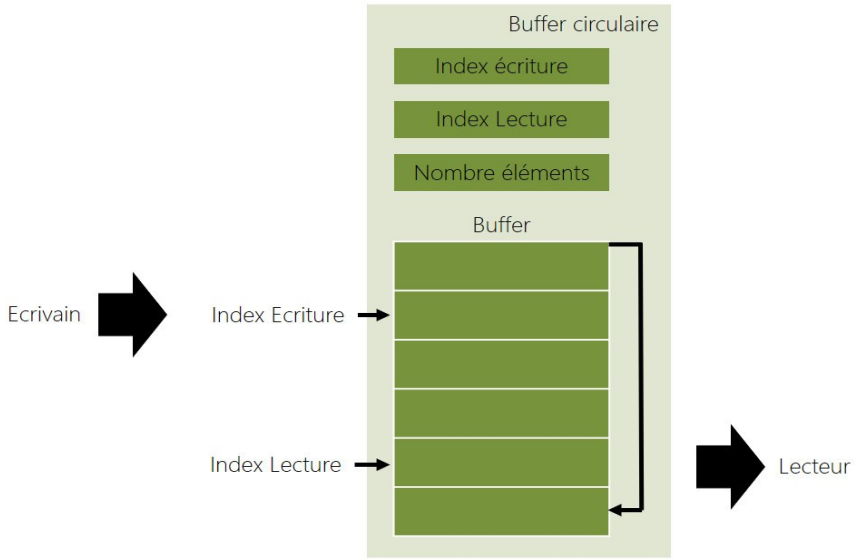
En langage C, toujours déclarer une variable globale dans un fichier source, le plus souvent dans le fichier source où est présent l'écrivain. De même, constater que le code de la fonction d'interruption n'est pas directement placé dans l'ISR. Ceci est lié à notre technologie processeur PIC18 et au fait qu'elle ne propose que deux fonctions d'interruptions de priorité haute ou basse pour tous les périphériques du MCU. Par exemple, dans notre ISR de priorité basse, nous n'appelons pour le moment qu'une fonction de traitement pour l'UART1 (uart1_isr_process). Mais à l'avenir, nous implémenterons plusieurs fonctions de traitement (UAR1, UART2, etc tout autre périphérique nécessaire à l'application). Cette technique nous permettra de garder visible nos ISR (priorités haute et basse) à côté de la fonction main au sein de nos applications. Notre programme gagne alors en simplicité, clarté et lisibilité.

- Valider le bon fonctionnement de votre programme en utilisant une console série sur ordinateur.
- Après validation, envoyer maintenant le contenu d'un fichier texte depuis la console. Voici la procédure sous TeraTerm :
  - *Se placer sur la console TeraTerm*
  - *File → Send file...*
  - *Sélectionner le fichier texte suivant disco/bsp/uart1/test/rx_test_file1.txt-> Ouvrir*
- Pourquoi n'observe-t-on qu'un caractère sur trois en retour sur la console ?
- Proposer des solutions à ce problème

4.10. Buffer circulaire de réception

Actuellement, le buffer d'échange entre l'ISR et la fonction de lecture de caractère ne fait qu'un octet. Nous allons le remplacer par un buffer de taille configurable (à la compilation) géré circulairement. Nous pouvons observer une définition de structure représentant un objet buffer circulaire dans le fichier d'en-tête *uart1.h*.

Dans notre cas, l'écrivain sera l'ISR et le lecteur la fonction *getc*. Lorsque l'écrivain écrit dans le buffer, il incrémente l'index d'écriture ainsi que le nombre d'éléments présents dans le buffer. Si le buffer est plein, l'écrivain stoppe les écritures. Les données reçues sont alors perdues. Lorsque que l'index d'écriture pointe la fin du buffer, il bascule au début de buffer en position initiale. Il est alors géré circulairement. Il en sera de même pour l'index de lecture. A chaque récupération de donnée correspondant à une lecture dans le buffer circulaire, le lecteur incrémente l'index de lecture et décrémente le nombre d'éléments présents dans le buffer. Si le buffer est vide, le lecteur passe son tour et attend une nouvelle écriture.



- Implémenter un buffer circulaire de 20 éléments entre l'ISR et fonction *getc*. Valider le fonctionnement du programme par envoi du fichier texte *rx-test-file1.txt* depuis le terminal de communication série.
  - *Se placer sur la console TeraTerm*
  - *File → Send file...*
  - *Sélectionner le fichier texte suivant disco/bsp/uart1/test/rx_test_file1.txt-> Ouvrir*
- Après validation, envoyer maintenant le contenu du fichier texte *rx-test-file3.txt* depuis la console. Pourquoi certains caractères sont-ils perdus durant la transmission ?

- Proposer des solutions à ce problème.

4.11. Contrôle de flux logiciel

A ce niveau des développements, deux solutions s'offrent à nous si nous ne souhaitons plus perdre de données à la réception. La première consisterait à étendre la taille du buffer circulaire. Néanmoins cette solution ne résout pas le problème mais repousse seulement les limites. Pour information, selon l'utilisation de l'UART dans une application, ces limitations peuvent être amplement suffisantes. Néanmoins, une solution optimale serait d'implémenter un contrôle de flux matériel ou logiciel.

Le concept de contrôle de flux est simple. Si le buffer de réception est plein, le récepteur demande explicitement à l'émetteur d'arrêter de transmettre. Dès que suffisamment de place a été libérée dans son buffer circulaire de réception, le récepteur prévient l'émetteur qu'il peut reprendre sa transmission. Ce contrôle des flux de communication peut être matériel ou logiciel :

- *Contrôle de flux matériel :* Utilisation de deux broches et conducteurs physiques complémentaires (broches CTS et RTS). Les deux signaux physiques sont croisés et permettent par simple logique booléenne aux récepteurs de chaque côté de la liaison série de stopper ou de valider les communications.

- *Contrôle de flux logiciel :* Sans utiliser de broches ni conducteurs physiques complémentaires, le contrôle de flux se fait par envoi de caractères spéciaux (XON ou 0x11 et XOFF ou 0x13). Si l'émetteur reçoit le caractère XOFF, il stoppe les transferts. Dès qu'il recevra le caractère XON, il reprendra les échanges.

- Implémenter et valider un contrôle de flux logiciel. La solution est simple, moins de 10 lignes de codes en tout. Ne pas se perdre donc dans l'implémentation. Les limites seront les suivantes :

  - *Envoi du caractère XOFF depuis l'ISR (écrivain) dès que le buffer circulaire de réception s'approche d'être plein à 4 éléments de sa capacité totale. La taille minimale du buffer circulaire sera fixée à 6 éléments.*

  - *Envoi du caractère XON depuis la fonction getc (lecteur) dès que la moitié du buffer a été vidée par l'application.*

- Configurer le contrôle de flux logiciel sur ordinateur :

  - *Se placer sous TeraTerm*

  - *Setup → Serial Port... → Flow control → Xon\|Xoff*

- Réitérer l'envoi et la réception du fichier texte *rx-test-file3.txt*

  - *Se placer sur la console TeraTerm*

  - *File → Send file...*

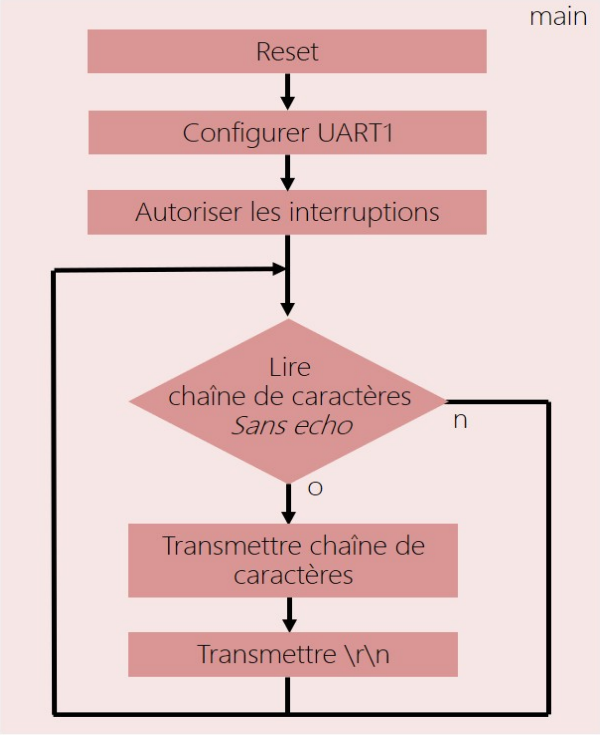
  - *Sélectionner le fichier texte suivant disco/bsp/uart1/test/rx_test_file3.txt-> Ouvrir*

  - *Normalement, la magie devrait opérer, félicitations !*

4.12. Réception de chaînes de caractères

- Modifier le fichier source *uart1_gets.c* afin d'assurer la réception de chaînes de caractères. Depuis la console, une action sur la touche Entrée (Enter ou '\r') signifiera la fin d'une saisie de chaîne de caractères. A la réception, la fonction devra remplacer le caractère '\r' par un '\0' afin de construire une chaîne de caractères. Si la chaîne de caractères capturée dépasse une taille fixée en entrée de fonction, l'écriture du tableau de destination se stoppera et se clôturera également par l'écriture d'un '\0'.

- Développer une application de test implémentant le diagramme de séquence suivant :



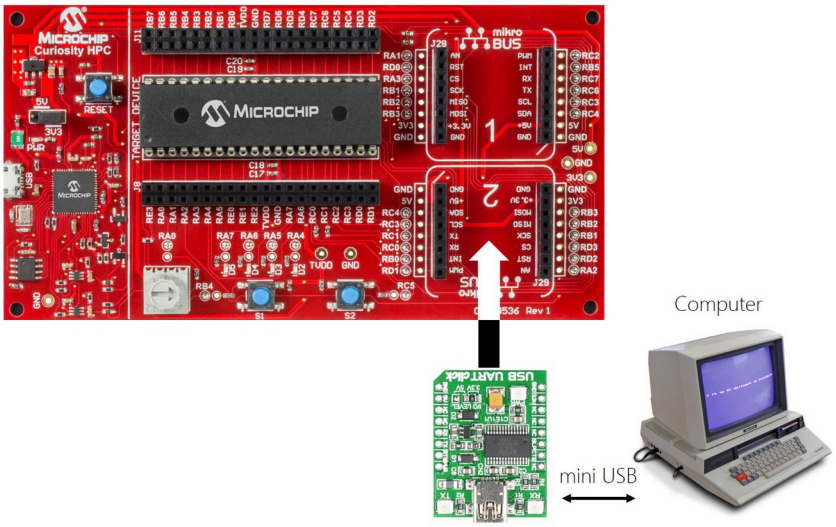
- Valider le bon fonctionnement de votre programme en utilisant une console série sur ordinateur.

- Envoyer maintenant le contenu du fichier texte *rx-test-file3.txt* depuis la console. *Si le contrôle de flux logiciel est implémenté ainsi qu'un buffer circulaire de taille suffisante, alors la magie devrait opérer ! Vous voilà apte à communiquer avec un ordinateur distant, bravo !*

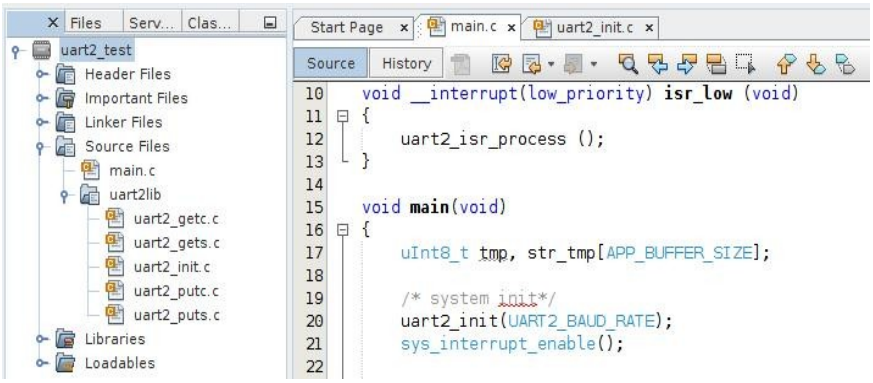
Nous venons de clôturer l'une des parties les plus techniques de la trame de TP. Synthétisons les développements, tests unitaires et validations réalisés durant les derniers exercices autour du périphérique de communication UART :

- Fonction de configuration du transmetteur et récepteur UART avec interruption à la réception
- Fonction d'envoi de caractères par polling (scrutation par boucle d'attente)
- Fonction d'envoi de chaînes de caractères par polling (scrutation par boucle d'attente)
- Fonction de réception de caractères par interruption
- Fonction de réception de chaînes de caractères par interruption
- Implémentation d'un buffer circulaire de réception
- Implémentation d'un contrôle de flux logiciel à la réception

4.13. Module périphérique UART2

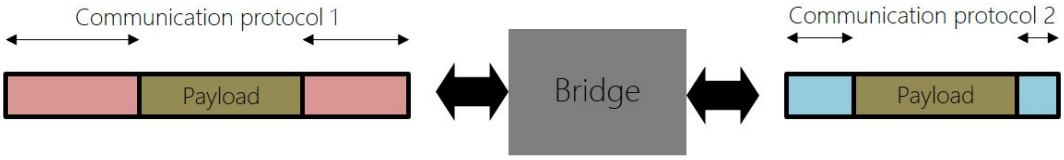


- Vérifier que le module *click board* de *Mikroelektronika USB to UART* soit physiquement connecté au socket *mikro BUS 2* dédié sur la carte Curiosity HPC
- Créer un projet MPLABX nommé *uart2_test* dans le répertoire *disco/bsp/uart2/test/pjct*



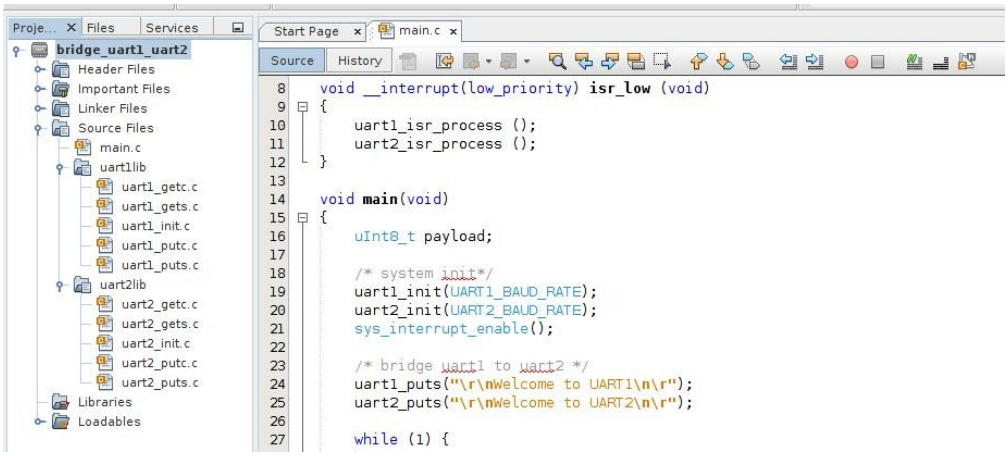
- Réitérer l'ensemble des configurations précédentes.Il s'agit quasiment que d'un exercice de recopie avec changement d'indice de 1 en 2. Réitérer les tests et valider la bibliothèque de fonctions pilotes pour l'UART2. Hormis pour la sélection des broches à adapter au module UART 2 (RX2 connecté à RC0 et TX2 connecté à RC1), il s'agit donc essentiellement d'un exercice de renommage de registres et de variables de l'UART1 vers l'UART2. Sous MPLABX (il y en a pour moins de 10mn) :
  - Copier les contenus (définitions) un à un des 5 fonctions pour l'UART1 vers les sources vides des 5 fonctions pour l'UART2 (~5mn), Puis réitérer les opérations qui suivent depuis le projet MPLABX pour l'UART2 …
  - Edit
  - Replace in Projects
  - Scope → Main Project *(choisir le projet principal affiché en gras sous MPLABX)*
  - Containing Text → *choisir le nom d'un registre UART1 à modifier*
  - Replace With → *nouveau nom du registre pour l'UART2*
  - Continue *(la modification est appliquée à tous le fichiers du projet principal)*
  - Réitérer l'opération pour tous les registres et variables à renommer (~5mn) !

4.14. Bridge de communication UART1 vers UART2



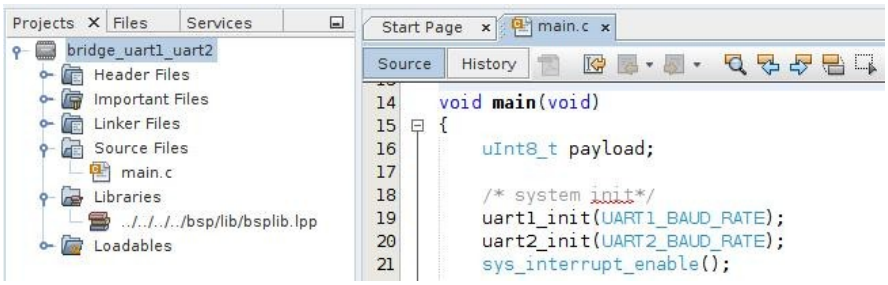
Un bridge ou passerelle de communication garantit la non-altération des données (payload) échangées, tout en offrant une transposition de protocole de communication à un autre. Prenons les exemples de votre box internet ADSL ou câble ou fibre (extérieur à l'habitat depuis l'opérateur internet) vers WIFI ou Ethernet (intérieur à l'habitat pour l'usager), des adaptateurs vidéo pour ordinateur VGA vers HDMI (ou DVI ou DisplayPort, etc), etc les applications sont multiples.

- Créer un projet MPLABX nommé *bridge_uart1_uart2* dans le répertoire applicatif *disco/apps/bridge_uart1_uart2/test/pjct* et ajouter au projet le fichier de test *disco/apps/bridge_uart1_uart2/test/main.c* . Ajouter également les sources de vos bibliothèques UART1 et UART2 précédemment développées puis compiler le projet.



- Analyser puis tester le programme fourni. Il s'agit d'une application simple de test assurant une passerelle de communication de périphérique UART1/2 à périphérique UART2/1.

- Créer une bibliothèque statique du BSP et réitérer le test (cf. *mcu/tp/doc/tutos*)



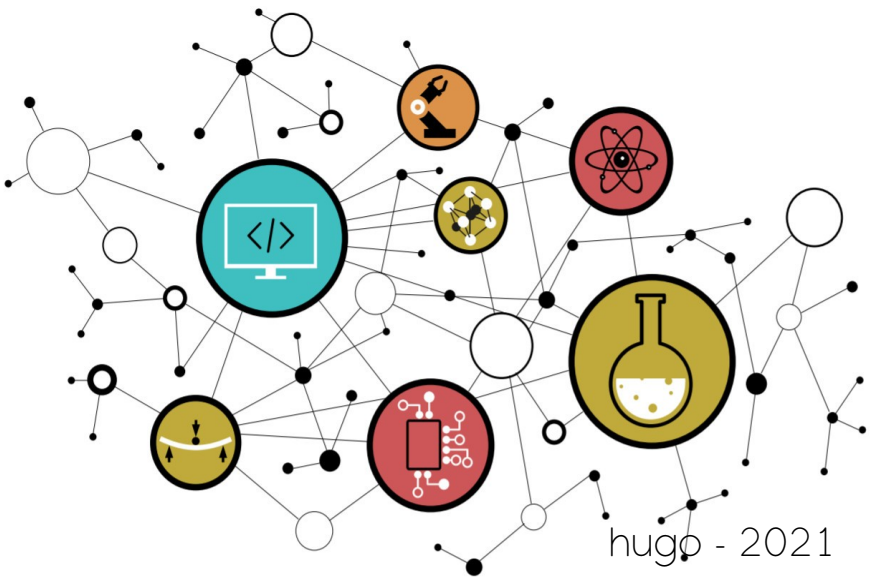
Cet exercice conclu bien des concepts en systèmes embarqués et dans le domaine des communications numériques. Nous venons d'atteindre ici même les objectifs strictement minimaux de la trame d'enseignement, félicitations ! Bien entendu, beaucoup reste encore à découvrir dans le domaine de l'embarqué. Comme tout domaine des sciences, l'apprentissage de l'embarqué est sans frontière ni limite …

Si certains concepts ou points techniques appréhendés jusqu'à présent ne sont pas pleinement assimilés, nous vous invitons à repasser sur la compétence. Car tout ce qui sera vu jusqu'en 3ième année en Systèmes Embarqués dépend des bases assises jusqu'à présent. Il en est de même en langage C.


notes

TRAVAUX PRATIQUES

MODULE AUDIO BLUETOOTH EXTERNE



hugo - 2021

SOMMAIRE

MODULE AUDIO BLUETOOTH EXTERNE

5. MODULE AUDIO BLUETOOTH EXTERNE

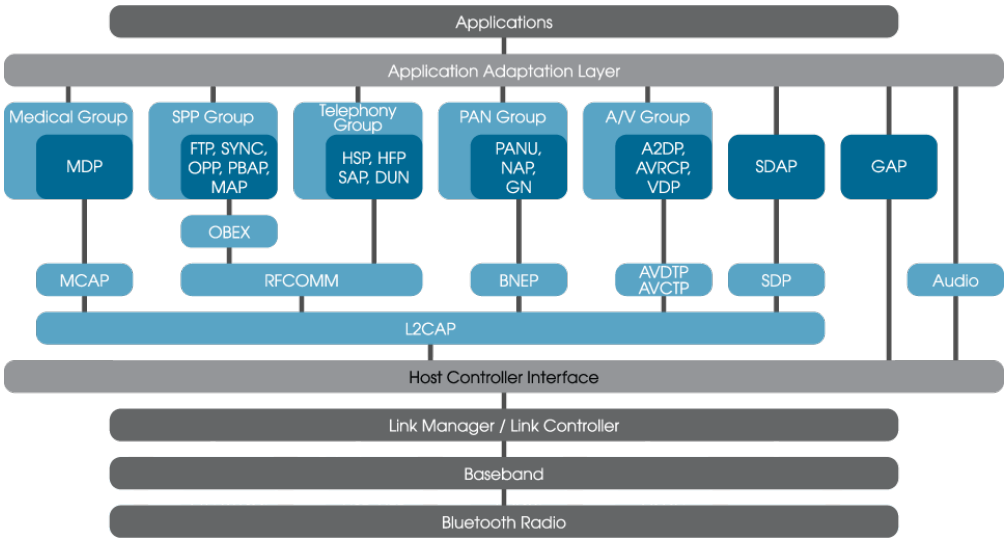
5.1. Introduction

Dans l'embarqué, la communication de machine à machine (M2M ou Machine to Machine) est un point souvent central dans la conception d'une application. Il s'agit notamment du cœur des systèmes dits nomades. Les solutions technologiques de communication les plus couramment rencontrées en embarqué à notre époque sont le WIFI, le Bluetooth, les technologies cellulaires (LTE, 4G, 3G et 2G), le NFC (Near Field Communication), LORA, Sigfox, etc. Aucune technologie n'est meilleure qu'une autre. Elle possèdent toutes des avantages et inconvénients. Elles répondent toutes à une famille de besoins parfois très différents.

