**Table des matières**

[Introduction Générale 5](#_Toc690741593)

[Chapitre 1 : Concept général du projet 6](#_Toc1115092296)

[1. Introduction : 6](#_Toc1752775322)

[2. Présentation de l’entreprise d’accueil 6](#_Toc302906575)

[2.1 test 6](#_Toc1158459592)

[3. Présentation du projet 6](#_Toc682779075)

[4. Etude de l’existant 6](#_Toc12258028)

[5. Conclusion 6](#_Toc1024784378)

[Chapitre 2 : exigences et méthodologie de travail 6](#_Toc1389723111)

[1. Introduction 7](#_Toc1656456588)

[2. Norme IEC-62 304 7](#_Toc291511391)

[3. Push-pull-commit (méthodologie de travail) 7](#_Toc967362228)

[4. Conclusion 7](#_Toc1449472437)

[Chapitre 3 : analyse et spécification de besoins 7](#_Toc1018535939)

[1. Introduction 7](#_Toc2425812)

[2. Spécification des besoins 7](#_Toc1875731906)

[3. Architecture globale « Software –Hardware » 7](#_Toc292563015)

[3.1 Hardware : 7](#_Toc898589993)

[3.2 Software : 7](#_Toc1528921005)

[4. Conclusion 7](#_Toc825179907)

[Chapitre 4 : conception et réalisation 7](#_Toc2107614341)

[1. Introduction : 7](#_Toc688162134)

[2. Os KERNEL : 7](#_Toc1505477884)

[2.1 Middleware ( FreeRTOS ) : 8](#_Toc1987159410)

[3. Moteur pas à pas 11](#_Toc1752350198)

[4. Interface homme machine « IHM » 16](#_Toc1260986812)

[5. Capteurs et mesure 16](#_Toc829963518)

[6. Connectivité 16](#_Toc2076020337)

[7. Taches en cours de développement 17](#_Toc677813409)

[8. Conclusion 17](#_Toc73634888)

[Conclusion générale 17](#_Toc411665403)

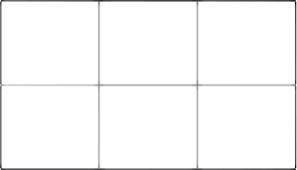
**Tables des figures :**

[Figure 1 2](file:///C:\Users\HP\Downloads\majdi-table.docx#_Toc101189233)

[Figure 2 856jdfjjfj5 6](file:///C:\Users\HP\Downloads\majdi-table.docx#_Toc101189234)

[Figure 3 : azizos 6](file:///C:\Users\HP\Downloads\majdi-table.docx#_Toc101189235)

[Figure 4 56565 6](file:///C:\Users\HP\Downloads\majdi-table.docx#_Toc101189236)



Figure

Liste des tableaux :

[Tableau 1 hhdhhdhh 7](#_Toc101189347)

# **Introduction Générale**

# **Chapitre 1 : Concept général du projet**

## **Introduction :**

## **Présentation de l’entreprise d’accueil**

### **2.1 test**

[place your text here]

## **Présentation du projet**

[place your text here]

## **Etude de l’existant**

[place your text here ]

Tableau 1 hhdhhdhh

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
|  |  |  |

## **Conclusion**

[place your text here]

# **Chapitre 2 : exigences et méthodologie de travail**

## **Introduction**

[place your text here ]

## **Norme IEC-62 304**

[place your text here]

## **Push-pull-commit (méthodologie de travail)**

[place your text here ]

## **Conclusion**

[place your text here ]

# **Chapitre 3 : analyse et spécification de besoins**

## **Introduction**

[place your text here]

## **Spécification des besoins**

[place your text here ]

## **Architecture globale « Software –Hardware »**

[place your text here]

### **Hardware :**

### **Software :**

## **Conclusion**

[place your text here]

# **Chapitre 4 : conception et réalisation**

## **Introduction :**

## **Os KERNEL :**

### **Middleware ( FreeRTOS ) :**

L’un des exigences majeures pour les dispositifs médicaux est la stabilité du système **(soft + le soft externe + le hardware autour)**.

Notre Solution était alors de travailler dans le domaine temps réels, où le taux d’erreur est minimal et la stabilité est maximal.

