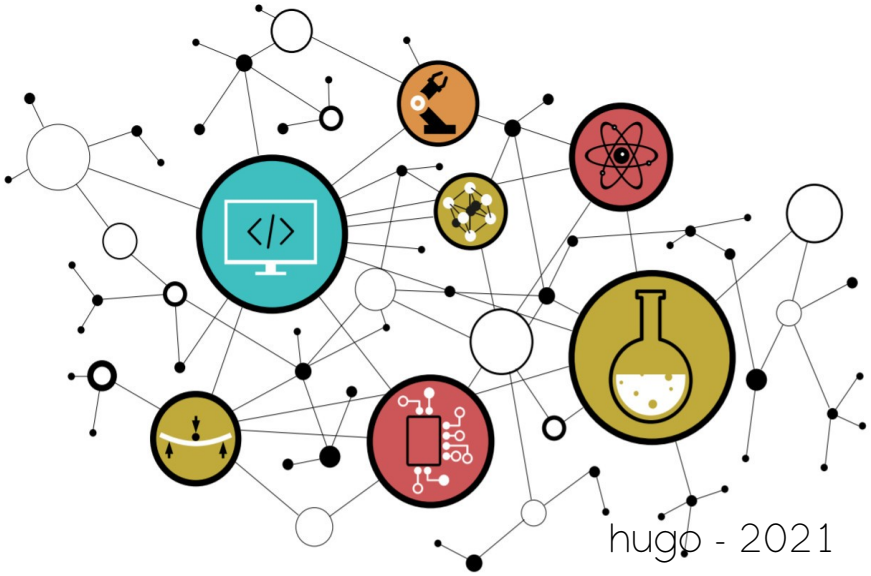
TRAVAUX PRATIQUES

CONCEPTION D'UNE APPLICATION ET ORDONNANCEMENT



hugo - 2021

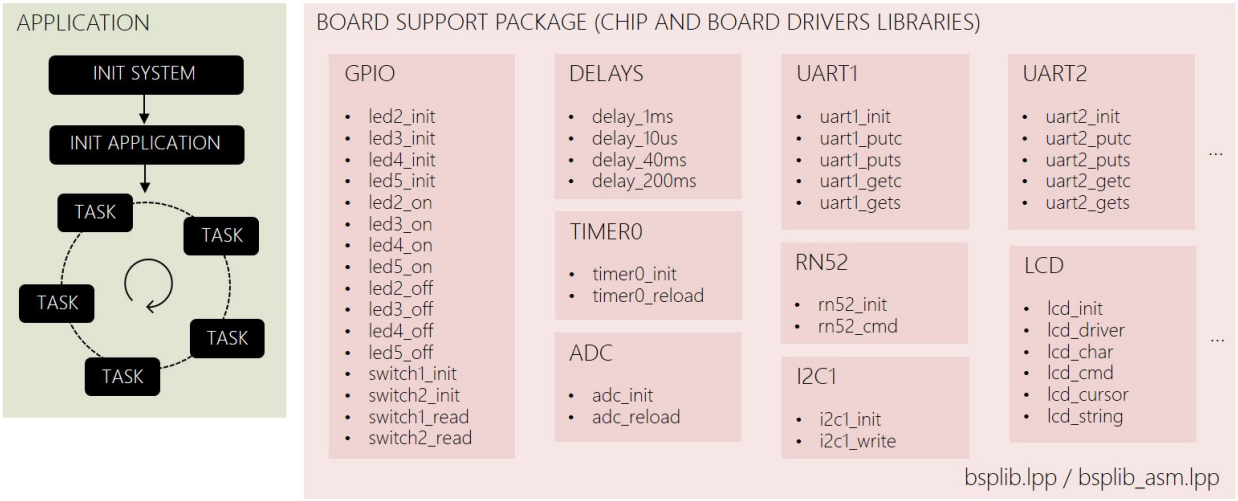
SOMMAIRE

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.