Par exemple, diffuser un flux vidéo streaming en WIFI est techniquement et physiquement très différent d'une simple lecture de température puis échange toutes les minutes sur un appareillage de production dans une usine. Il existe donc des protocoles et solutions techniques adaptées à chaque besoin. Le tableau ci-dessous synthétise "approximativement" les périmètres d'actions des principales solutions technologiques actuelles du marché.

Protocol	Optimized for Battery Life	Nominal Range Limit	Tyical Data Rate	Spectrum
Bluetooth		<10m	2Mbps	ISM 2.4GHz unlicensed
WIFI		<100m	>100Mbps	ISM 2.4GHz/5GHz unlicensed
LORAWAN		>10Km	<50Kbps	ISM 900MHz unlicensed
2G/3G		>30Km	<2Mbps	*Licensed cellular*
4G		>30Km	>100Mbps	*Licensed cellular*
NFC		<4cm	100Kbps	ISM 13.56MHz unlicensed

Le Bluetooth est par exemple une norme de communication permettant l'échange bidirectionnel de données sur de courte distance (<100m voire <qqm suivant la classe de fonctionnement) en utilisant des ondes radio dans la bande UHF (bande de fréquence autour de 2,4GHz). Comme la plupart des normes et protocoles de communication, le Bluetooth peut être représenté en couches protocolaires plus ou moins proches du monde physique (couche Radio). Nous parlons souvent de Stack (ou Pile) protocolaire en faisant référence à certaines bibliothèques logiciel. Dans l'exercice de la trame de TP, le module RN52 utilisé intègre et implémente le profil Audio A2DP (groupe A/V ou Audio/Vidéo), utilisant lui-même la couche liaison de multiplexage L2CAP, elle-même interfacée par les couches en bandes de bases et Radio.



Il existe sur le marché des modules de communication plus ou moins intégrés (emport potentiel de couches protocolaires ou stack Bluetooth) avec des échelles de coûts souvent liées à la richesse des fonctionnalités embarquées dans le module de communication (exemple du schéma fonctionnel ci-dessous). Dans le cadre de notre application, nous utiliserons une solution intégrée gérant déjà pleinement une partie du protocole Bluetooth désiré (profil Audio A2DP, couche liaison L2CAP, couche en bande de base et couche radio) et s'interfaçant par simple liaison série asynchrone et périphérique UART. Par exemple, le module Bluetooth RN52 de Microchip supporte déjà les couches protocolaires nécessaires à l'application (cf. schéma de droite ci-dessous). Nous n'aurons qu'à le configurer et le contrôler depuis le MCU par envoi de chaînes de caractères ASCII avec une communication par liaison série asynchrone via UART (cf. tableaux ci-dessous).

Nous pouvons par exemple observer ci-dessous un extrait de la documentation technique du module Bluetooth RN52 (cf. *mcu/tp/doc/datasheets*) présentant la synthèse de toutes les commandes de configuration et d'action supportées. Par exemple, si notre MCU envoi par UART les suites de caractères ASCII suivantes (cf. figure de gauche ci-dessus), le module RN52 réalisera les traitements demandés :

- AV+ : incrémente le volume (Audio Volume +)
- AT- : rejoue la dernière piste Audio (Audio Track -)
- SN, <string> : change le nom du réseau Bluetooth créé par le module RN52 par <string>
- etc

SET COMMANDS

Command	Description
S|,<*hex16*>	Audio output routing
S-,<*string*>	Sets the normalized name
S^,<*dec*>	Automatic Shutdown on Idle
S%,<*hex16*>	Extended Features
SA,<0,1,2,4>	Authentication enable/disable
SC,<*hex24*>	Service class
SD,<*hex8*>	Discovery profile mask
SF,1	Factory defaults
SK,<*hex8*>	Connection profile mask
SM,<*hex32*>	Microphone/LINEIN gain
SN,<*string*>	Device name
SS,<*hex8*>	Speaker Level
ST,<*hex8*>	Tone Level
STA,<*dec*>	Connection Delay
STP,<*dec*>	Pairing Timeout
SU,<*hex8*>	UART Baudrate

ACTION COMMANDS

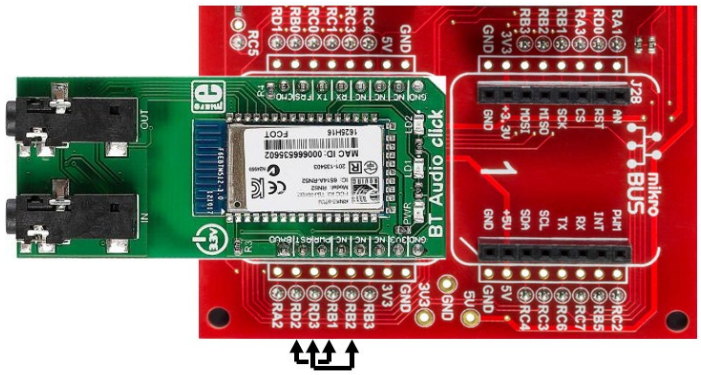
Command	Description
+	Toggle the local echo of RX characters in Command mode
@,<*flag*>	Toggle whether the module is discoverable
#,<0,1>	Accept/reject pairing
$	Put the module into DFU mode
A,<*telephone number*>	Initial a voice call to <*telephone number*>
AD	Retrieve track metadata information
AR	Redial last dialed number
AV+	Increase the volume (AVRCP command)
AV-	Decrease the volume (AVRCP command)
AT+	Play the next track (AVRCP command)
AT-	Play the previous track (AVRCP command)
AP	Pause or start playback (AVRCP command)
B	Reconnect Bluetooth® profiles to the most recently paired and connected device
C	Accept an incoming voice call
E	Terminate an active call or reject an incoming call
F	Release all held calls
I@	Read GPIO configuration
I@,<hex16>	Set GPIO configuration
I&	Reads current GPIO levels for input
I&,<hex16>	Set GPIO levels for output
J	Accept waiting calls and release active calls
K,<*hex8*>	Kill the currently active connection
L	Accept waiting calls and hold active calls
M,<*flag*>	Toggle the on hold/mute function
N	Add held call
O	Connect two calls and disconnect the subscriber
P	Activate Voice Command
Q	Query the current connection status
R,1	Reboot
U	Reset Paired Device List (PDL)
T	Retrieves caller ID information
X,<0,1>	Transfer call between HF and AG

5.2. Configuration du module Audio Bluetooth RN52

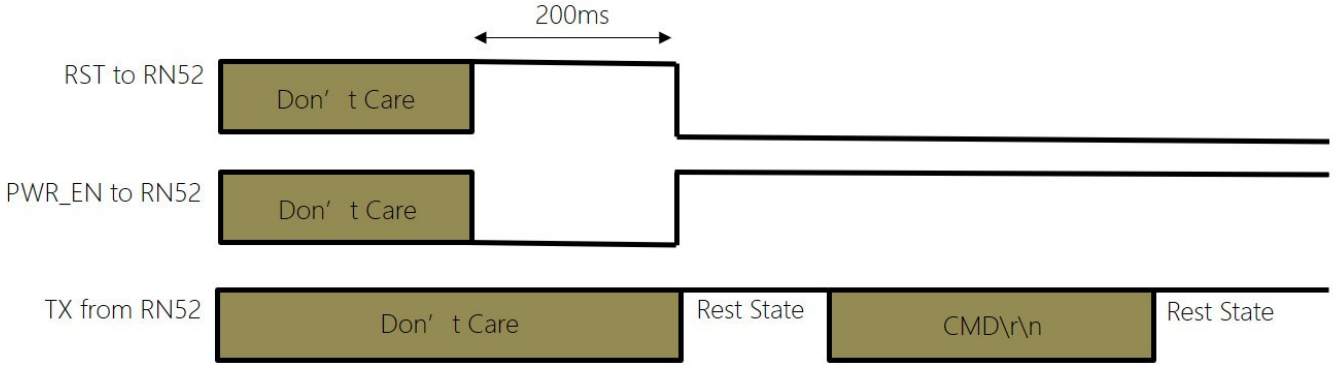
- Créer un projet MPLABX nommé *rn52_test* dans le répertoire *disco/bsp/rn52/test/pjct*. Inclure les fichiers *bsp/rn52/test/main.c*, *bsp/common/delay_200ms.c* et *rn52_init.c*, *rn52_cmd.c* présents dans le répertoire *bsp/rn52/src/*. S'assurer de la bonne compilation du projet. S'aider de l'annexe 1.

- Modifier les sources *rn52_init.c* et *rn52_cmd.c* afin de configurer le module RN52 (cf. page suivante). La configuration déploiera un réseau bluetooth (nom du réseau à fixer) avec un profil audio A2DP. Par défaut, aucun code d'association ne sera demandé. Le module RN52 sera interfacé par l'UART2 avec un débit de 9600Bd/s et respectant la configuration spécifiée dans le fichier d'en-tête *bsp/rn52/include/rn52.h*. Le module utilisera en tout 4 broches du MCU, respectivement RB1/RB2/RC0-RX2/RC1-TX2 (côté MCU) et RST/PWR_EN/TX/RX (côté RN52). Afin d'assurer le chemin de l'information, il nous faudra réaliser deux ponts filaires entre les broches :

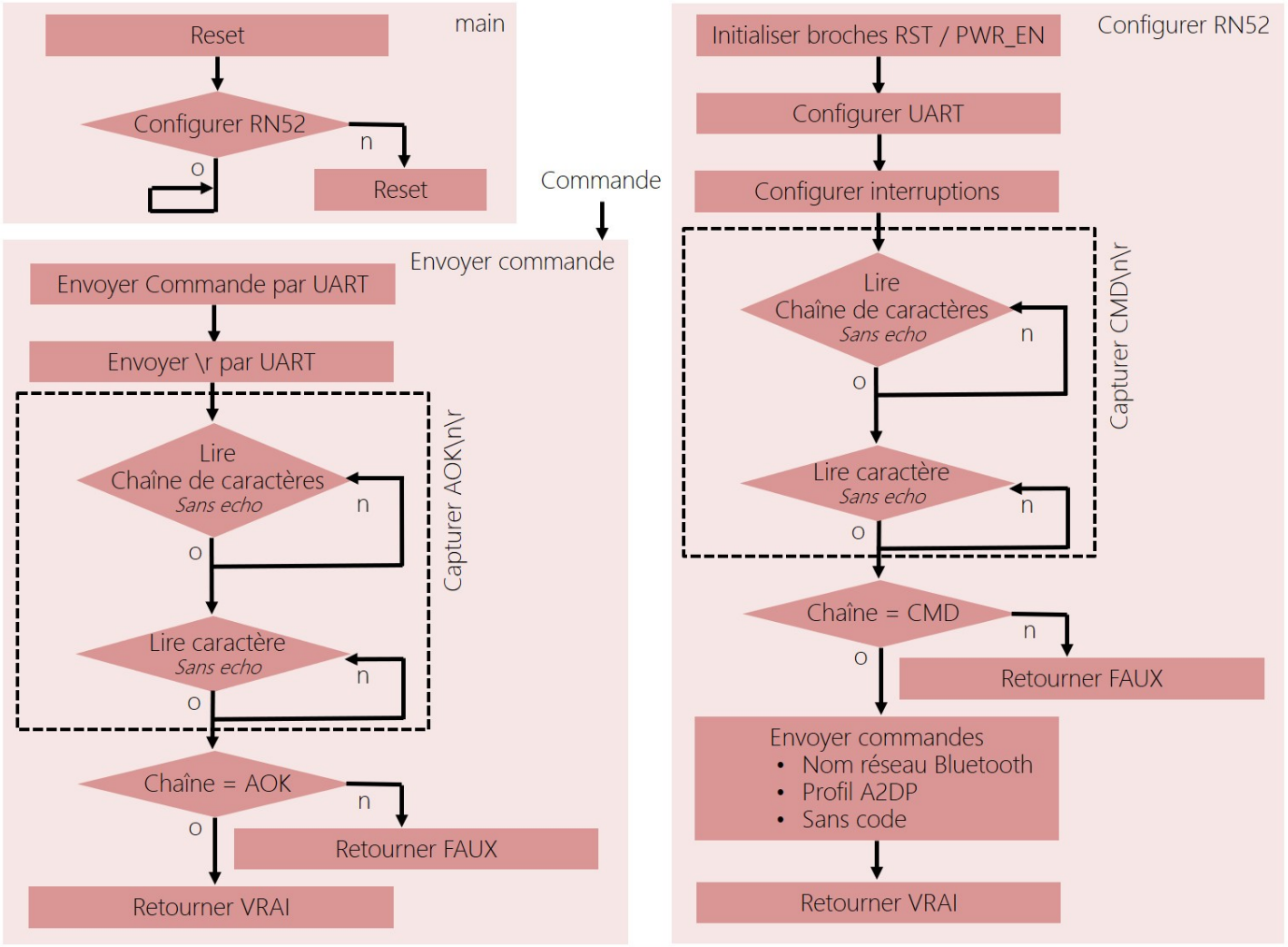
  - *RB1 <-> RD2 via le connecteur J27de la carte Curiosity HPC (cf. ci-dessous)*
  - *RB2 <-> RD3 via le connecteur J27de la carte Curiosity HPC (cf. ci-dessous)*



- S'aider du guide utilisateur présentant le jeu de commande du module RN52 (cf. *disco/bsp/doc/datasheets*). Nous utiliserons le module Bluetooth RN52 en mode commande (broche GPIO9 à l'état bas). Au démarrage, nous devons attendre un temps supérieur à 100ms (200ms dans notre cas) afin de laisser le module démarrer puis passer en mode CMD (commande). Ce mode nous permettra de piloter le module par simple envoie de caractères ASCII (cf. tableaux précédents SET COMMANDS et ACTIONS COMMANDS). Nous aurons donc à respecter la séquence de mise sous tension ci-dessous (broches RST/GPIO3 et PWR_EN). Le module retourne par UART la chaîne de caractères ''CMD\r\n''. De même, après envoie d'une commande valide, le module retourne la chaîne de caractères ''AOK\r\n'' pour valider la bonne réception de cette même commande. Les broches RST/GPIO3 (RB1 côté MCU) et PWR_EN (RB2 côté MCU) du module RN52 doivent respecter la séquence suivante au démarrage (phase de reset puis phase de validation à la mise sous tension) :



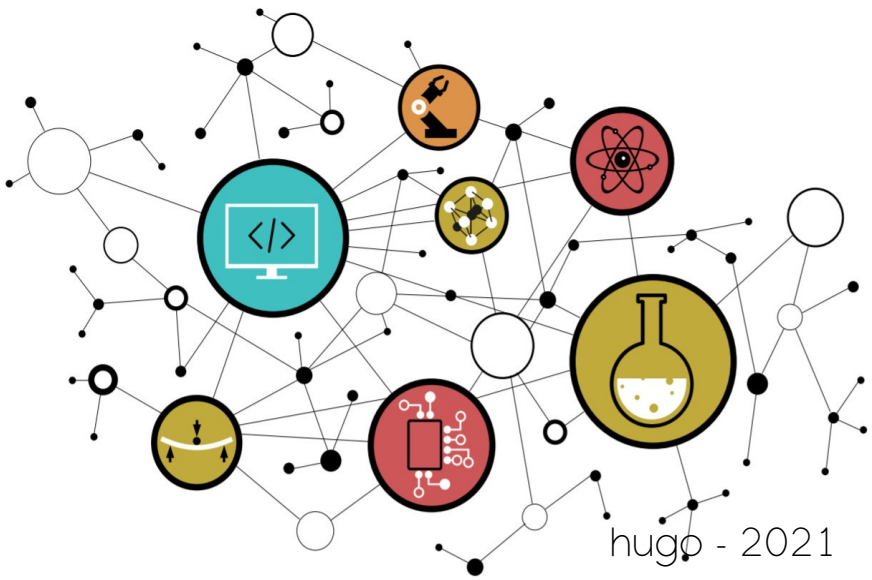
- Développer une application de test implémentant le diagramme de séquence suivant en utilisant l'UART2 pour piloter le module RN52. *Ne pas hésiter à interfacer l'UART1 depuis un ordinateur comme console de debug de sortie (non présenté ci-dessous). A l'image du debug par printf souvent fait sur ordinateur :*




notes

TRAVAUX PRATIQUES

MODULE DE CONVERSION ADC



hugo - 2021

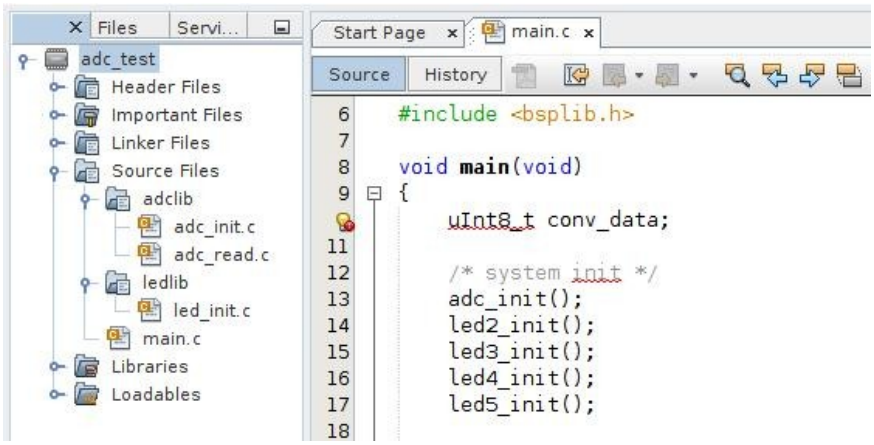
SOMMAIRE

MODULE DE CONVERSION ADC

7. MODULE DE CONVERSION ADC

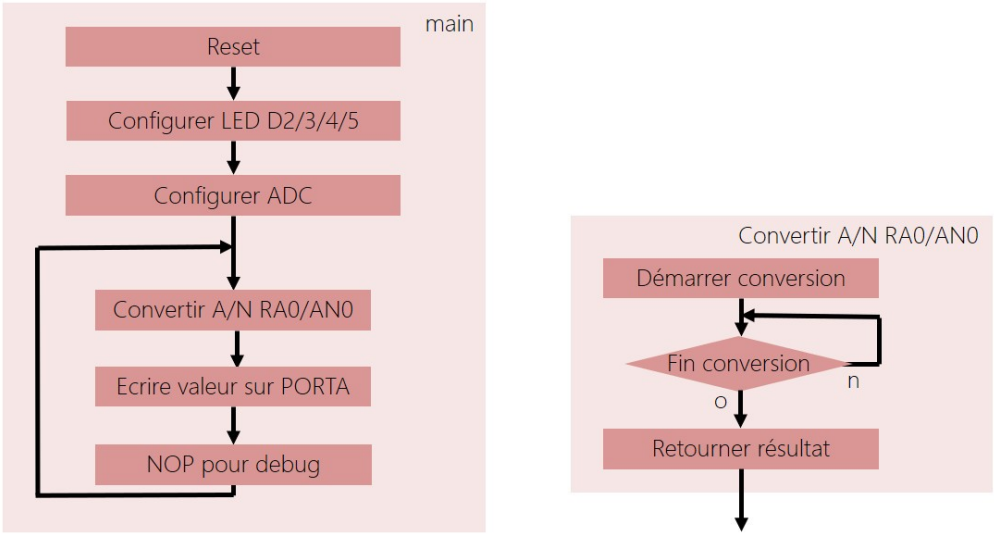
Cet exercice vise à lier et illustrer deux grands pans du large domaine de l'électronique, celui de la passerelle entre domaine de l'analogique et du numérique. Nous allons découvrir une problématique de conversion sur technologie de convertisseur à approximation successive, solution la plus couramment rencontrée sur MCU pour sa faible empreinte sur silicium.

- Créer un projet MPLABX nommé *adc_test* dans le répertoire *disco/bsp/adc/test/pjct*. Inclure les fichiers *bsp/adc/test/main.c, bsp/adc/src/adc_initc, bsp/adc/src/adc_read.c* et s'assurer de la bonne compilation du projet. S'aider de l'annexe 1.



- Modifier les sources du projet afin de configurer l'ADC pour qu'il puisse réaliser une conversion analogique/numérique sur la broche RA0/AN0 (broche connectée au potentiomètre sur la carte Curiosity HPC). S'aider du fichier d'en-tête *bsp/adc/include/adc.h* et de la documentation (cartouche doxygen) de la fonction d'initialisation dans ce même header.