##### **RTOS :**

Le système d’exploitation temps réel est un OS qui gère plusieurs tâches concurrentes selon leurs degrés de priorités, il est utilisé quand il y a des exigences temporelles sur les processus. Ce type d’ordonnancement, appelé ordonnancement préemptif.

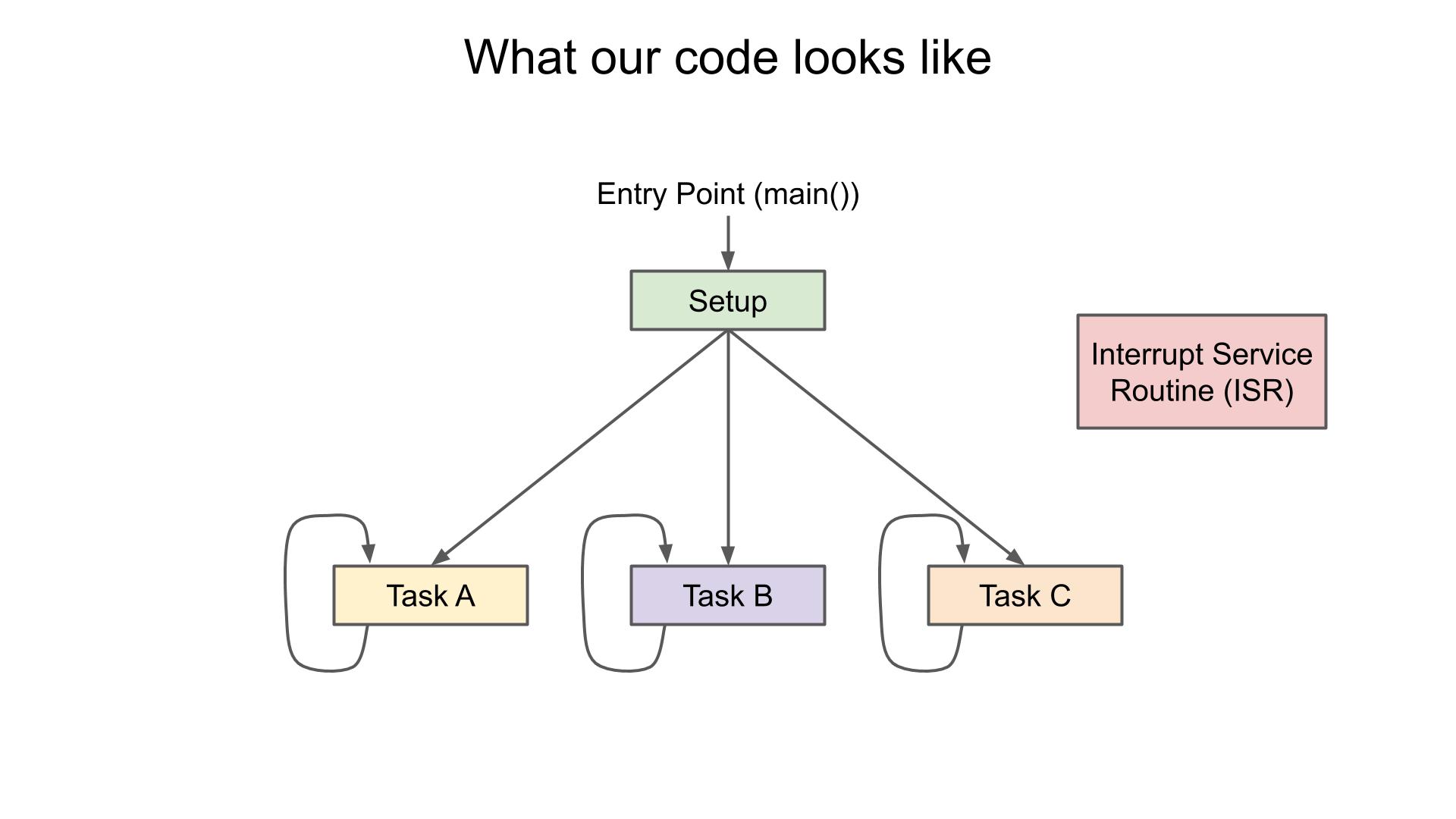
RTOS garantis la performance maximale du processeur et la bonne gestion de la mémoire ainsi que le fonctionnement sans erreur (Error-Free) offert par ces types de systèmes.