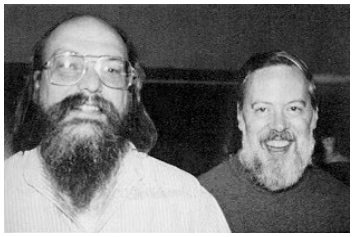
Une application *software* est une solution logicielle de supervision d'un produit répondant à un besoin (souris, clavier, lecteur MP3, assistance au freinage, etc) et a été développée pour une mission spécifique. Cette mission globale sera toujours la même pour un produit donné et peut intégrer des 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*

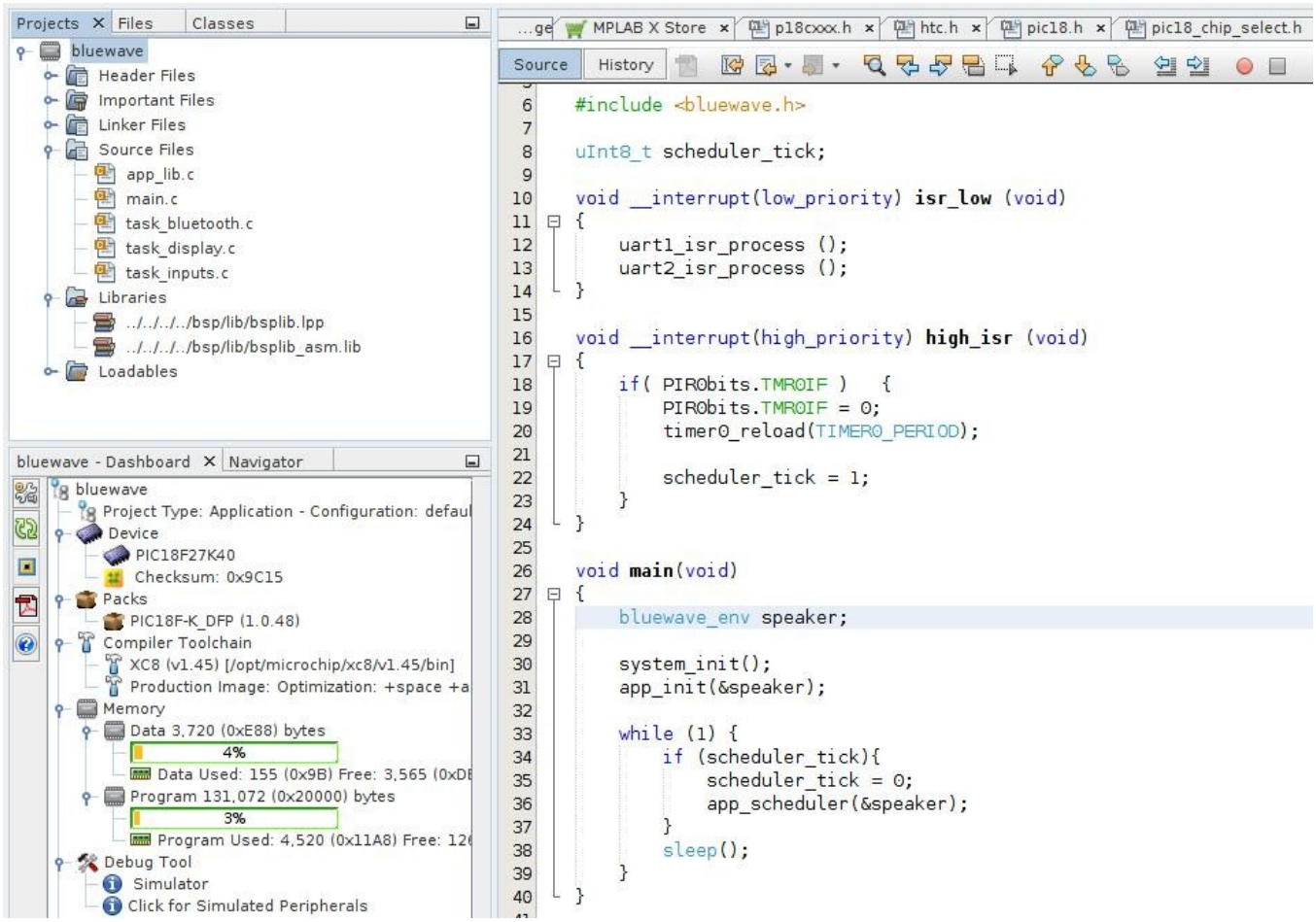
8.2. Conception et ordonnancement de l'application