- Développer une application de test implémentant le logigramme suivant :



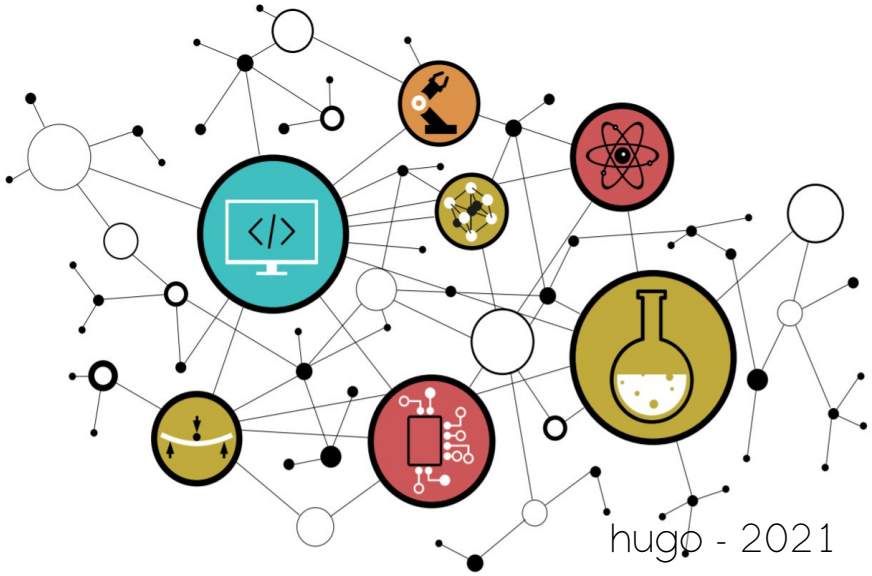
- Valider le fonctionnement de l'application par observation des états logiques appliqués sur les LED 2/3/4/5 connectées au port A.

- En mode debug, mettre un point d'arrêt sur l'instruction *nop*. Observer les valeurs numériques converties présentes dans les registres de sortie de l'ADC 10bits (ADRESH et ADRESL) lorsque le potentiomètre est en butée (gauche et droite) : PIC/Target Memory Views → SFRs. Pourquoi la plage numérique convertie n'est-elle pas comprise entre 0x000 et 0x3FF (0V-3,3V) ?


notes

TRAVAUX PRATIQUES

CONCEPTION D'UNE APPLICATION ET ORDONNANCEMENT



hugo - 2021

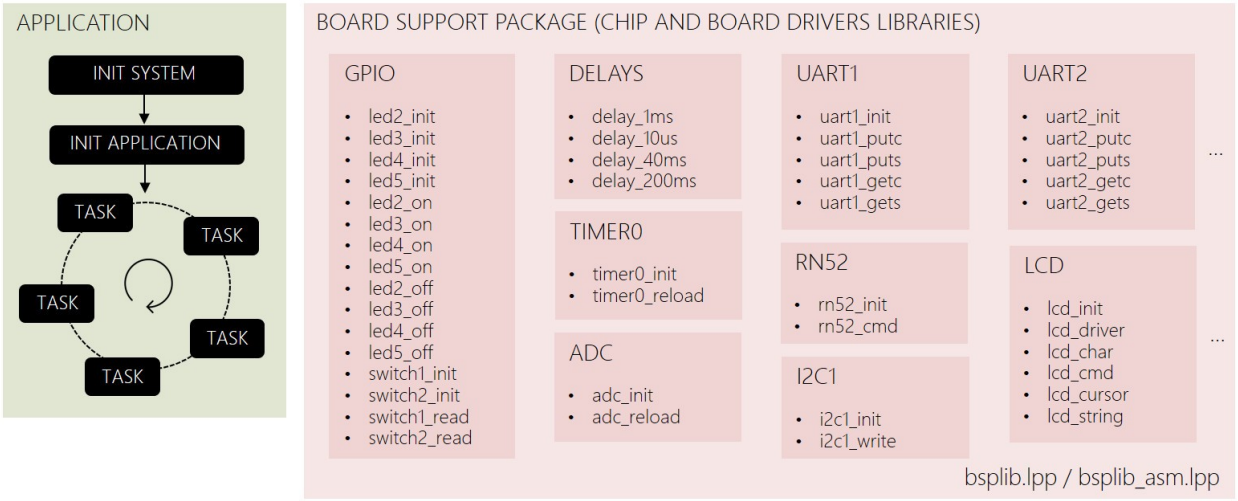
SOMMAIRE

CONCEPTION D'APPLICATION ET ORDONNANCEMENT

8. CONCEPTION D'UNE APPLICATION ET ORDONNANCEMENT

Toute application logicielle en Systèmes Embarqués débutera par la configuration séquentielle des services matériels nécessaires (périphériques internes et externes). En fonction de l'état des entrées du système (switch, bouton poussoir, capteurs divers, interfaces réseaux, etc) suivra ensuite l'initialisation de l'environnement logicielle de l'application (variables d'environnement) et la mise à jour des interfaces utilisateur (afficheur, LED, déploiement de réseau de communication, etc) avec l'état du produit par défaut au démarrage. L'application pourra alors débuter.

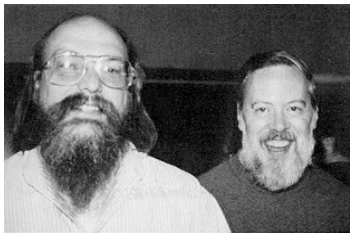
Une application *software* est une solution logicielle de supervision d'un produit répondant à un besoin (souris, clavier, lecteur MP3, assistance au freinage, etc) et a été développée pour une mission spécifique. Cette mission globale sera toujours la même pour un produit donné et peut intégrer des sous missions (lire les entrées du système, gérer l'affichage utilisateur, gérer les interfaces de communication, etc). Ces sous missions sont nommées tâches. Une application aura donc toujours plusieurs tâches à réaliser. A titre indicatif, le *scheduler* ou ordonnanceur d'un système d'exploitation est chargé de gérer et ordonner un environnement multi-tâches voire multi-applications et de répartir les besoins (tâches ou *threads* souhaitant s'exécuter) sur les ressources d'exécution (un ou plusieurs CPU). Cependant, nous ne découvrirons ce type d'ordonnancement (*scheduling online*) qu'en 2ième année à l'école. En première année, nous développerons une application dite *Baremetal*, soit nue sur le MCU.



L'un des objectifs premier de cette trame d'enseignement est le développement d'un BSP (Board Support Package) pour la carte Curiosity HPC sur PIC18F27K40 (développements, tests, validations voire documentation). Des projets de tests unitaires ont été réalisés par fonction périphérique et certaines intégrations partielles ont également été validées (UART1 avec UART2, ou Timer0 avec GPIO par exemples). A travers le développement d'une application, nous avons à valider l'intégration de l'ensemble des modules (GPIO, Timer0, UART1, UART2, I2C, etc) nécessaires à développement du produit (application Audio Bluetooth dans notre cas). La conception d'une application nécessite une grande attention afin de définir une architecture ainsi qu'une stratégie d'ordonnancement répondant de façon optimale au besoin et garantissant modularité, clarté, simplicité, évolutivité et robustesse au projet logiciel.

Nous allons nous attacher à développer une application conçue autour d'un *scheduler offline*. Le séquencement et les mesures temporelles des différentes tâches applicatives seront réalisées et validées avant la mise en production durant les phases de développement et de test. Nous maîtriserons donc le déterminisme à l'exécution de notre application. Ce type d'ordonnancement est souvent rencontré dans les systèmes critiques, solutions où le droit à l'erreur n'est pas permis malgré une complexité systémique pouvant être importante (par exemple dans le domaine de l'avionique avec *Airbus*). Les *scheduler offline* peuvent cependant être déployés sur de plus petits systèmes, par exemple dans des applications industrielles (Automate Programmable Industriel, exemple du compteur Linky, etc). Cependant, ce type d'ordonnancement est plus difficilement évolutif et maintenable que d'autres solutions dîtes *online* (ordonnancement et partage du temps CPU à l'exécution).

8.1. Philosophie Unix



L'un des plus grands défis de l'ingénieur est de converger vers la solution la plus simple durant la conception et le développement d'une solution (électronique, informatique, mécanique, etc). Nous parlons souvent de principe de parcimonie ou du rasoir d'Ockham. Nous pouvons voir ci-dessus Ken Thompson (à gauche, concepteur d'Unix et inventeur du langage B) et Dennis Ritchie (à droite, inventeur du langage C et codéveloppeur d'Unix), deux des pères des langages informatique et des systèmes d'exploitation ayant formalisé des bases philosophiques dans le développement des systèmes d'information. Ces règles peuvent être appliquées à bien des domaines. Vous trouverez par exemple ci-dessous 13 des 17 règles Unix proposées par Eric S. Raymond, un hacker Américain notamment connu pour avoir popularisé le terme ''Open Source'' par opposition à ''free software'', dont Richard Stallman est l'initiateur (le père du projet GNU et de la Free Software Foundation).

«La simplicité est la sophistication suprême » Léonard de Vinci

- Règle de Modularité : *Écrire des éléments simples reliés par de bonnes interfaces*
- Règle de Clarté : *La Clarté vaut mieux que l'ingéniosité*
- Règle de Séparation : *Séparer les règles du fonctionnement; Séparer les interfaces du mécanisme*
- Règle de Simplicité : *Concevoir pour la simplicité; ajouter de la complexité seulement par obligation*
- Règle de Parcimonie : *Écrire un gros programme seulement lorsqu'il est clairement démontrable que c'est l'unique solution*
- Règle de Transparence : *Concevoir pour la visibilité de façon à faciliter la revue et le déverminage*
- Règle de Robustesse : *La robustesse est l'enfant de la transparence et de la simplicité*
- Règle de Représentation: *Inclure le savoir dans les données, de manière que l'algorithme puisse être bête et robuste*
- Règle du Silence : *Quand un programme n'a rien d'étonnant à dire, il doit se taire*
- Règle de Dépannage : *Si le programme échoue, il faut le faire bruyamment et le plus tôt possible*
- Règle d'Optimisation : *Prototyper avant de fignoler. Mettre au point avant d'optimiser*
- Règle de Diversité : *Se méfier des affirmations de « Unique bonne solution »*
- Règle d'Extensibilité : *Concevoir pour le futur, car il arrivera plus vite que prévu*
- *etc*

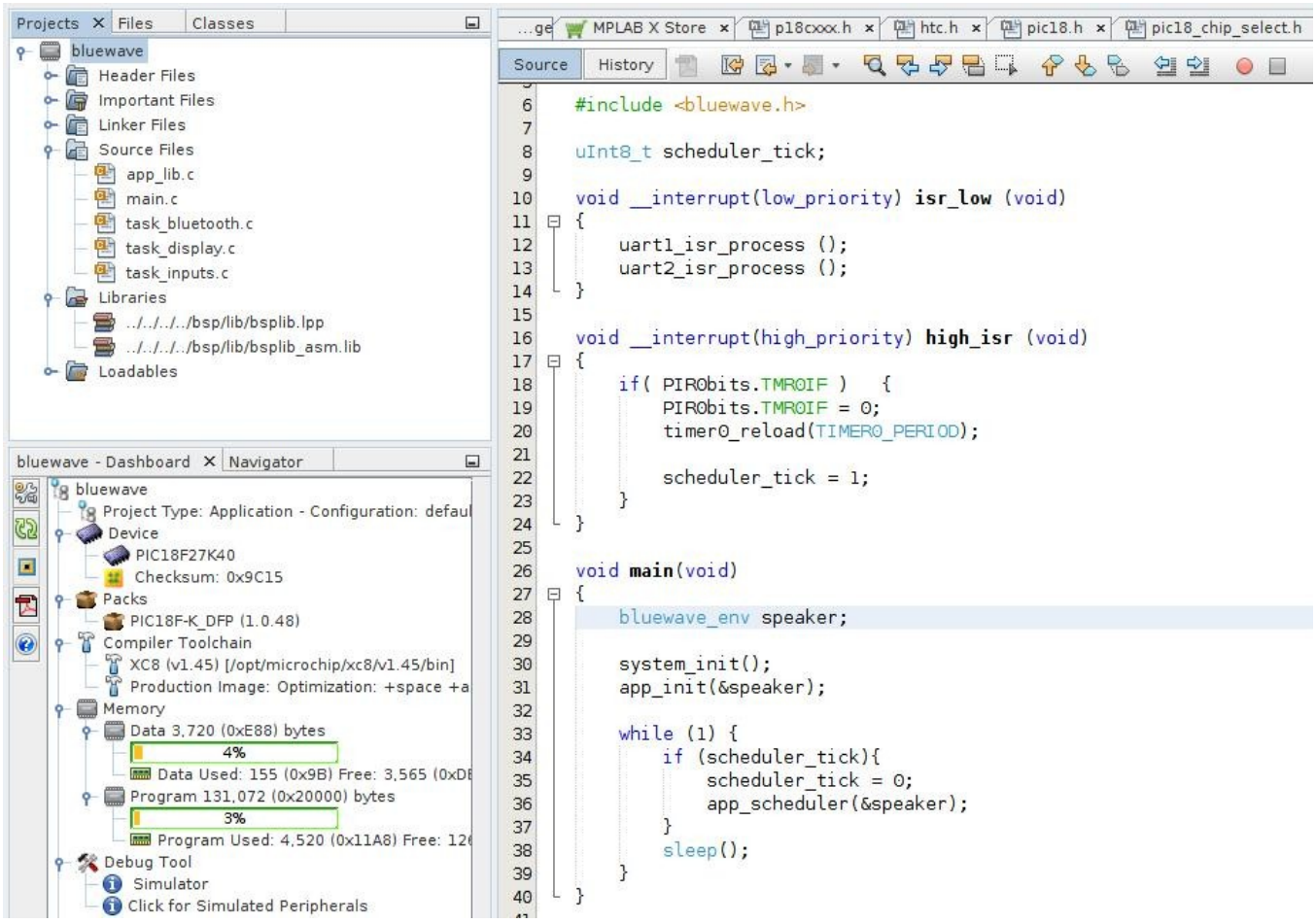
8.2. Conception et ordonnancement de l'application

- Si ce n'est pas déjà réalisé, créer deux projets MPLABX pour la générations des bibliothèques statiques nommés *bsplib_<your_name>.lpp* et *bsplib_asm_<your_name>.lpp* (retirer la librairie assembleur et inclure les sources ASM au projet si problèmes rencontrés à l'édition des liens) dans le répertoire *disco/bsp/lib*. S'aider des documents présents dans *mcu/tp/doc/tutos*

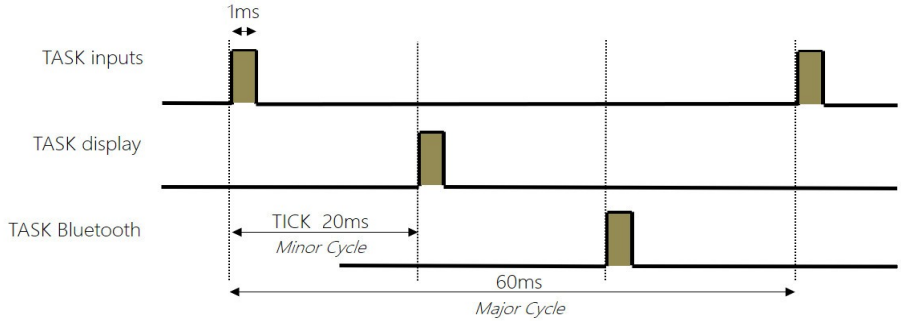
- Créer un projet MPLABX nommé *bluewave* dans le répertoire *disco/apps/bluewave/pjct*. Inclure tous les fichiers sources présents dans le répertoire *bsp/apps/bluewave/src/*. S'assurer de la bonne compilation du projet. S'aider des documents présents dans *mcu/tp/doc/tutos*

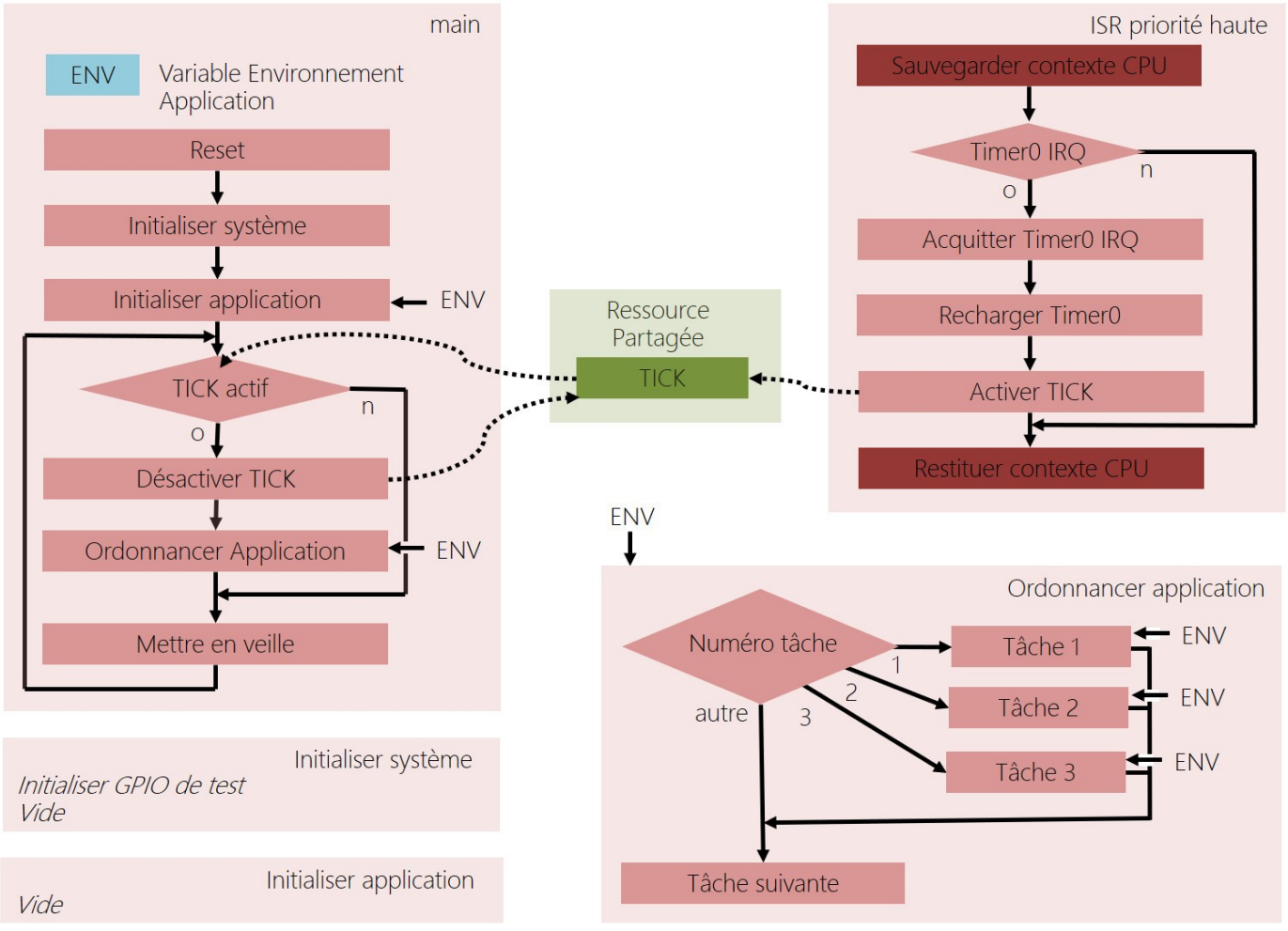
- Ajouter au projet les bibliothèques statiques *bsplib_<your_name>.lpp* et *bsplib_asm_<your_name>.lpp* précédemment générées et validées (retirer la librairie assembleur et inclure les sources ASM au projet si problèmes rencontrés à l'édition des liens – cf. capture d'écran ci-dessous).