##### **FreeRTOS/CMSISV2**

Dans le domaine de l’embarqué les ressources sont relativement limitées en termes de mémoire et de traitements. Dans ce projet on utilise un microcontrôleur Stm32H7 **(Arm Cortex-M7)** qui est de la catégorie haute performance, mais on reste toujours limités de ressources.

C’est pour cela qu’on a choisis **FreeRTOS**, c’est un système d’exploitation embarqué multitâches temps réel préemptif supporte actuellement 35 architectures. Il est aujourd’hui parmi les plus utilisés dans le marché des systèmes d'exploitation temps réel pour l’embarquée grâce à sa faible taille qui est de l’ordre de 4000 à 9000 octets.

On a utilisé le **CMSIS-RTOS API v2** comme une couche d’abstraction à FreeRTOS afin de garantir un système optimisé et améliorer la portabilité du code entre les différents processeurs ARM.



* 1. **Tâches Et Queues (File d’attente) :**

##### **Création des Taches :**

Pour développer une application basée sur un OS, on décompose l’application en un ensemble de tâches. Dans FreeRTOS une tâche est fonction C contenant une boucle infinie et ne renvoie pas un résultat.

**void vATaskFunction( void \*pvParameters )**

**{**

**for( ; ; )**

**{**

**}**

**}**

Une tâche est créée par l'intermédiaire de la fonction “osThreadNew” qui retourne l’id de la tache RTOS

**osThreadNew**(**osThreadFunc\_t** func,**void \*** argument,**const osThreadAttr\_t \*** attr)

[**in**] **fun** thread function.

[**in**] **argument** pointer that is passed to the thread function as start argument.

[**in**] **attr** thread attributes **(ici les priorités sont spécifiées (figure ci-dessous)).**



Ci-dessous sont les taches utilisées dans le projet classé selon leurs priorités d’exécution.

* **Tache 1 : stepperHandle**

Cette tache dispose la priorité maximale osPriorityHigh car elle gère le moteur pas à pas qui est le cœur du projet et tous les autres services fonctionnent en fonction de son état et de son avancement.

* **Tache 2 : IHMHandle**

Elle dispose comme priorité osPriorityAboveNormal, elle gère les flux de données entre l’interface homme machine et les autres taches.

* **Tache 3 : ConnectivityHandle :**

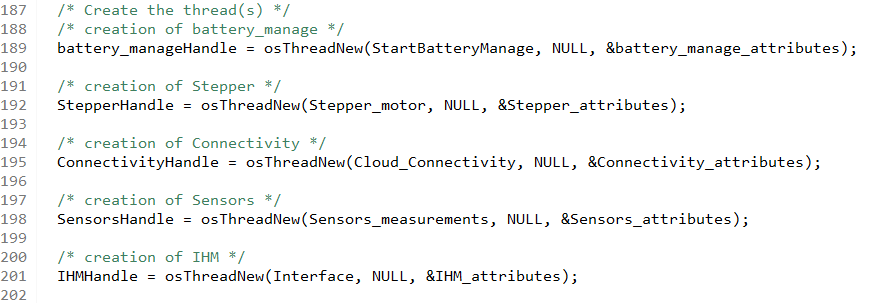
La connectivité admet le même degré de priorité que la tache IHMHandle osPriorityAboveNormal, toutes les données nécessaires sont envoyées à travers cette tache vers le Cloud afin de les récupérer en temps réels par une application de supervision.

* **Tache 4 : SensorsHandle**

Cette tâche représente l’unité de traitements de tous les capteurs utilisés dans ce projet elle dispose comme priorité osPriorityNormal1.

* **Tache 5 : Battery\_manageHandle**

La gestion de batterie est assurée par cette tâche avec une priorité osPriorityNormal. (Elle est en cours de développement)



Une tâche FreeRTOS peut se trouver dans l’un des états suivants :

**Prête (Ready)** : une tâche qui possède toutes les ressources nécessaires à son exécution. Elle lui manque seulement le processeur.

**Active (Running)** : Tâche en cours d’exécution, elle est actuellement en possession du processeur.

**Attente (Blocked)** : Tâche en attente d’un événement (queue de messages, sémaphores, timeout ...). Une fois l'événement arrivé, la tâche concernée repasse alors à l'état prêt.