- 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*

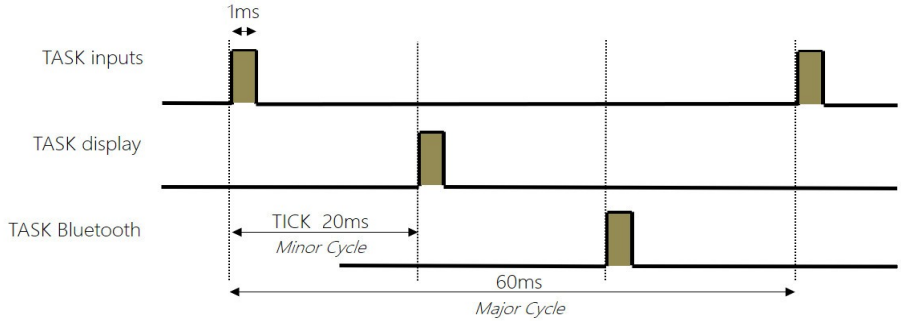
- Créer un projet MPLABX nommé *bluewave* dans le répertoire *disco/apps/bluewave/pjct*. Inclure tous les fichiers sources présents dans le répertoire *bsp/apps/bluewave/src/*. S'assurer de la bonne compilation du projet. S'aider des documents présents dans *mcu/tp/doc/tutos*

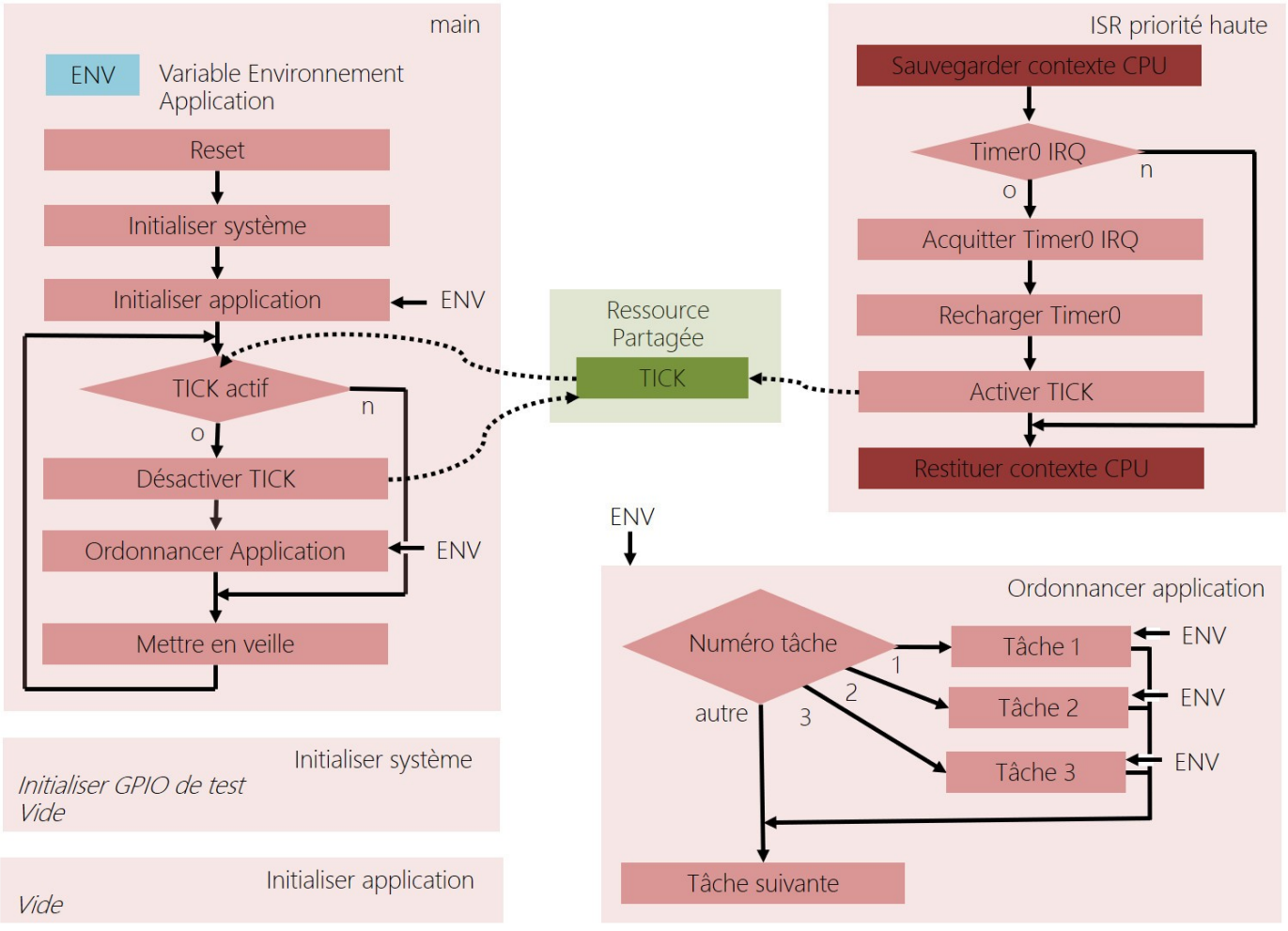
- Ajouter au projet les bibliothèques statiques *bsplib_<your_name>.lpp* et *bsplib_asm_<your_name>.lpp* précédemment générées et validées (retirer la librairie assembleur et inclure les sources ASM au projet si problèmes rencontrés à l'édition des liens – cf. capture d'écran ci-dessous).