  - *Projects → bluewave*
  - *Libraries (clic droit) → Add Existing Item...*
  - *Ajouter bsplib_<your_name>.lpp* puis *bsplib_asm_<your_name>.lpp*
  - *Si vous n'avez pas encore développé le BSP, ajouter les bibliothèques du BSP fournies par défaut dans disco/bsp/lib* ou *disco/bsp/lib/backup (cf.ci-dessous)*



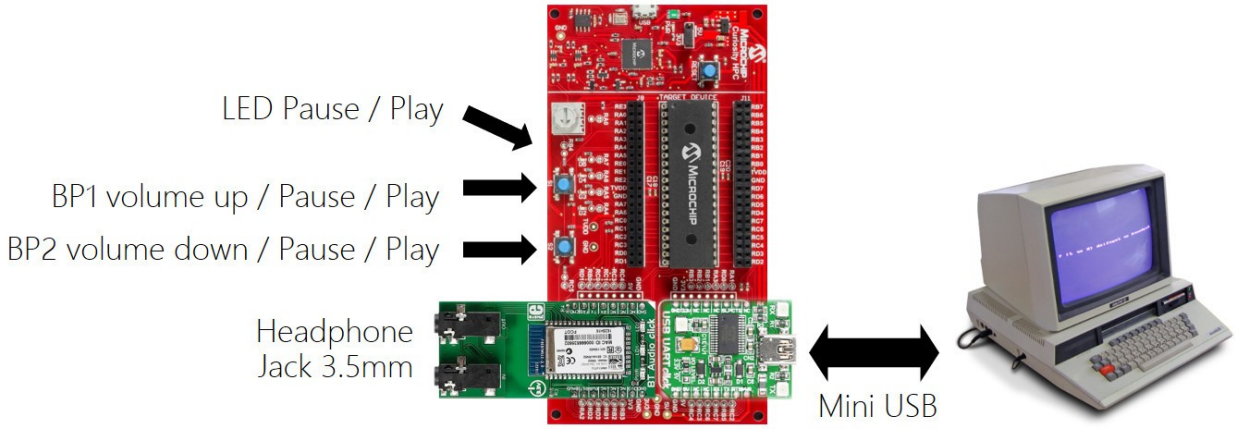
- Développer et valider une application de test implémentant le diagramme de séquence et respectant les chronogrammes suivants (ordonnancement des tâches applicatives). *Dans un premier temps, seule la validation de l'architecture logicielle de l'application nous intéresse (squelette vide). Les fonctions d'initialisation seront laissées vides, les tâches applicatives n'implémenteront qu'un délai de 1ms avec activation d'une GPIO à l'entrée et désactivation en sortie de la tâche. La référence temporel de préemption pour un ordonnanceur est le plus souvent nommée TICK (Timer Clock). Elle sera fixée à 20ms dans notre cas et sera gérée par timer. Le TICK doit être suffisamment rapide pour capter les événements externes (bouton poussoir, potentiomètre, etc), assurer une bonne réactivité du système (affichage, contrôle audio, etc), garantir un partage du temps CPU suffisant entre tâches applicatives mais cependant suffisamment lent pour maintenir le système au repos (veille) la majorité du temps pour des considérations énergétiques (nomadisme).*





8.3. Cahier des charges du POC (Proof Of Concept)

Le développement d'un voire plusieurs POC (Proof Of Concept) ou démonstrateurs est le plus souvent la première état de développement d'un produit. Le ou les POC permettent, par pas itératifs, de présenter les solutions, évolutions et cas d'usages (use case) du produit au client. A chaque POC, des redirections techniques sont alors envisagées afin de converger le plus rapidement possible et de façon efficiente au produit final qui sera envoyé en production. Nous allons ici développer un POC démontrant la faisabilité technique et technologique permettant de réaliser une enceinte Audio Bluetooth.

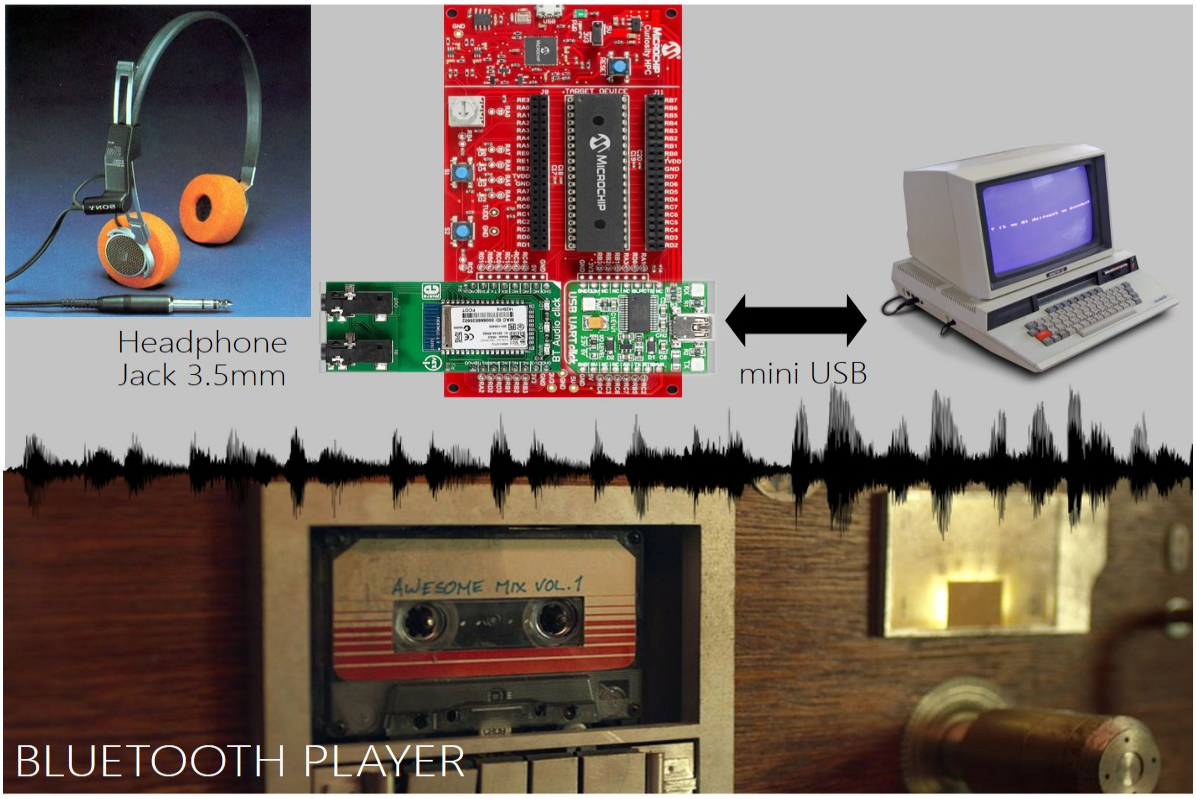


- BP1 (BP2) : Une action seule (sans action sur BP2) permet d'augmenter le volume sur le terminal Bluetooth d'émission (smartphone par exemple). Une action simultanée BP1/BP2 permet de mettre la lecture en pause. Une seconde action simultanée permet de remettre en route la lecture du média audio (Play).
- BP2 : Une action seule (sans action sur BP1) permet de diminuer le volume sur le terminal Bluetooth d'émission.
- LED : Choisir librement une LED. Lorsque qu'une lecture audio est en pause, la LED sera éteinte. Lorsqu'une lecture audio est active, la LED est allumée.
- USB to UART (console de supervision depuis un ordinateur) : Une console en sortie (visualisation seulement) sera disponible depuis un ordinateur afin d'observer les échanges entre le module RN52 et le MCU de supervision du système embarqué.
- Démarrage : Le système cherchera par défaut à se connecter au dernier appareil associé puis débutera la dernière lecture audio active.
- Énergie et sécurité (facultatif) : Tout en gardant l'architecture précédemment spécifiée, testée et validée, nous chercherons à minimiser l'empreinte énergétique de notre système embarqué. En résumé, ne travailler que lorsqu'il devient impératif de le faire. Le reste du temps, mettre le système en veille. Une fois l'application complète fonctionnelle, testée et validée, implémenter un *watchdog* en ultime sécurité.

Directives de développement

La solution proposée devra impérativement respecter le *coding styl*e présent dans le répertoire *mcu/tp/doc/misc*. La solution se voudra la plus simple, lisible, claire et efficace possible. A chaque moment du développement, votre solution doit pouvoir être présentée à un client, un collègue ou un responsable hiérarchique technique expert du domaine sans aucune honte. En résumé, pour cet exercice, fini les ''bidouilles'' de TP, s'imposer de faire le mieux possible à chaque ligne de code produite ! L'ingénierie est un art, s'efforcer de ne pas perdre cela de l'esprit ...

8.4. Développement du POC (Proof Of Concept)



- Développer la fonction ''Initialiser système'' chargée de configurer l'ensemble des fonctions matérielles périphériques et système de notre application
- Développer la fonction ''Initialiser application'' chargée de récupérer l'état au réveil du système (interfaces externes d'entrée), de mettre à jour l'affichage utilisateur (interfaces externes de sortie) puis d'initialiser la variable d'environnement de notre application
- Développer la tâche ''inputs'' chargée de récupérer l'état des interfaces externes d'entrée du système à chaque début de cycle majeur (Bouton poussoir, interrupteur, potentiomètre, etc). Laisser les codes de test permettant les mesures des temps d'exécution et retirer les délais de 1ms dans chaque tâche
- Développer la tâche ''display'' chargée de mettre à jour si nécessaire l'état des interfaces externes de sortie du système et donc l'affichage utilisateur (Afficheur LCD, LED, UART, etc)
- Développer la tâche ''Bluetooth'' chargée de mettre à jour si nécessaire l'état de la communication et de contrôler le flux audio par Bluetooth (Play, Pause, Volume, etc)

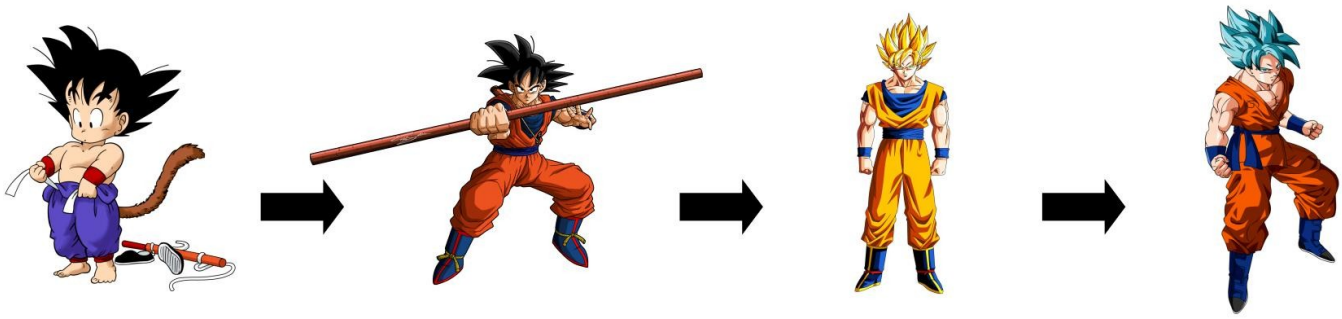
Test, mesure et validation

- Après développement et validation fonctionnelle de l'application, vérifier les durées d'exécution maximales de chaque tâche en simulant des appels des branches de code les plus longues. Reporter les mesures ci-dessous. Utiliser une LED pour valider vos mesures à l'oscilloscope. Chaque temps mesuré doit-être inférieur à 20ms :

task_inputs	task_display	task_bluetooth

8.5. Évolutions et améliorations

La question suivante peut être très complexe sachant que la solution n'est pas unique. Seule une réflexion poussée et une recherche approfondie pourraient permettre de converger vers une réponse optimale et parcimonieuse. Ce n'est pas ce qui est demandé ici, sauf si cela est votre souhait.

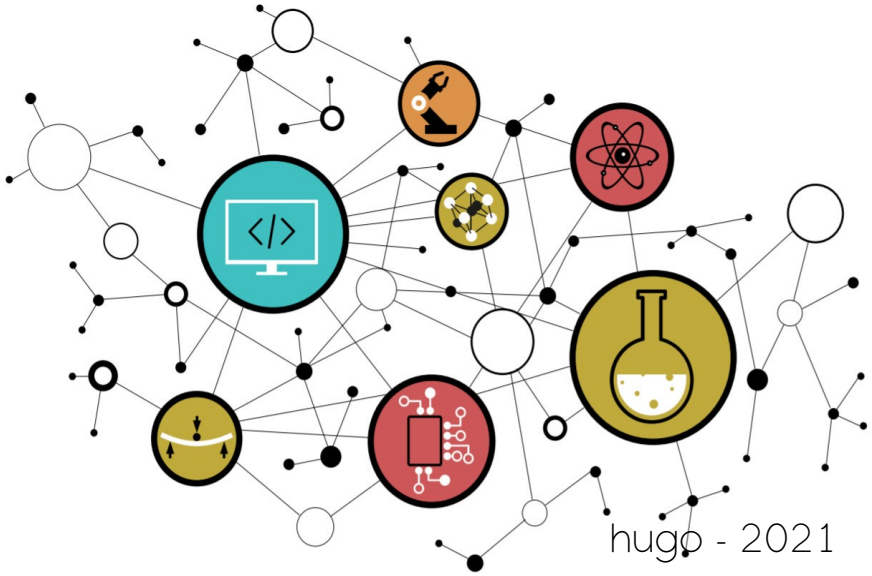


- Durant cette trame de travaux pratiques, nous nous sommes efforcés de présenter un *workflow* (méthodologie de travail) typique rencontrée dans le domaine des systèmes embarqués (développements, tests et validations unitaires, tests d'intégrations, développement de l'application, etc). La solution finale utilise des technologies actuelles du marché et pourrait être technologiquement viable pour une éventuelle mise en production et commercialisation. Cependant, cette solution n'est pas optimale. Proposer des évolutions techniques architecturales, logicielles voire matérielles afin de diminuer la consommation énergétique (système nomade sur batterie), d'améliorer la robustesse (sûreté et sécurité de la solution) voire de baisser les coûts de production (fabrication, maintenance, recyclage, etc). Aucune obligation de répondre à cette question … mais elle est le reflet d'une réflexion entre ingénieurs en entreprise !


notes

TRAVAUX PRATIQUES

DOCUMENTATION TECHNIQUE ET LIVRABLES



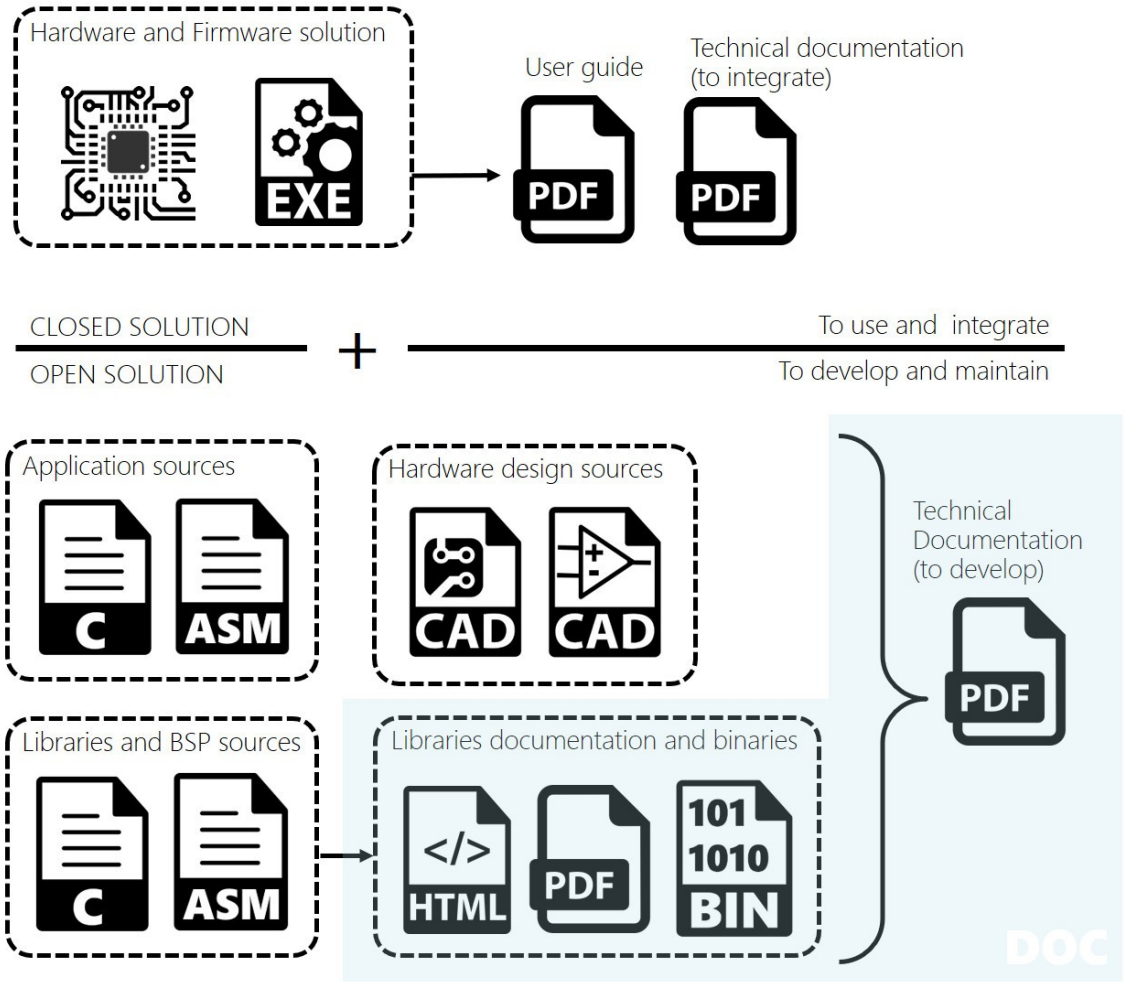
hugo - 2021

SOMMAIRE

DOCUMENTATION TECHNIQUE ET LIVRABLES

9. DOCUMENTATION TECHNIQUE ET LIVRABLES

Le schéma ci-dessous présente l'ensemble des livrables constituant une solution (système embarqué complet). En fonction du contrat signé entre client (MOA ou Maîtrise d'ouvrage) et prestataire (MOE ou Maîtrise d'œuvre), certaines parties de la solution resteront fermées.



La documentation peut être un point essentiel dans le processus de développement, d'intégration, de validation et de maintenance d'une solution. Parlons d'histoires réelles et vécues par bien des ingénieurs. Un ingénieur développe durant des semaines voire des mois une solution pour un client (sous-traitance, équipe interne, société de services, etc). Celui-ci change de mission voire d'entreprise, ou le contrat n'inclue pas forcément la maintenance. Sa solution est fonctionnelle et a été intégrée à une version donnée du produit. De même, elle était maladroitement architecturée et développée (ingénieur junior, stagiaire, incompétences techniques, etc) et maladroitement documentée. Un jour, un autre ingénieur doit se plonger dans sa solution (déverminage, correctif, évolution, intégration vers une nouvelle version du produit, portage, etc). La perte de temps peut être considérable, des jours, des semaines voire bien plus, sachant que le problème peut se reproduire. Cette histoire est très courante en industrie, de nombreux retours de ce type nous sont faits chaque année.

Cependant, pour information, dans le domaine du logiciel, le concept de ''code propre'' émerge. Il met la solution logicielle en avant et au centre de tout, et part du principe qu'une solution bien architecturée, conçue et bien codée (modularité, clarté, simplicité, séparation, etc) se suffit à elle-même. Elle ne nécessitera alors que très peu de documentation. Selon votre majeure et expertise, une formation dédiée en génie logiciel, développement agile et en code propre sera réalisée.

9.2. Doxygen et documentation d'une bibliothèque

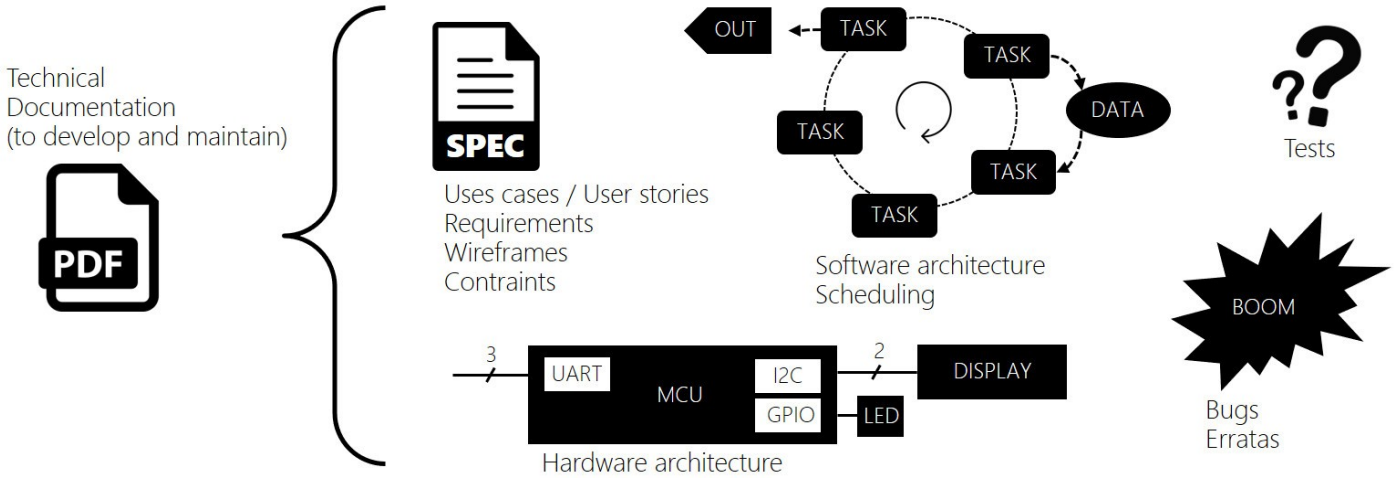
http://www.doxygen.nl

Doxygen est un générateur de documentation sous licence libre dédié à la production de documentation logicielle à partir du code source d'un programme. Il tient compte de la syntaxe du langage ainsi que des commentaires devant respecter un formalisme spécifique. Nous parlons de balises *Doxygen* (balises préfixées par @ présentes dans les fichiers d'en-têtes). Vous avez déjà rencontré ces balises dans l'ensemble des fichiers d'en-tête du BSP. Sur ce point, la majorité du travail a déjà été réalisée de notre côté. L'architecture modulaire de la documentation *Doxygen* a été conçue et développée. Le fichier racine de la documentation est le suivant *disco/bsp/doc/doxygen/doxygen.h* . Vous allez devoir maintenant générer une documentation HTML compilé (.chm) explicitement estampillée à votre nom. Ce travail passe d'abord par une phase d'installation d'outils, de création d'un projet *Doxygen* (pour génération d'un fichier *Doxyfile*) puis de génération de la documentation au format demandé.