**Suspendu (Suspended)** : tâche à l’état dormant, elle ne fait pas partie de l’ensemble des tâches ordonnançables**.**

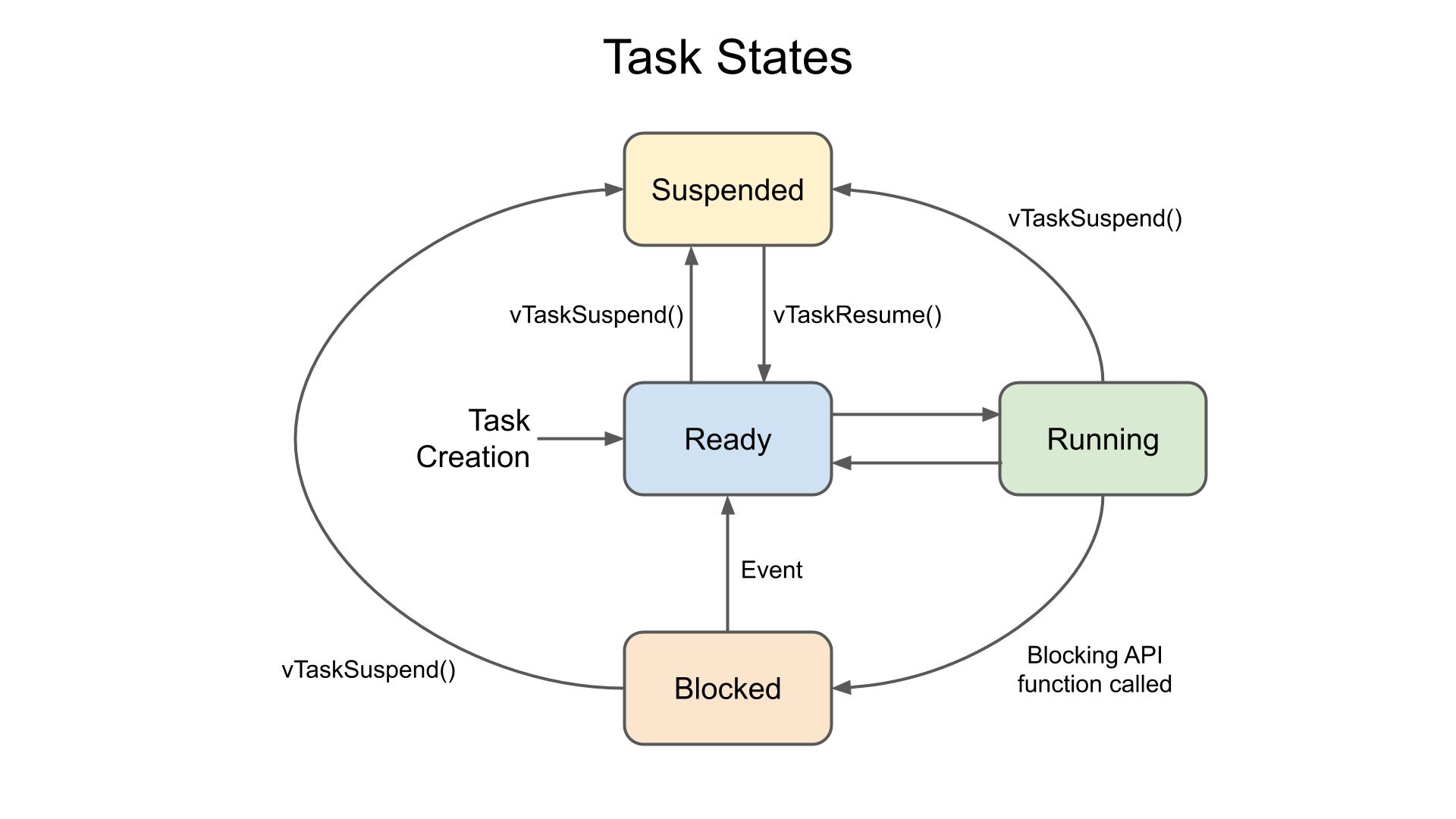
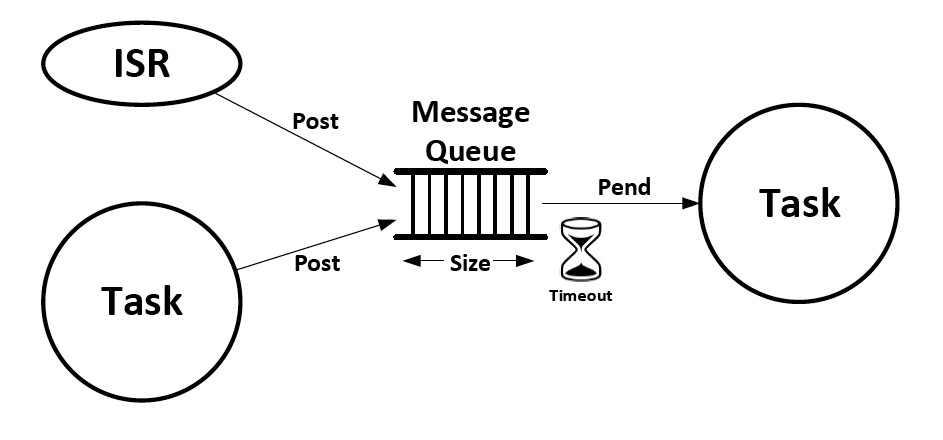