  - *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)*



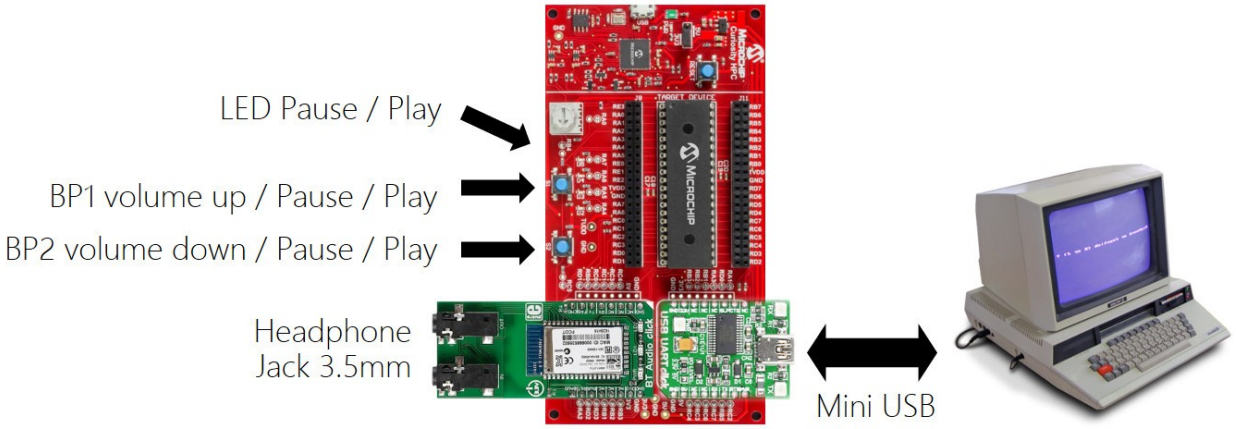
- Développer et valider une application de test implémentant le diagramme de séquence et respectant les chronogrammes suivants (ordonnancement des tâches applicatives). *Dans un premier temps, seule la validation de l'architecture logicielle de l'application nous intéresse (squelette vide). Les fonctions 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.