- Installer Doxygen/Doxywizard
- Installer Dot : https://graphviz.gitlab.io/download/
  - Sous Doxywizard > Expert > Dot > DOT_PATH > C:\<*dot_path*>*bin*
- Installer HCC : https://www.microsoft.com/en-us/download/details.aspx?id=21138
  - Sous Doxywizard > Expert > HTML > HHC_LOCATION > C:/<*hhc_path*>*/hhc.exe*
  - Sous Doxywizard > Expert > HTML > CHM_FILE > *bsplib_tiny.chm*
  - Sous Doxywizard > Expert > HTML > *disable SEARCHENGINE*

- Générer une documentation *Doxygen* du BSP au format HTML compilé (.chm) nommée *bsplib_<student_name>.chm*. Le nom du projet *Doxygen* devra être ''*bsplib - <student_name> - v1.0*'' et sera sauvé, tout comme le *Doxyfile*, dans le répertoire *disco/bsp/doc/doxygen/*. Une fois générée, copier la documentation présente dans le dossier *disco/bsp/doc/doxygen/html/* dans le répertoire *disco/bsp/doc/. S'aider d'internet pour ce travail et notamment du site officiel de Doxygen*

9.3. Documentation technique



La documentation technique est directement destinée aux équipes techniques de développement et de maintenance du produit. L'objectif premier étant de permettre à un ingénieur développeur, intégrateur ou architecte d'appréhender le plus rapidement possible les architectures et solutions logicielles comme matérielles anciennement développées, ainsi que le positionnement du sous système dans un projet global de plus grande envergure.

Éditer une documentation technique présentant l'ensemble de vos développements (BSP et application Bluewave). Pour ce travail, la construction du plan et donc la définition de l'architecture du document est la première chose à réaliser. Ce travail demande un esprit de synthèse, une vision d'ensemble du projet et une approche progressive. Toujours se mettre à la place du lecteur (un ingénieur) qui découvre votre projet et qui doit comprendre rapidement où travailler et opérer. S'aider de représentations graphiques et de schémas commentés afin de présenter vos travaux de développement. Être concis et précis, aller à l'essentiel en s'efforçant d'être le plus rigoureux possible quant au vocabulaire et descriptions techniques utilisés. Voici les points à présenter :

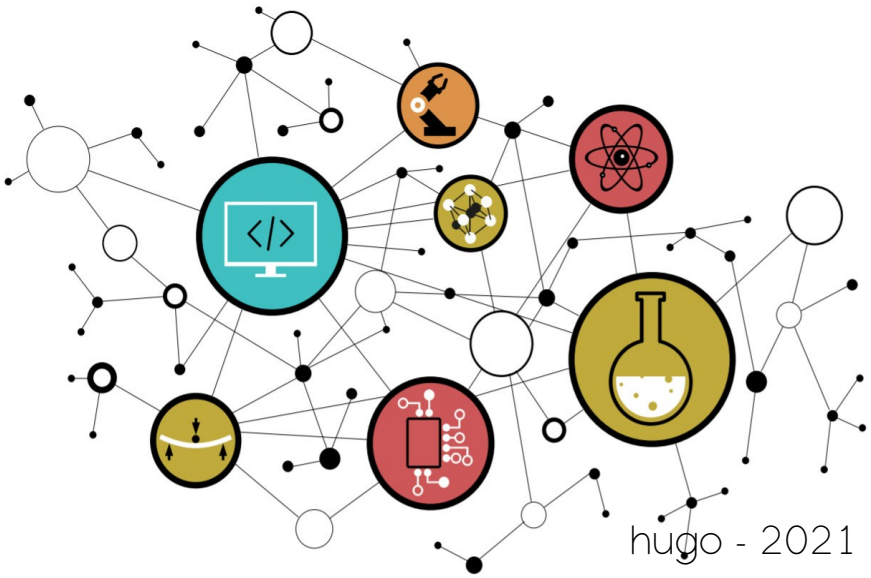
- Présenter le *projet dans son ensemble* : Expression du besoin, périmètre d'action du produit, guide d'utilisation et cas d'usages (use cases and user stories), cahier des charges, spécifications techniques et contraintes (requirements), etc
- Présenter les *outils de développement et technologies* déployées sans oublier les *versions* de chaque outil (IDE, toolchains, sonde de programmation, processeurs, cartes, etc)
- Présenter à l'aide d'un schéma fonctionnel commenté l'*architecture matérielle du produit* (interfaces externes utilisateur, bus et signaux d'interconnexion avec le processeur, périphériques internes au processeur, brochages et nommage des signaux, etc). Mettre en documents annexes les schémas électriques et layout de routage (version PDF et sources CAO).
- Présenter à l'aide d'un schéma fonctionnel commenté l'*architecture logicielle de l'application* (tâches applicatives, ressources partagées, description du travail de chaque tâche, stratégie d'ordonnancement, etc)
- Présenter les *procédures de test* de la solution (boîte blanche et/ou boîte noire) ainsi que les *résultats des mesures physiques* liés au comportement de l'application (temps d'exécution des tâches applicatives, charge CPU, réactivités des interfaces, consommation énergétique en veille et en fonctionnement, etc)
- Présenter *explicitement et clairement les bugs et erratas connus* voire des méthodes de résolution ou de contournement
- Proposer des *améliorations et évolutions techniques voire architecturales* du produit (sans pour autant avoir essayé de les développer)


notes

TRAVAUX PREPARATOIRES ET ANNEXES

TRAVAUX PRATIQUES

PREPARATIONS



hugo - 2021

SOMMAIRE

La trame de TP minimale que nous considérons comme être les compétences minimales à acquérir afin d'accéder à aux métiers de base du domaine suit le séquencent suivant : chapitres 1, 2, 3, 4, 5 et 8. Le reste de la trame ne sera pas évalué et représente une extension au jeu de compétences minimales relatif au domaine en cours d'étude (* devant chapitres facultatifs voire complémentaires). Libre à vous d'aller plus loin selon votre temps disponible et votre volonté de mieux comprendre et maîtriser ce domaine !

2. MODULE DE BROCHE GPIO ET ASSEMBLEUR PIC18

Travail préparatoire

- Installer les outils de développement sur votre ordinateur personnel. Suivre les instructions présentes sur la plateforme pédagogique d'enseignement (section *outils de développemen*t) :

  https://foad.ensicaen.fr/course/view.php?id=116

  - IDE MPLABX
  - Toolchain C 8bits XC8
  - Simulateur PICSimLab (support MCU PIC18F27K40 et carte Curiosity HPC Microchip)
  - Émulateur Null Modem com0com
  - Terminal asynchrone de communication TeraTerm
  - Driver VCP USB to UART FTDI

- Lire attentivement le document de prélude et l'introduction à ce chapitre. Ne pas hésiter à s'aider également d'internet. Pour travailler efficacement, utiliser également la documentation technique ou datasheet du MCU PIC18F27K40 présente dans *disco/tp/doc/datasheets* (lien officiel vers les ressources techniques de notre MCU PIC18F27K40 sur le site de Microchip) :

  https://www.microchip.com/wwwproducts/en/PIC18F27K40

- (1pt) Qu'est-ce qu'une GPIO ? Proposer des exemples d'utilisation dans des applications autour de vous
- (1pt) A quoi servent les registres TRISx ? Pourquoi se nomment-ils TRISx ?
- (1pt) Proposer une configuration en assembleur permettant de configurer la broche RB7 du port B en entrée, et les broches RC0 à RC2 du port C en sortie. Appliquer un niveau logique haut sur ces 3 même broches. *Tester votre programme en mode simulation et valider son fonctionnement en environnement de debug (s'aider de tutoriel internet pour réaliser ces opérations voire des annexes)*
- (1pt) Réitérer le travail précédent mais en langage C. *S'aider d'internet pour répondre à cette question*
- (1pt) Qu'est-ce qu'un registre et de quoi est-il constitué ? Pour quelles raisons tous les registres des PIC18 font 8bits ?
- (1pt) Ouvrir la documentation utilisateur de la carte de développement utilisée en TP (Curiosity HPC de Microchip). En regardant le schéma électrique, préciser sur quelles broches sont connectées les LED D2/D3/D4/D5 et les boutons poussoirs S1/S2.

  https://www.microchip.com/developmenttools/ProductDetails/dm164136

- (1pt) Sur quelle broche est connecté le bouton poussoir du reset ?

3. MODULE DE COMPTAGE TIMER ET GESTION DES INTERRUPTIONS

Travail préparatoire

- Lire attentivement l'introduction à ce chapitre. Ne pas hésiter à s'aider également d'internet. Pour travailler efficacement, utiliser également la documentation technique ou datasheet du MCU PIC18F27K40 présente dans *disco/tp/doc/datasheets*
- (1pt) Qu'est-ce qu'une IRQ ? *Proposer un schéma.Ne pas répondre uniquement Interrupt Request*
- (1pt) Qu'est-ce qu'une ISR ? *Proposer un schéma.Ne pas répondre uniquement Interrupt Service Routine*
- (1pt) Qu'est-ce qu'un vecteur d'interruption et expliquer son rôle ? *Proposer un schéma.*
- (1pt) Proposer une configuration en assembleur permettant de configurer l'interruption de priorité haute pour le Timer0 avec démasquage global d'interruption. Réitérer ce même travail pour le Timer1.
- (1pt) Proposer une configuration en assembleur pour le Timer0. Nous souhaitons lever une interruption toutes les 20ms. Nous travaillerons avec une référence d'horloge interne (CPU clock 64MHz)

4. MODULE DE COMMUNICATION UART ET LIAISON SÉRIE

Travail préparatoire

- Lire attentivement l'introduction à ce chapitre. Ne pas hésiter à s'aider également d'internet. Pour travailler efficacement, utiliser également la documentation technique ou datasheet du MCU PIC18F27K40 présente dans *disco/tp/doc/datasheets*
- (1pt) Qu'est-ce qu'un périphérique UART ? *Proposer un schéma commenté*
- (1pt) Présenter le protocole de communication d'une liaison série asynchrone. *Proposer un chronogramme commenté*
- (1pt) Proposer un chronogramme présentant l'envoi successif sans temps d'attente des caractères 'D', 'U', 'B' (8bits de donnée, 1 bit de stop et pas de bit de parité).
- (1pt) Qu'est-ce que la norme RS232 et préciser les principales spécifications techniques.
- (1pt) FTDI est un fabricant de composants électroniques. En vous rendant sur leur site web, préciser en quoi ils sont spécialisés sur le marché du silicium.
- (1pt) Quels sont les registres à configurer pour utiliser l'UART1 du MCU PIC18F27K40 utilisé en TP.
- (1pt) Proposer une configuration en assembleur de l'UART1 respectant les contraintes suivantes :
    - Protocole : *payload 8bits,1bit de stop, pas de détection d'erreur*
    - Débit : *9600KBd/s, 16bits mode, high speed mode*

5. MODULE AUDIO BLUETOOTH EXTERNE

Travail préparatoire

- Lire attentivement l'introduction à ce chapitre. Ne pas hésiter à s'aider également d'internet. Pour travailler efficacement, utiliser également la documentation technique ou datasheet du module RN52 présente dans *disco/tp/doc/datasheets*

- (2pts) Présenter et introduire la norme de communication Bluetooth ainsi que sa genèse (une page). *S'aider d'internet pour votre recherche*

- (1pt) Qu'est-ce qu'un piconet (ou picoréseau), ainsi qu'un scatternet (ou réseau dispersé) ? *Proposer impérativement un schéma afin d'illustrer votre réponse et s'aider d'internet pour votre recherche*

- (1pt) Le module bluetooth RN52 de Roving Networks (racheté par Microchip) utilisé en TP supporte la norme 3.0 du Bluetooth. Préciser les évolutions amenées par les normes Bluetooth 3.0 et 4.0 + LE (Low Energy) par rapport aux versions précédentes ? *S'aider d'internet pour votre recherche*

- (1pt) Le module Bluetooth RN52 de TP supporte le profil A2DP (stack ou bibliothèque embarquée). Qu'est-ce qu'un profil Bluetooth ? En quoi le profil A2DP est-il dédié ? *S'aider d'internet pour votre recherche*

- (1pt) Sur le schéma électrique du module BT Audio click board de Mikroelektronika (cf. *disco/bsp/doc*), nous pouvons observer des filtres analogiques en sortie du module RN52 (entre RN52 et sortie jack audio femelle 3.5mm). A quoi servent-ils ?

- (1pt) Pour cette question, s'aider du guide utilisateur présentant le jeu de commandes du module RN52 (cf. *disco/bsp/doc*). Nous utiliserons le module Bluetooth RN52 en mode commande (broche GPIO9 à l'état bas). Au démarrage, celui-ci nous retourne par UART la chaîne de caractères ''CMD\r\n''. De même, après envoi d'une commande valide, le module nous retourne la chaîne de caractères ''AOK\r\n'' (Acknowledgment OK) pour valider la bonne réception de chaque ordre reçu. Préciser les commandes à envoyer par UART afin de :

    - Sélectionner un profil A2DP à la configuration
    - Passer à la piste suivante
    - Monter le volume
    - Reconnecter le module

7. MODULE DE CONVERSION ADC

Travail préparatoire

- (2pts) L'ADC (Analog to Digital Converter) intégré dans le PIC18F27K40 de TP génère après conversion un résultat sur 10bits binaire par approximations successives. A l'aide d'un schéma commenté, des cours connexes à l'école et d'internet, expliquer le fonctionnement d'un ADC à approximation successive.

- (1pt) Sachant que la plage analogique de notre ADC sera configurée entre la masse de référence et la tension d'alimentation Vcc (plage analogique 0V-3,3V) et que nous utiliserons notre ADC en mode 8bits (plage numérique 0x00-0xFF). Représenter graphiquement la caractéristique de transfert analogique/numérique de l'ADC ainsi configuré.

- (1pt) Pour la configuration précédemment présentée, quelle est la résolution de l'ADC ?

- (1pt) Qu'est-ce qu'un échantillonneur bloqueur (s'aider des cours connexes à l'école et d'internet) ? En vous aidant de la documentation technique du MCU (section relative à l'ADC, (ADC2) Analog-to-Digital Converter with Computation Module), représenter le schéma de l'échantillonneur bloqueur intégré dans le MCU (spécifier les valeurs des composants passifs responsables des phases de précharge et d'acquisition).

- (2pts) Pour un ADC à approximation successive, par quels facteurs sera contraint le choix de la période d'échantillonnage ? *Attention, la réponse est subtile (analogique externe et interne au MCU, structure de l'ADC, etc), pas de réponse hâtive.*

- (1pt) En analysant le schéma électrique de la carte Curiosity HPC utilisée en TP, dessiner le schéma de connexion du potentiomètre présent sur la carte avec le MCU 28 broches PIC18F27K40. Quelle broche du MCU est utilisée par défaut comme entrée analogique ?

- (1pt) Proposer une configuration en assembleur des registres ADC0N0 (configuration), ADREF (référence analogique) et ADPCH (sélection du canal) assurant l'initialisation de l'ADC suivante :

    - Mode : *8bits*
    - Plage analogique : *0V-3,3V*
    - Broche : *RA0/AN0*
    - Source d'horloge : *interne FRC (Fixed Reference Clock)*
    - Divers : *mode discontinu, pas d'interruption*

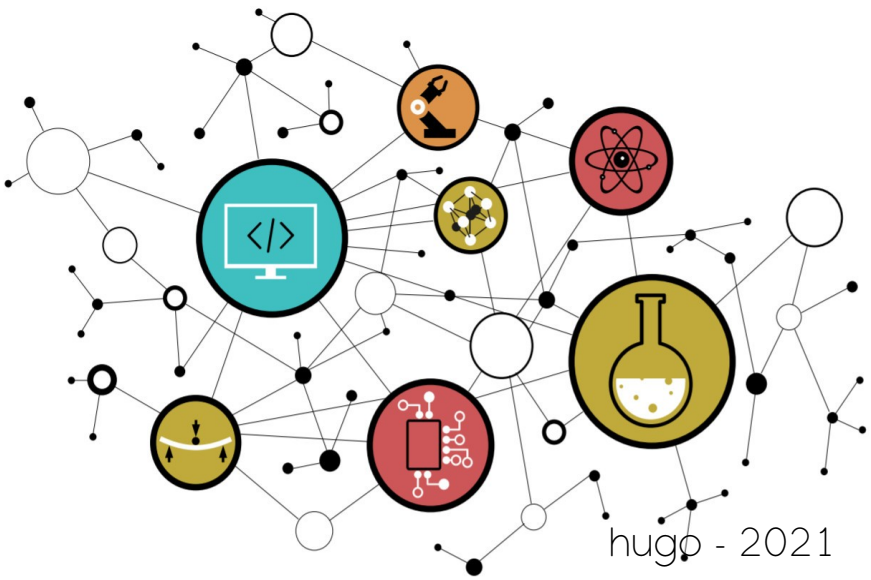
8. CONCEPTION D'UNE APPLICATION ET ORDONNANCEMENT

Travail préparatoire

- Lire attentivement l'introduction à ce chapitre.
- (1pt) Qu'est-ce qu'un système temps réel ? *S'aider d'internet*
- (1pt) Un système dit ''temps réel'' peut proposer différentes tâches ou fonctions applicatives imposant des contraintes temporelles plus ou moins impératives (dur, ferme, mou ou hard, firm, soft). Donner des exemples d'application et de produit pour chaque type de contrainte (souris, TV, système de freinage, lecteur MP3, etc) ? *S'aider d'internet*
- (1pt) Qu'est-ce qu'un ordonnanceur ou *scheduler* dans un système d'exploitation ou un système embarqué ? *S'aider d'internet*
- (1pt) Qu'est-ce qu'un ordonnanceur hors ligne ou scheduler offline ? Quels sont leurs avantages et inconvénients ? *S'aider d'internet*

TUTORIEL

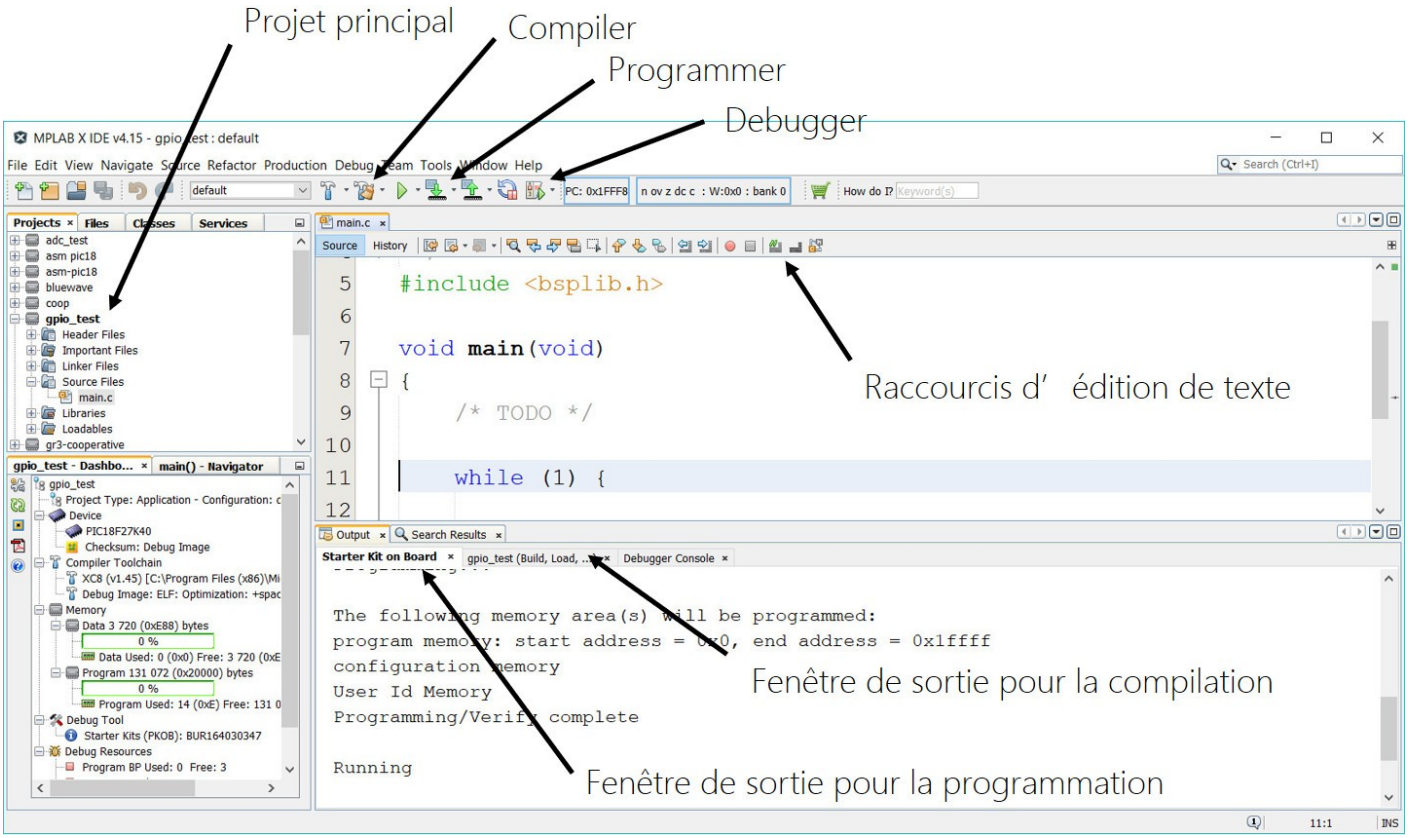
CRÉATION DE PROJET SOUS IDE MPLABX



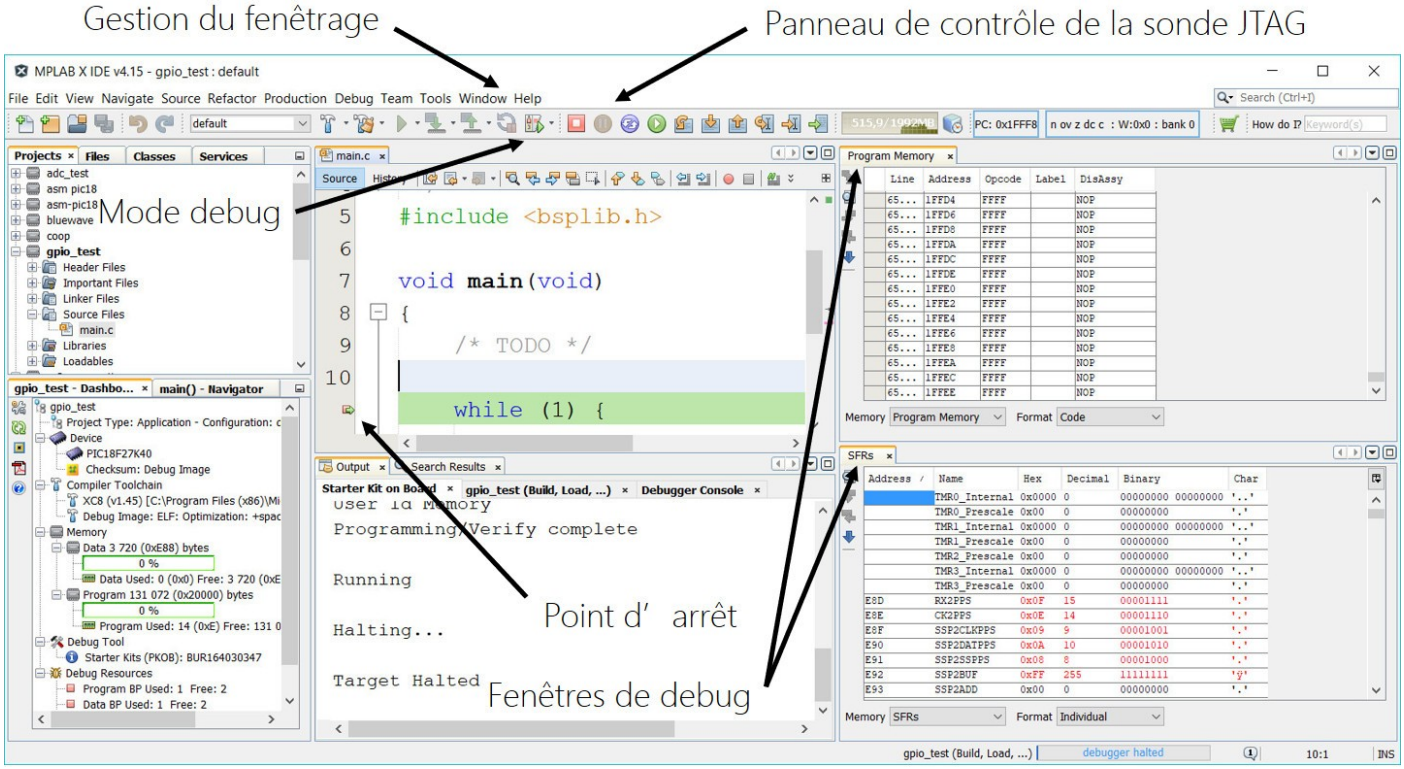
hugo - 2021

ENVIRONNEMENT GRAPHIQUE MPLABX

Environnement de développement



Environnement de debug



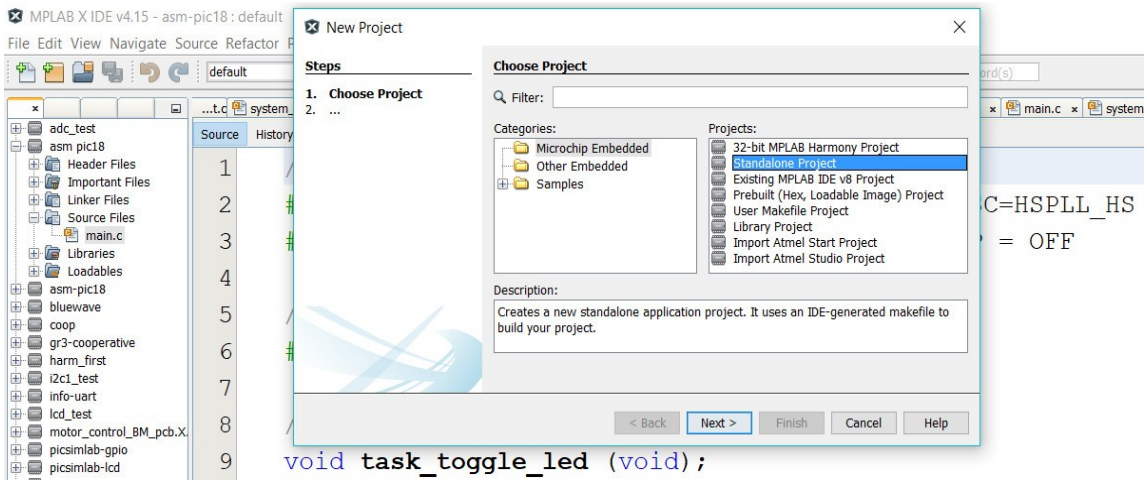
CRÉATION DE PROJET SOUS MPLABX

Il est à noter que la totalité des développements pourrait être réalisée sans environnement de développement ou IDE (Integrated Development Environment). Nous utiliserions alors la console système (UNIX ou MS-DOS) pour invoquer et paramétrer la chaîne de compilation XC8 (toolchain 8bits Microchip). De même, l'édition des fichiers sources se ferait depuis un éditeur de texte (vi, vim, nano, Notepad++, etc), l'automatisation du processus de compilation par édition d'un Makefile, etc.

Néanmoins, dans bien des cas, l'utilisation d'un IDE peut s'avérer pratique. Un IDE est dédié à centraliser et lier dans un même outil tous les services nécessaires à l'ingénieur développeur (gestion et paramétrage des chaînes de compilation, génération de Makefile, éditeur de texte intégré, interface et gestion des outils de debug, interface graphique développeur, etc).

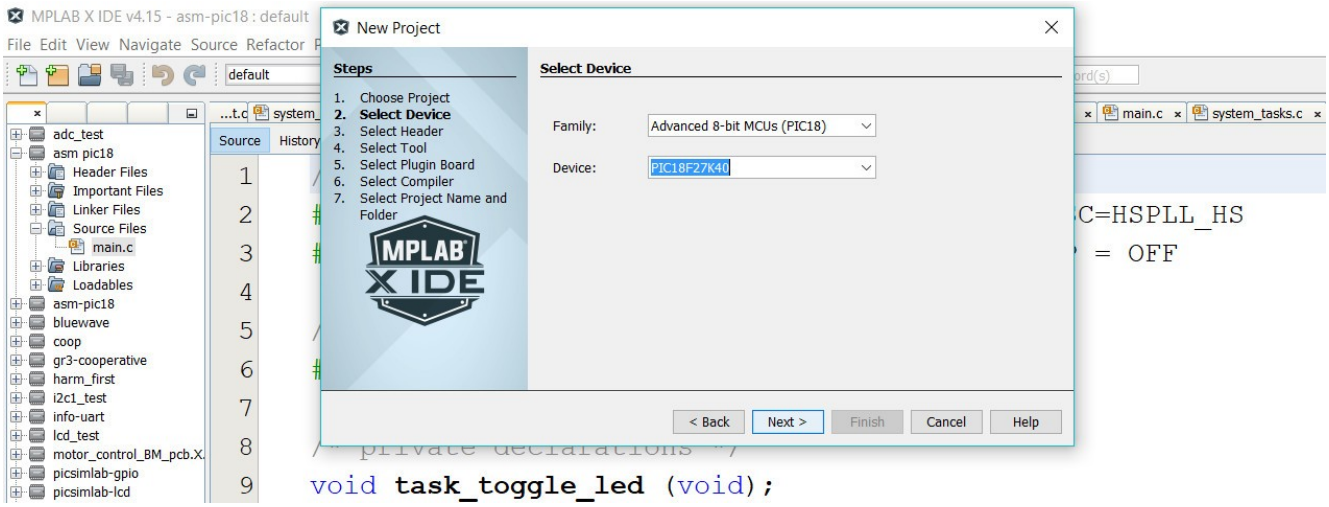
Sélection du type de projet

- Télécharger l'archive *mcu.zip* sur la plateforme *moodle* dans *download*
- *Ouvrir MPLABX → File → New Project...*
- Sélectionner le type de projet souhaité : *Standalone (développement d'application), Prebuilt (charger un firmware précompilé), Library (développement bibliothèque statique), etc*
- *Microchip Embedded → Standalone Project → Next*



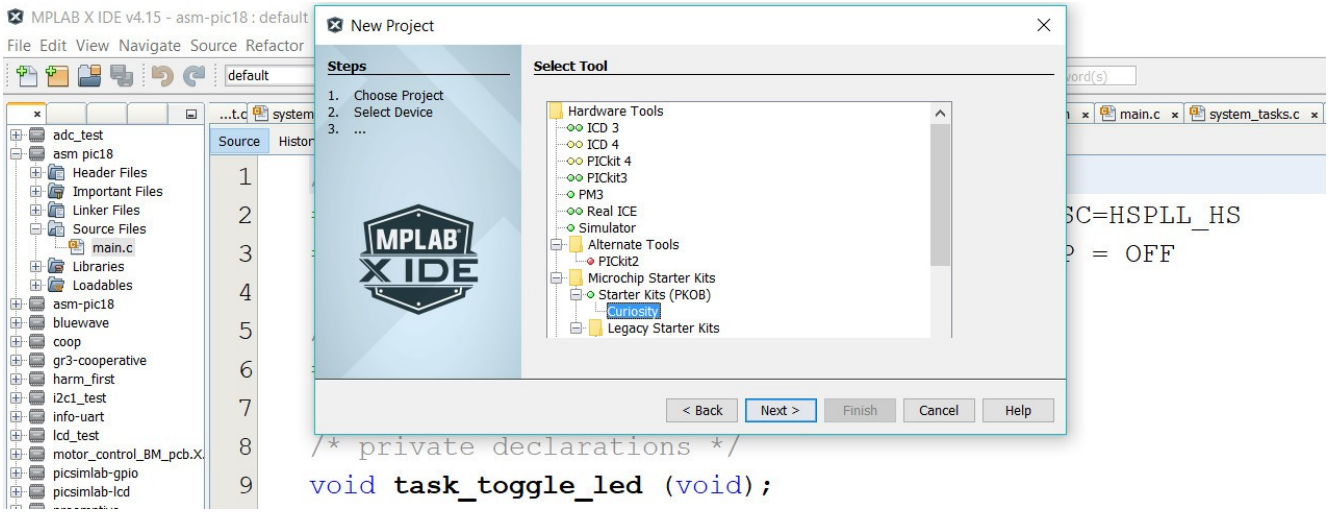
Sélection du processeur cible

- L'étape qui suit est à adapter en fonction du MCU ciblé par les développements en cours (*PIC18F27K40 en TP 1A Systèmes Embarqués*). Attention, le nom du MCU doit être impérativement le bon, sous peine de voir apparaître des erreurs au chargement sur la cible ou à la compilation. Néanmoins, toutes les étapes présentées dans ce tutoriel sont modifiables par la suite après création du projet.
- *Family → Advanced 8-bits MCUs (PIC18)*
- *Device → PIC18F27K40 → Next*

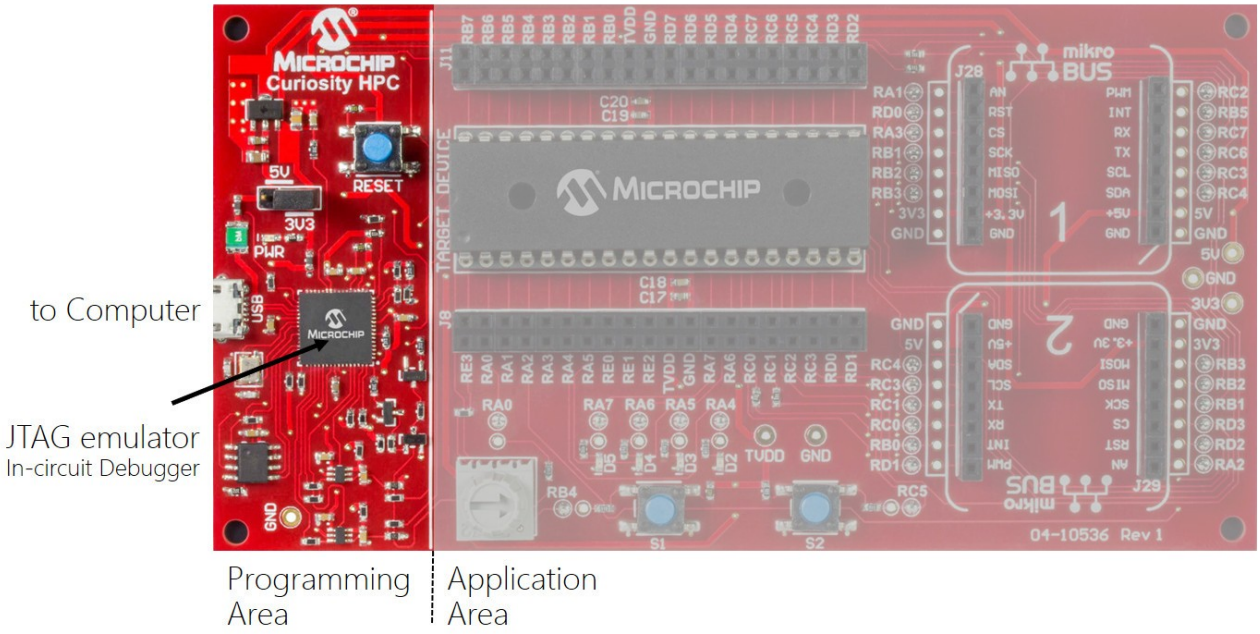


Sélection de la sonde de programmation ou du starter kit

- Sélectionner l'outil de chargement du fichier exécutable de sortie (ELF, HEX, COFF, setc) vers le processeur cible. Nous parlons de sonde JTAG (Join Test Action Group, norme IEEE 1149.1), ou sonde de programmation, ou sonde d'émulation, etc.
- *Starter Kits (PKOB) → Curiosity → Next* *(si carte physique en possession)* ou *Simulator → Next*



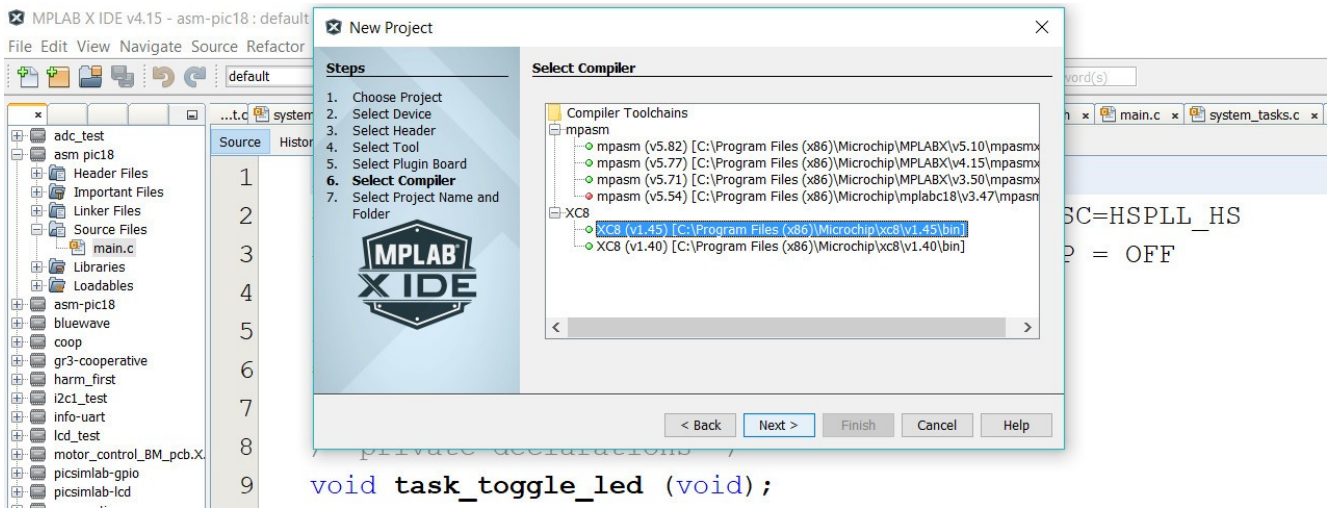
- Depuis sa normalisation en 1990, le JTAG est devenu le plus grand standard dans l'industrie pour la programmation et le test de processeur. Observons la sonde de programmation JTAG présente sur la carte de démarrage ou starter kit Curiosity HPC de Microchip



Sélection de la chaîne de compilation

- Sélectionner la chaîne de compilation. Sur un même ordinateur de développement, un certain nombre de toolchains peuvent être installées. De même, pour une même chaîne de compilation dédiée à une famille de CPU, plusieurs versions peuvent également être installées. L'exemple ci-dessous montre que l'IDE MPLABX a reconnu plusieurs toolchain C pour PIC18. Dans le cas présent, 2 versions de XC8, la chaîne de compilation développée par Microchip pour leur famille de CPU PIC18.

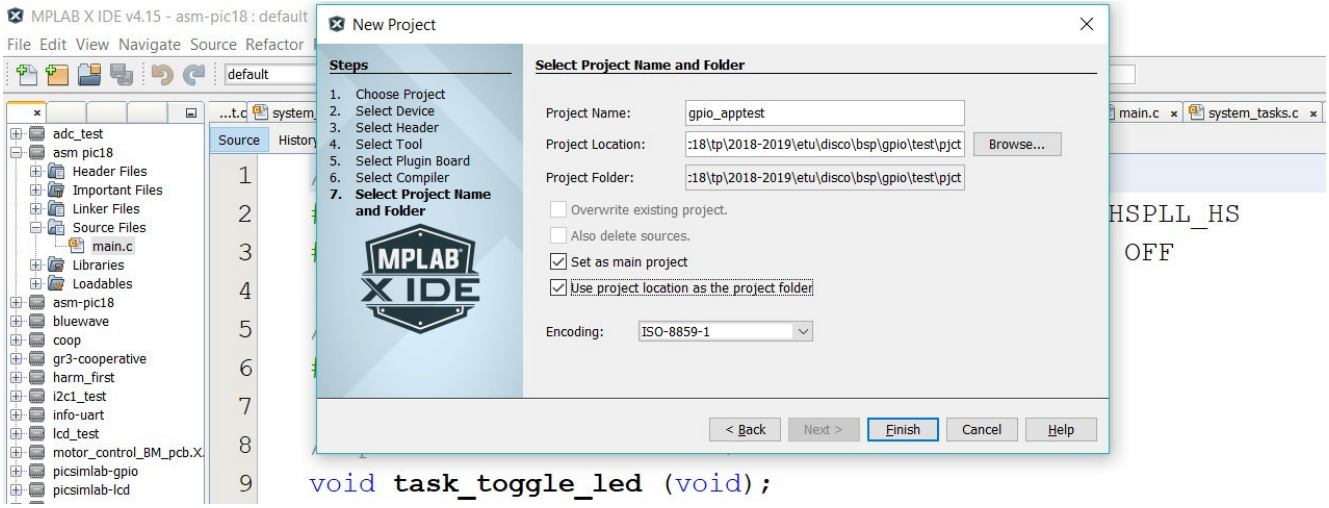
- *XC8 (v1.45) → Next*



Sélection de l'emplacement du projet

- En première année, l'arborescence du système de fichiers du projet vous est donnée et a été spécifiée par l'architecte du projet. Construire une telle arborescence demande déjà une certaine maturité en ingénierie afin de la penser parcimonieuse, simple, claire, modulable, etc. Ce travail vous sera demandé durant les projets de 2ième et 3ième année en spécialisation. Chaque sous projet possède une arborescence locale commune (cf. ci-dessous). Toujours créer vos projets dans le répertoire *pjct* correspondant au travail demandé. De même, toujours donner à vos projets un nom en lien avec les développements en cours. Par exemple, gpio, adc_pjct, uart_test et donc éviter des nommages ne permettant pas de connaître le contenu du projet, par exemple test, projet, tp1, exo3, etc :
    - *src* (fichiers sources .c)
    - *include* (fichiers d'en-tête .h)
    - *pjct* ou *test/pjct* (emplacement des fichiers internes de MPLABX, Makefile, fichiers de compilation et de production .obj, .elf, etc)

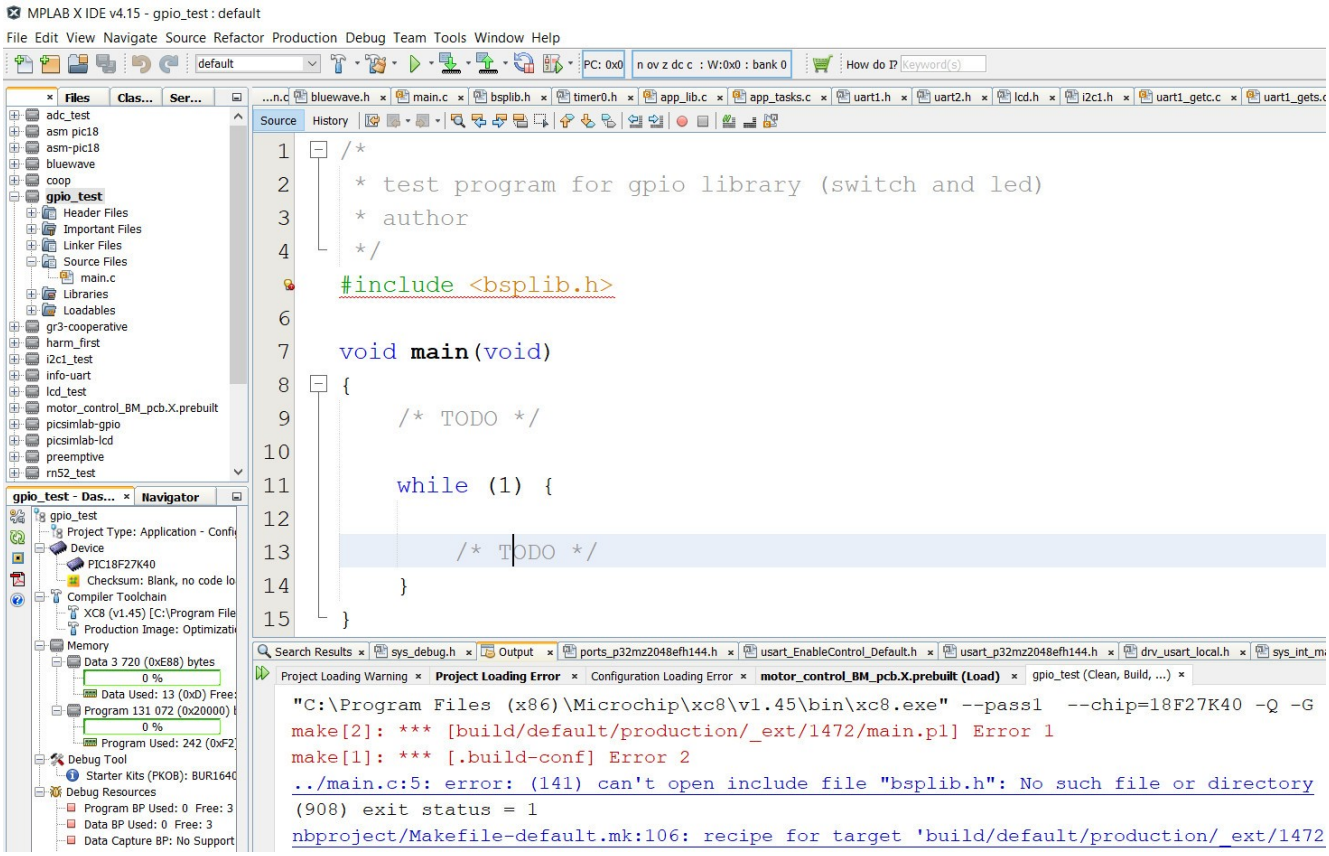
- *Project Name → <choisir_un_nom_court_sans_accent_sans espace_ayant_un_sens>*
- *Project Location → <emplacement_sur_votre_ordinateur>/mcu/tp/disco/bsp/gpio/test/pjct* (à adapter d'un projet à un autre)
- *[x] Use project location as the project folder → Finish*



- La création du projet est maintenant terminée. Reste à inclure les sources, configurer les outils de compilation et s'assurer de la bonne compilation du projet complet. Les développements pourront alors commencer. Vous pouvez d'ailleurs constater sur chaque capture d'écran précédente que de nombreux projets avaient déjà été créés sur mon ordinateur personnel de travail.