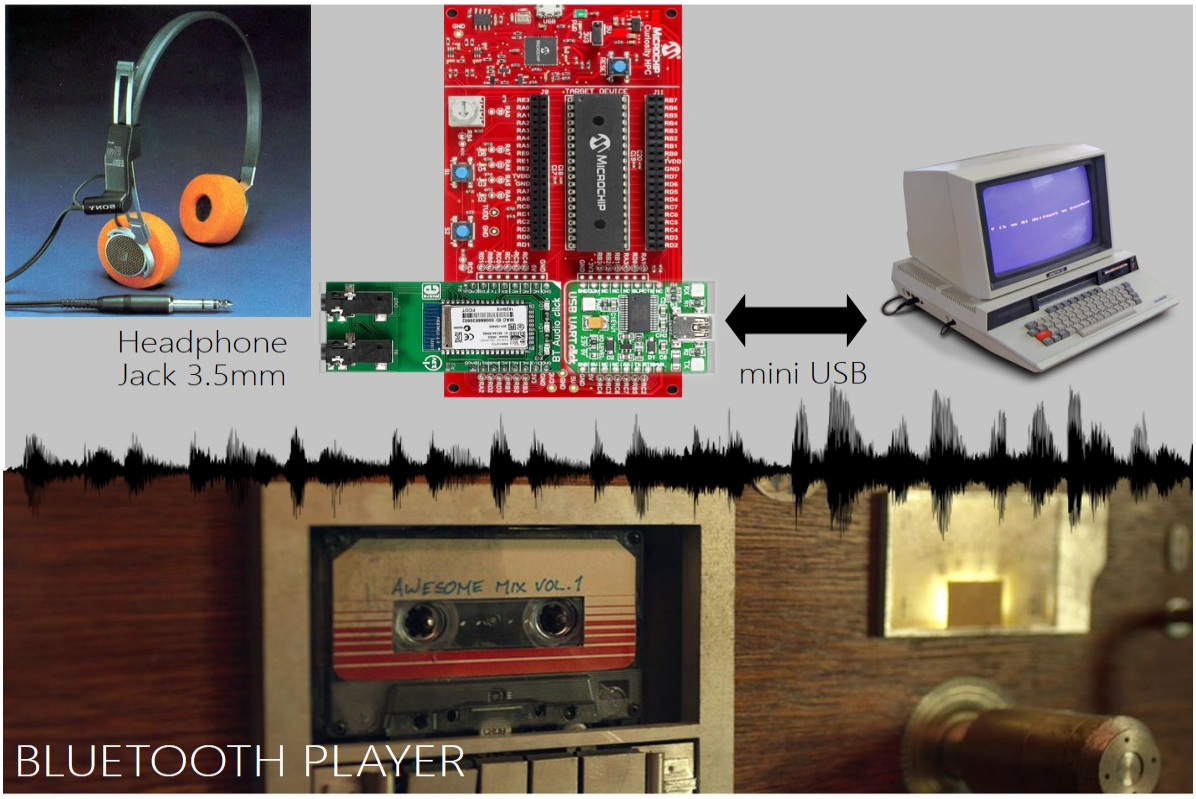


- BP1 (BP2) : Une action seule (sans action sur BP2) permet d'augmenter le volume sur le terminal Bluetooth d'émission (smartphone par exemple). Une action simultanée BP1/BP2 permet de mettre la lecture en pause. Une seconde action simultanée permet de remettre en route la lecture du média audio (Play).
- 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)

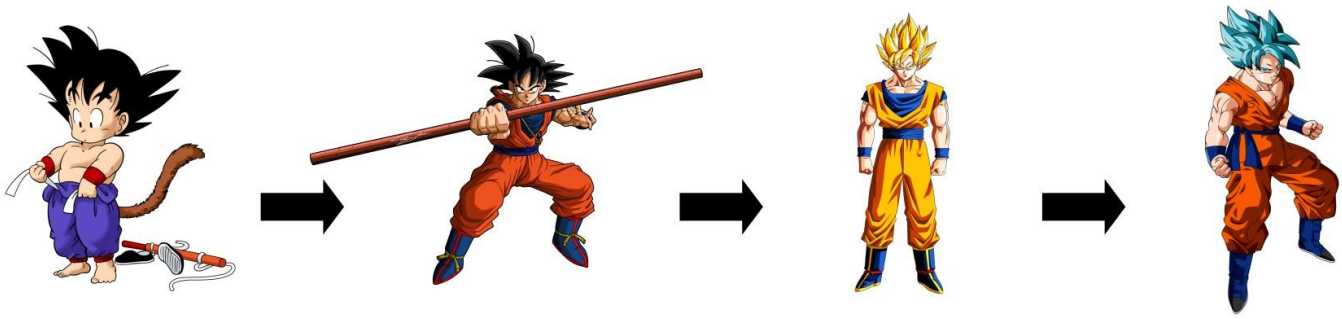
Test, mesure et validation

- 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