Ajouter les fichiers sources

- *Clic droit sur le nom du projet (fenêtre à gauche) → Set as Main Project*
- *Clic droit sur Source Files → Add Existing Items... → <your_parth>/test/main.c → Select*
- Compiler le projet. *Clic droit sur le nom du projet → Build* ou *cliquer le marteau et le balai*



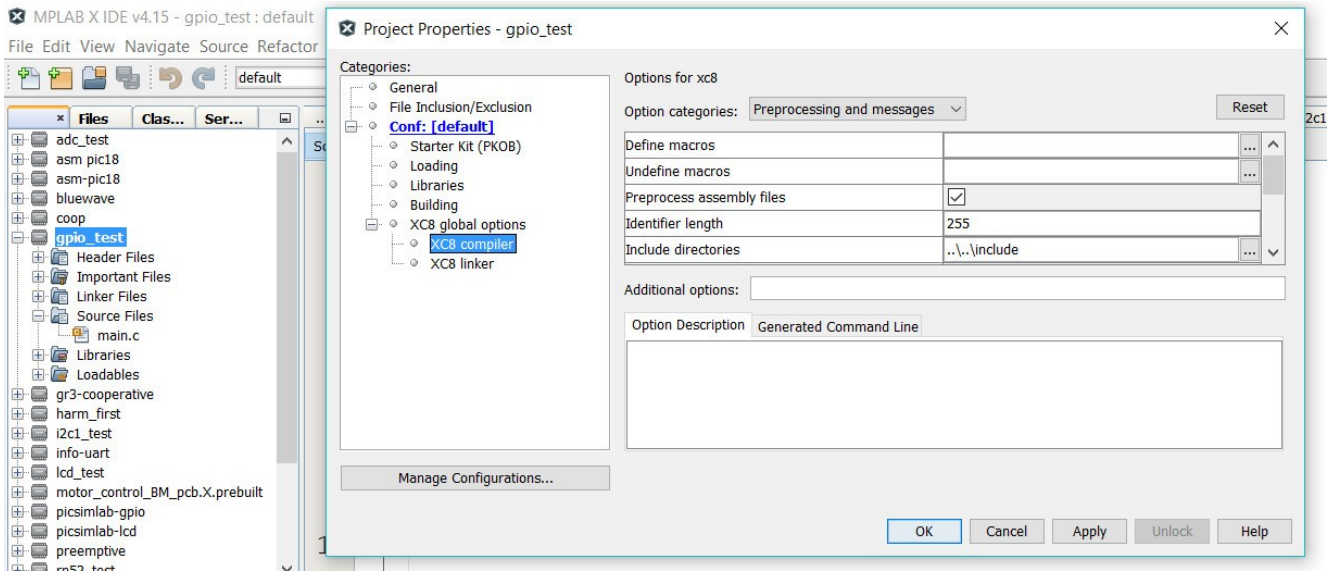
- Première erreur de compilation. Pour être plus précis, il s'agit du préprocesseur du C qui ne trouve pas le fichier *bsplib.h*. En effet, ce fichier a été développé spécifiquement pour l'application de TP sur la carte Curiosity HPC. La chaîne de compilation ne fera jamais d'hypothèse sur l'emplacement potentiel d'un fichier d'en-tête (header ou .h) applicatif. Les chemins doivent être explicitement précisés à la chaîne de compilation (toolchain) !

Configurer les outils de compilation

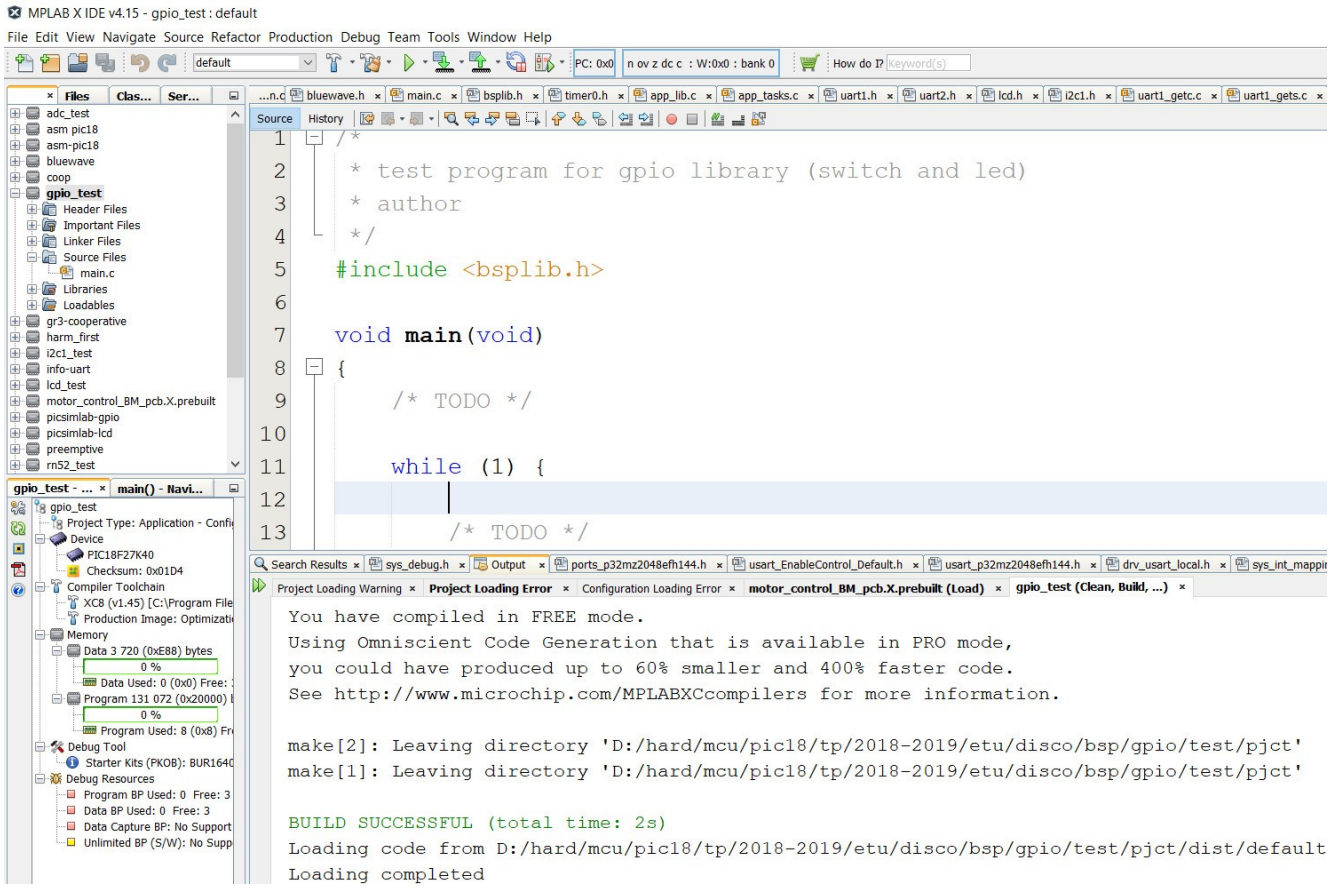
- *Clic droit sur le nom du projet → Properties*
- *Cliquer sur XC8 compiler*
- *Option Categories → Preprocessing and message*
- Vérifier l'emplacement du fichier. Recherche Windows de *bsplib.h* dans le répertoire *disco*

- *Include directories → ... → Browse... → <your_parth>/bsp/ → Apply*

- Vous constaterez que l'IDE défini par défaut des chemins relatifs à l'emplacement courant du projet sur la machine. Cela améliore la portabilité des projets de machine en machine. *Attention, le chemin illustré ci-dessous ne sera peut-être pas le même que celui sur votre machine. Tout dépend de l'emplacement où aura été créé le projet !*



- Compiler le projet. *Clic droit sur le nom du projet → Build*

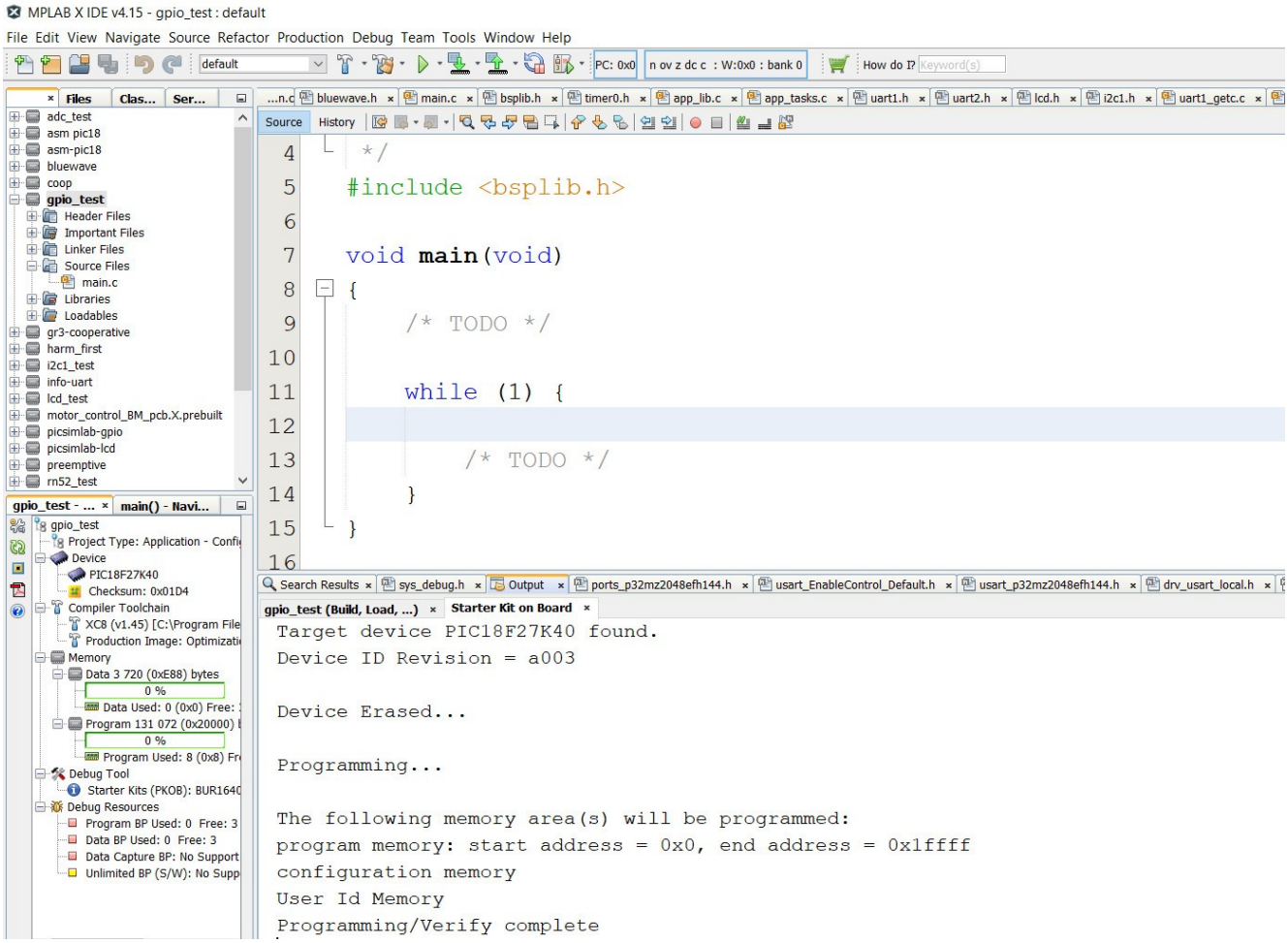


- *Le projet compile. Reste maintenant à charger le firmware de sortie sur la cible !*

Charger et exécuter le programme sur la cible

- Une fois la compilation du projet validée, il reste à charger le firmware en mémoire programme flash interne du MCU (ou sur simulateur logique). Nous disons souvent ''flasher'' le processeur. Le transfert se fera par la sonde JTAG précédemment présentée. Au chargement, vous constaterez que la fenêtre de sortie de l'IDE commutera de la fenêtre ''build, load,...'' (compilation) à ''Starter Kit on Board'' (programmation et exécution). L'IDE est alors en train d'échanger avec la sonde JTAG.

- *Clic droit sur le nom du projet → Run* ou icône ci-dessous ''Fichier avec flèche verte vers le bas en direction de la puce du microcontrôleur''



- *L'affichage **Programming/Verify complete** spécifie le bon chargement du firmware dans le MCU cible et le début de son exécution. Il reste maintenant à vérifier le bon fonctionnement de votre application !*

Test sur carte physique Curiosity HPC

- Ouvrir les propriétés du projet : *Fenêtre Projects > clic droit sur le nom du projet > Properties*
- Sélectionner la carte Curiosity HPC : *Hardware Tool > Microchip Kits > Curiosity > Apply > OK*
- Compiler et exécuter le programme : *Fenêtre Project > clic droit sur le nom du projet > Run*

Test sur simulateur MPLABX IDE (sans carte de développement)

- Ouvrir les propriétés du projet : *Fenêtre Projects > clic droit sur le nom du projet > Properties*
- Sélectionner le simulateur : *Hardware Tool > Simulator > Apply > OK*
- Compiler et exécuter le programme : *Fenêtre Projects > clic droit sur le nom du projet > Debug*
- Arrêter la simulation en cliquant sur le bouton carré rouge STOP (ou [Shift] + [F5])

Test sur simulateur de carte PICSimLab

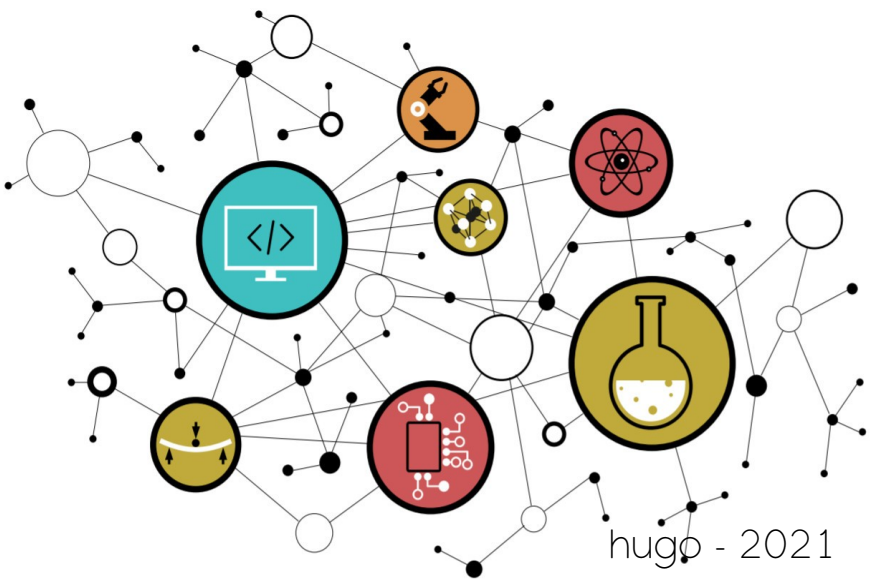
- Sous MPLABX, compiler le programme : *Fenêtre Projects > clic droit sur le nom du projet > Clean and Build*
- Sous MPLABX, observer après compilation le dossier où a été sauvé le fichier .hex de sortie:

```
BUILD SUCCESSFUL
Loading code from /<your_computer_path>/dist/default/production/<your_project_name>.hex
```

- *Ouvrir le simulateur de carte PICSIMLab : File > Load Hex > charger le fichier .hex précédemment généré.*
- *Ne pas hésiter à utiliser le module oscilloscope pour analyser la sortie : Modules > Oscilloscope*

TUTORIEL

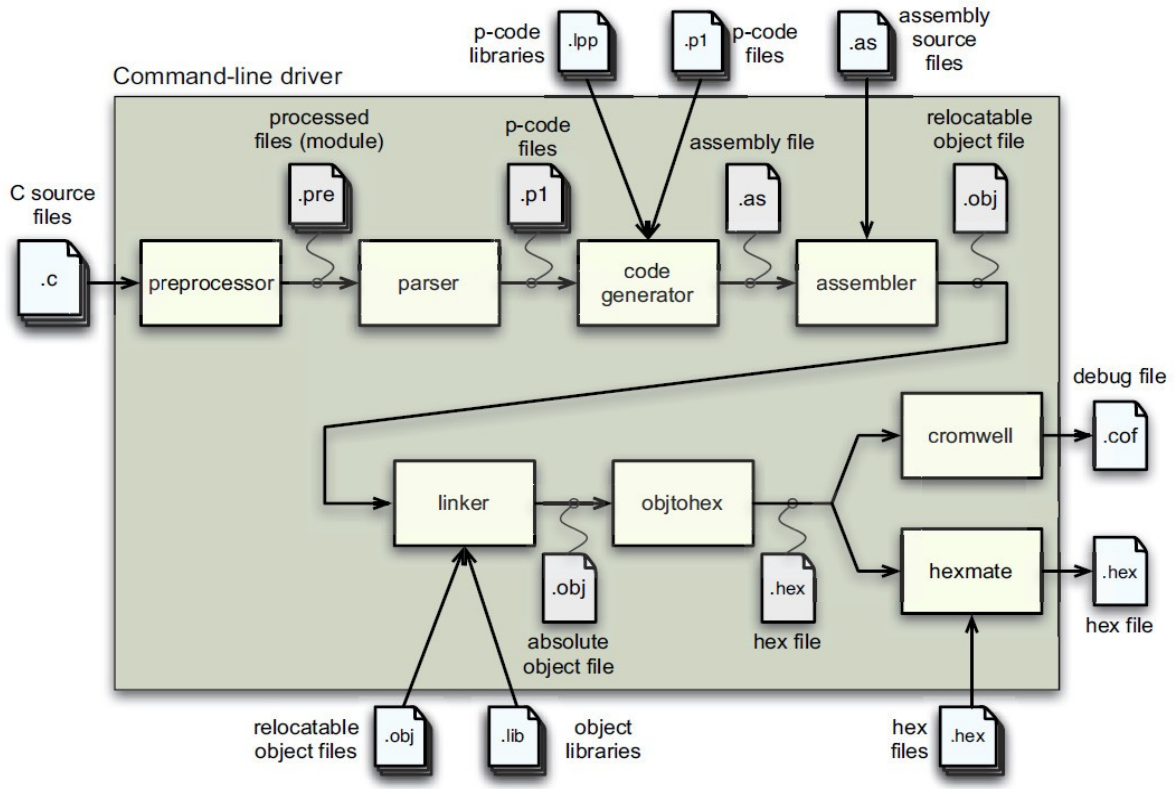
SYNTHÈSE PIC18F27K40 ET TOOLCHAIN XC8



hugo - 2021

SYNTHÈSE PIC18 ET XC8

CHAÎNE DE COMPILATION XC8



Tailles des types de donnée

Type	Size (bits)	Arithmetic Type
`bit`	1	Unsigned integer
`signed char`	8	Signed integer
`unsigned char`	8	Unsigned integer
`signed short`	16	Signed integer
`unsigned short`	16	Unsigned integer
`signed int`	16	Signed integer
`unsigned int`	16	Unsigned integer
`signed short long`	24	Signed integer
`unsigned short long`	24	Unsigned integer
`signed long`	32	Signed integer
`unsigned long`	32	Unsigned integer
`signed long long`	32	Signed integer
`unsigned long long`	32	Unsigned integer

MCU PIC18F27K40

MICROCHIP

PIC18(L)F27/47K40

28/40/44-Pin, Low-Power, High-Performance Microcontrollers with XLP Technology

Description CPU et mémoire

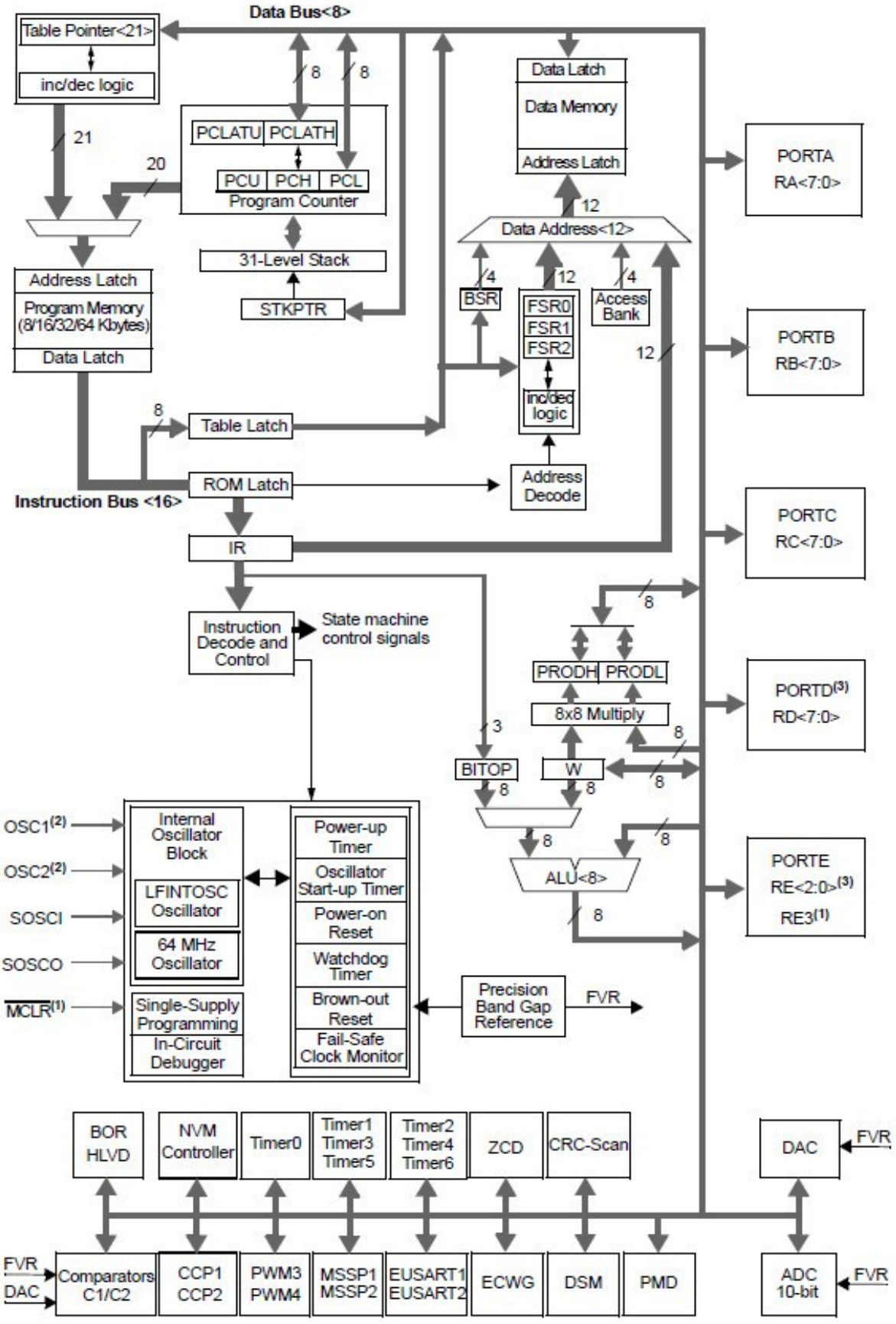
Core Features

- C Compiler Optimized RISC Architecture
- Operating Speed:
  - DC – 64 MHz clock input over the full V_{DD} range
  - 62.5 ns minimum instruction cycle
- Programmable 2-Level Interrupt Priority
- 31-Level Deep Hardware Stack
- Three 8-Bit Timers (TMR2/4/6) with Hardware Limit Timer (HLT)
- Four 16-Bit Timers (TMR0/1/3/5)
- Low-Current Power-on Reset (POR)
- Power-up Timer (PWRT)
- Brown-out Reset (BOR)
- Low-Power BOR (LPBOR) Option
- Windowed Watchdog Timer (WWDT):
  - Watchdog Reset on too long or too short interval between watchdog clear events
  - Variable prescaler selection
  - Variable window size selection
  - All sources configurable in hardware or software

Memory

- 128K Bytes Program Flash Memory
- 3728 Bytes Data SRAM Memory

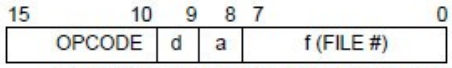
ARCHITECTURE INTERNE MCU PIC18F27K40



ASSEMBLEUR PIC18

Formats binaires des instructions

Byte-oriented file register operations

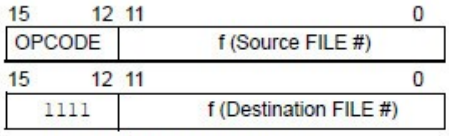


d = 0 for result destination to be WREG register
d = 1 for result destination to be file register (f)
a = 0 to force Access Bank
a = 1 for BSR to select bank
f = 8-bit file register address

Example Instruction

```
ADDWF MYREG, W, B
```

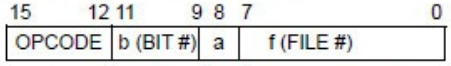
Byte to Byte move operations (2-word)



f = 12-bit file register address

```
MOVFF MYREG1, MYREG2
```

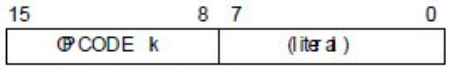
Bit-oriented file register operations



b = 3-bit position of bit in file register (f)
a = 0 to force Access Bank
a = 1 for BSR to select bank
f = 8-bit file register address

```
BSF MYREG, bit, B
```

Literal operations

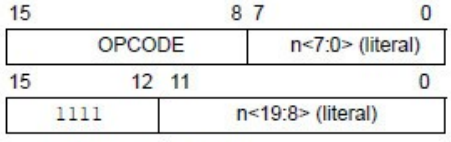


k = 8-bit immediate value

```
MOVLW 7Fh
```

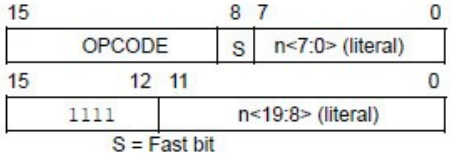
Control operations

CALL, GOTO and Branch operations



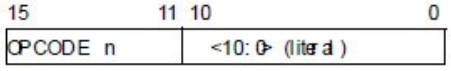
n = 20-bit immediate value

```
GOTO Label
```

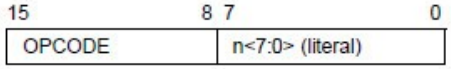


S = Fast bit

```
CALL MYFUNC
```




```
BRA MYFUNC
```




```
BC MYFUNC
```

Jeu d'instructions PIC18

Mnemonic, Operands		Description	Cycles	16-Bit Instruction Word				Status Affected	Notes
				MSb			LSb		
BYTE-ORIENTED OPERATIONS									
ADDWF	f, d, a	Add WREG and f	1	0010	01da	ffff	ffff	C, DC, Z, OV, N	1, 2
ADDWFC	f, d, a	Add WREG and CARRY bit to f	1	0010	00da	ffff	ffff	C, DC, Z, OV, N	1, 2
ANDWF	f, d, a	AND WREG with f	1	0001	01da	ffff	ffff	Z, N	1, 2
CLRF	f, a	Clear f	1	0110	101a	ffff	ffff	Z	2
COMF	f, d, a	Complement f	1	0001	11da	ffff	ffff	Z, N	1, 2
CPFSEQ	f, a	Compare f with WREG, skip =	1 (2 or 3)	0110	001a	ffff	ffff	None	4
CPFSGT	f, a	Compare f with WREG, skip >	1 (2 or 3)	0110	010a	ffff	ffff	None	4
CPFSLT	f, a	Compare f with WREG, skip <	1 (2 or 3)	0110	000a	ffff	ffff	None	1, 2
DECF	f, d, a	Decrement f	1	0000	01da	ffff	ffff	C, DC, Z, OV, N	1, 2, 3, 4
DECFSZ	f, d, a	Decrement f, Skip if 0	1 (2 or 3)	0010	11da	ffff	ffff	None	1, 2, 3, 4
DCFSNZ	f, d, a	Decrement f, Skip if Not 0	1 (2 or 3)	0100	11da	ffff	ffff	None	1, 2
INCF	f, d, a	Increment f	1	0010	10da	ffff	ffff	C, DC, Z, OV, N	1, 2, 3, 4
INCFSZ	f, d, a	Increment f, Skip if 0	1 (2 or 3)	0011	11da	ffff	ffff	None	4
INFSNZ	f, d, a	Increment f, Skip if Not 0	1 (2 or 3)	0100	11da	ffff	ffff	None	1, 2
IORWF	f, d, a	Inclusive OR WREG with f	1	0001	00da	ffff	ffff	Z, N	1, 2
MOVF	f, d, a	Move f	1	0101	00da	ffff	ffff	Z, N	1
MOVFF	f_s, f_d	Move f_s (source) to 1st word	2	1100	ffff	ffff	ffff	None	

Jeu d'instructions PIC18

Mnemonic, Operands		Description	Cycles	16-Bit Instruction Word				Status Affected	Notes
				MSb			LSb		
		f_d (destination) 2nd word		1111	ffff	ffff	ffff		
MOVWF	f, a	Move WREG to f	1	0110	111a	ffff	ffff	None	
MULWF	f, a	Multiply WREG with f	1	0000	001a	ffff	ffff	None	1, 2
NEGF	f, a	Negate f	1	0110	110a	ffff	ffff	C, DC, Z, OV, N	
RLCF	f, d, a	Rotate Left f through Carry	1	0011	01da	ffff	ffff	C, Z, N	1, 2
RLNCF	f, d, a	Rotate Left f (No Carry)	1	0100	01da	ffff	ffff	Z, N	
RRCF	f, d, a	Rotate Right f through Carry	1	0011	00da	ffff	ffff	C, Z, N	
RRNCF	f, d, a	Rotate Right f (No Carry)	1	0100	00da	ffff	ffff	Z, N	
SETF	f, a	Set f	1	0110	00da	ffff	ffff	None	1, 2
SUBFWB	f, d, a	Subtract f from WREG with borrow	1	0101	01da	ffff	ffff	C, DC, Z, OV, N	
SUBWF	f, d, a	Subtract WREG from f	1	0101	11da	ffff	ffff	C, DC, Z, OV, N	1, 2
SUBWFB	f, d, a	Subtract WREG from f with borrow	1	0101	10da	ffff	ffff	C, DC, Z, OV, N	
SWAPF	f, d, a	Swap nibbles in f	1	0011	10da	ffff	ffff	None	4
TSTFSZ	f, a	Test f, skip if 0	1 (2 or 3)	0110	011a	ffff	ffff	None	1, 2
XORWF	f, d, a	Exclusive OR WREG with f	1	0001	10da	ffff	ffff	Z, N	
BIT-ORIENTED OPERATIONS									
BCF	f, b, a	Bit Clear f	1	1001	bbba	ffff	ffff	None	1, 2
BSF	f, b, a	Bit Set f	1	1000	bbba	ffff	ffff	None	1, 2

Jeu d'instructions PIC18

Mnemonic, Operands		Description	Cycles	16-Bit Instruction Word				Status Affected	Notes
				MSb			LSb		
BTFSC	f, b, a	Bit Test f, Skip if Clear	1 (2 or 3)	1011	bbba	ffff	ffff	None	3, 4
BTFSS	f, b, a	Bit Test f, Skip if Set	1 (2 or 3)	1010	bbba	ffff	ffff	None	3, 4
BTG	f, b, a	Bit Toggle f	1	0111	bbba	ffff	ffff	None	1, 2
CONTROL OPERATIONS									
BC	n	Branch if Carry	1 (2)	1110	0010	nnnn	nnnn	None	4
BN	n	Branch if Negative	1 (2)	1110	0110	nnnn	nnnn	None	
BNC	n	Branch if Not Carry	1 (2)	1110	0011	nnnn	nnnn	None	
BNN	n	Branch if Not Negative	1 (2)	1110	0111	nnnn	nnnn	None	
BNOV	n	Branch if Not Overflow	1 (2)	1110	0101	nnnn	nnnn	None	
BNZ	n	Branch if Not Zero	1 (2)	1110	0001	nnnn	nnnn	None	
BOV	n	Branch if Overflow	1 (2)	1110	0100	nnnn	nnnn	None	
BRA	n	Branch Unconditionally	2	1101	0nnn	nnnn	nnnn	None	
BZ	n	Branch if Zero	1 (2)	1110	0000	nnnn	nnnn	None	
CALL	k, s	Call subroutine 1st word	2	1110	110s	kkkk	kkkk	None	
		2nd word		1111	kkkk	kkkk	kkkk		
CLRWDT	—	Clear Watchdog Timer	1	0000	0000	0000	0100	$\overline{TO}$, $\overline{PD}$	
DAW	—	Decimal Adjust WREG	1	0000	0000	0000	0111	C	
GOTO	k	Go to address 1st word	2	1110	1111	kkkk	kkkk	None	
		2nd word		1111	kkkk	kkkk	kkkk		

Jeu d'instructions PIC18

Mnemonic, Operands		Description	Cycles	16-Bit Instruction Word				Status Affected	Notes
				MSb			LSb		
NOP	—	No Operation	1	0000	0000	0000	0000	None	
NOP	—	No Operation	1	1111	xxxx	xxxx	xxxx	None	
POP	—	Pop top of return stack (TOS)	1	0000	0000	0000	0110	None	
PUSH	—	Push top of return stack (TOS)	1	0000	0000	0000	0101	None	
RCALL	n	Relative Call	2	1101	1nnn	nnnn	nnnn	None	
RESET		Software device Reset	1	0000	0000	1111	1111	All	
RETFIE	s	Return from interrupt enable	2	0000	0000	0001	000s	GIE/GIEH, PEIE/GIEL	
RETLW	k	Return with literal in WREG	2	0000	1100	kkkk	kkkk	None	
RETURN	s	Return from Subroutine	2	0000	0000	0001	001s	None	
SLEEP	—	Go into Standby mode	1	0000	0000	0000	0011	$\overline{TO}$, $\overline{PD}$	
LITERAL OPERATIONS									
ADDLW	k	Add literal and WREG	1	0000	1111	kkkk	kkkk	C, DC, Z, OV, N	
ANDLW	k	AND literal with WREG	1	0000	1011	kkkk	kkkk	Z, N	
IORLW	k	Inclusive OR literal with WREG	1	0000	1001	kkkk	kkkk	Z, N	
LFSR	f, k	Move literal (12-bit) 2nd word	2	1110	1110	00ff	kkkk	None	
		to FSR(f) 1st word		1111	0000	kkkk	kkkk		
MOVLB	k	Move literal to BSR<3:0>	1	0000	0001	0000	kkkk	None	
MOVLW	k	Move literal to WREG	1	0000	1110	kkkk	kkkk	None	
MULLW	k	Multiply literal with WREG	1	0000	1101	kkkk	kkkk	None	

Jeu d'instructions PIC18

Mnemonic, Operands		Description	Cycles	16-Bit Instruction Word				Status Affected	Notes
				MSb			LSb		
RETLW	k	Return with literal in WREG	2	0000	1100	kkkk	kkkk	None	
SUBLW	k	Subtract WREG from literal	1	0000	1000	kkkk	kkkk	C, DC, Z, OV, N	
XORLW	k	Exclusive OR literal with WREG	1	0000	1010	kkkk	kkkk	Z, N	
DATA MEMORY ↔ PROGRAM MEMORY OPERATIONS									
TBLRD*		Table Read	2	0000	0000	0000	1000	None	
TBLRD*+		Table Read with post-increment		0000	0000	0000	1001	None	
TBLRD*-		Table Read with post-decrement		0000	0000	0000	1010	None	
TBLRD+*		Table Read with pre-increment		0000	0000	0000	1011	None	
TBLWT*		Table Write	2	0000	0000	0000	1100	None	
TBLWT*+		Table Write with post-increment		0000	0000	0000	1101	None	
TBLWT*-		Table Write with post-decrement		0000	0000	0000	1110	None	
TBLWT+*		Table Write with pre-increment		0000	0000	0000	1111	None	

Note:

1. When a PORT register is modified as a function of itself (e.g., MOVF PORTB, 1, 0), the value used will be that value present on the pins themselves. For example, if the data latch is '1' for a pin configured as input and is driven low by an external device, the data will be written back with a '0'.
2. If this instruction is executed on the TMR0 register (and where applicable, 'd' = 1), the prescaler will be cleared if assigned.
3. If Program Counter (PC) is modified or a conditional test is true, the instruction requires two cycles. The second cycle is executed as a `NOP`.
4. Some instructions are two-word instructions. The second word of these instructions will be executed as a `NOP` unless the first word of the instruction retrieves the information embedded in these 16 bits. This ensures that all program memory locations have a valid instruction.

GLOSSAIRE Systèmes Embarqués

A

- ABI : Application Binary Interface
- ADC : Analog to Digital Converter
- ALU : Arithmetic and Logical Unit
- AMD : Advanced Micro Devices
- ANSI : American National Standards Institute
- API : Application Programming Interface
- APU : Accelerrated Processor Unit
- ARM : société anglaise proposant des architectures CPU RISC 32bits
- ASCII : American Standar Code for Information Interchange

B

- BP : Base Pointer
- BSL : Board Support Library
- BSP : Board Support Package

C

- CCS : Code Composer Studio
- CEM : Compatibilité ElectroMagnétique
- CISC : Complex Instruction Set Computer
- CPU : Central Processing Unit
- CSL : Chip Support Library

D

- DAC : Digital to Analog Converter
- DDR : Double Data Rate
- DDR SDRAM: Double Data Rate Synchronous Dynamic Random Access Memory
- DMA : Direct Memory Access
- DSP : Digital Signal Processor
- DSP : Digital Signal Processing

GLOSSAIRE Systèmes Embarqués

E

- EDMA : Enhanced Direct Memory Access
- EUSART : Enhanced Universal Synchronous Asynchronous Receiver Transmitter
- EMIF : External Memory Interface
- EPIC : Explicitly Parallel Instruction Computing

F

- FPU : Floating Point Unit
- FLOPS : Floating-Point Operations Per Second
- FMA: Fused Multiply-Add

G

- GCC : Gnu Collection Compiler
- GLCD : Graphical Liquid Crytal Display
- GNU : GNU's Not UNIX
- GPIO : General Purpose Input Output
- GPP : General Purpose Processor
- GPU : Graphical Processing Unit

I

- IA-64 : Intel Architecture 64bits
- I2C : Inter Integrated Circuit
- ICC : Intel C++ Compiler
- ICC : Interface Chaise Clavier – les problèmes viennent le plus souvent de cette interface !
- IDE : Integrated Development Environment
- IDMA : Internal Direct memory Access
- IRQ : Interrupt ReQuest
- ISR : Interrupt Software Routine
- ISR : Interrupt Service Routine

L

- L1D : Level 1 Data Memory
- L1I : Level 1 Instruction Memory (idem L1P)
- L1P : Level 1 Program Memory (idem L1I)
- Lx : Level x Memory
- LCD : Liquid Crytal Display
- LRU : Least Recently Used

M

- MAC: Multiply Accumulate
- MCU : Micro Controller Unit
- MIMD : Multiple Instructions on Multiple Data
- MIPS : Mega Instructions Per Second

- MMU : Memory Managment Unit
- MPLABX : MicrochiP LABoratory 10, IDE Microchip
- MPU : Micro Processor Unit ou GPP
- MPU : Memory Protect Unit

O

- OS : Operating System

P

- PC : Program Counter
- PC : Personal Computer
- PIC18 : Famille MCU 8bits Microchip
- PLD : Programmable Logic Device
- POSIX : Portable Operating System Interface, héritage d'UNIX (norme IEEE 1003)
- PPC : Power PC

R

- RAM : Random Access Memory
- RISC : Reduced Instruction Set Computer
- RS232 : Norme standardisant un protocole de communication série asynchrone
- RTOS : Real Time Operating System

S

- SDK : Software Development Kit
- SIMD : Single Instruction Multiple Date
- SIP : System In Package
- SOB : System On Board
- SOC : System On Chip
- SOP : Sums of products
- SP : Stack Pointer
- SP : Serial Port
- SPI : Serial Peripheral Interface
- SRAM : Static Random Access Memory
- SSE : Streaming SIMD Extensions
- STM32 : STMicroelectronics 32bits MCU

T

- TI : Texas Instruments
- TNS : Traitement Numérique du Signal
- TSC : Time Stamp Counter
- TTM : Time To Market

U

- UART : Universal Asynchronous Receiver Transmitter
- USB : Universal Serial Bus

V

- VHDL : VHSIC Hardware Description langage
- VHSIC : Very High Speed Integrated Circuit
- VLIW : Very Long Intruction Word

PRISE DE NOTES

PRISE DE NOTES

PRISE DE NOTES

PRISE DE NOTES

PRISE DE NOTES

PRISE DE NOTES

PRISE DE NOTES