Figure 5: état des taches

* **Queues (Files d’attente) :**

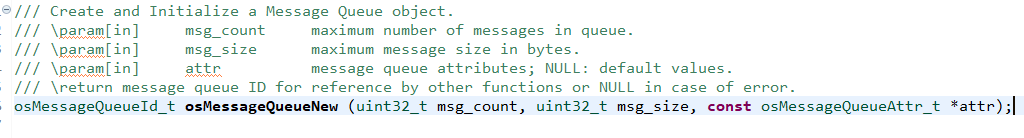
Avant de parler des queues il faut parler des problèmes majeurs lors de l’utilisation d’un système temps réels et surtout s’il s’agit d’un système préemptif. Les variables globales ne sont plus une solution optimale pour stocker l’information à cause de la concurrence des taches. C’est très probable alors que deux taches écrivent en même temps dans une variable x, les données ne sont plus utilisables dans ce cas. Beaucoup d’autres problèmes sont rencontrés lors de l’utilisation des variables globales.

Les queues sont utilisées pour résoudre ces problèmes avec une opération atomique càd une écriture ou lecture dans la queue ne peut pas être interrompue avant la fin de l’opération. Il s’agit d’un système FIFO (first input first output). Les files d’attentes sont utilisées dans ce projet dans tous les flux de communications inter tâches.



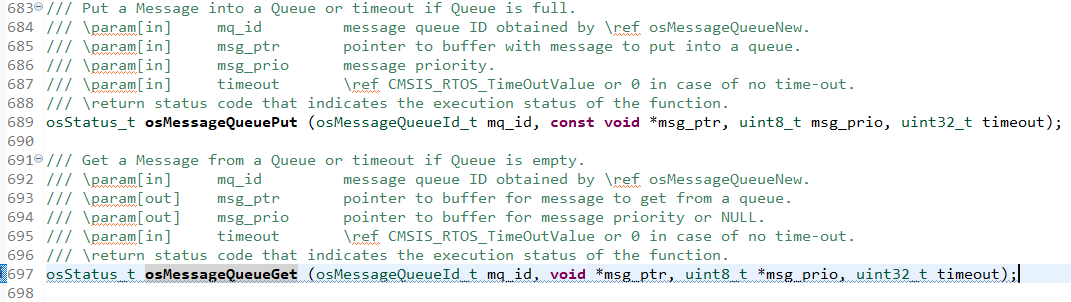
La bibliothèque “Cmsis\_Os2” dispose les fonctions necessaires pour créer, Lire et écrire dans les files d’attente (queues).

Pour créer une queue on utilise la fonction “osMessageQueueNew”, elle retourne en résultat l’id pour la file créée de type “osMessageQueueId\_t”.



Pour déposer un message dans la file d’attente on utilise la fonction “osMessageQueuePut”, elle place le message pointé par “msg\_ptr” dans la file d'attente spécifiée par le paramètre “mq\_id”. Le paramètre “msg\_prio” est utilisé pour trier les messages en fonction de leur priorité (les chiffres les plus élevés indiquent une plus grande priorité) lors de l'insertion.

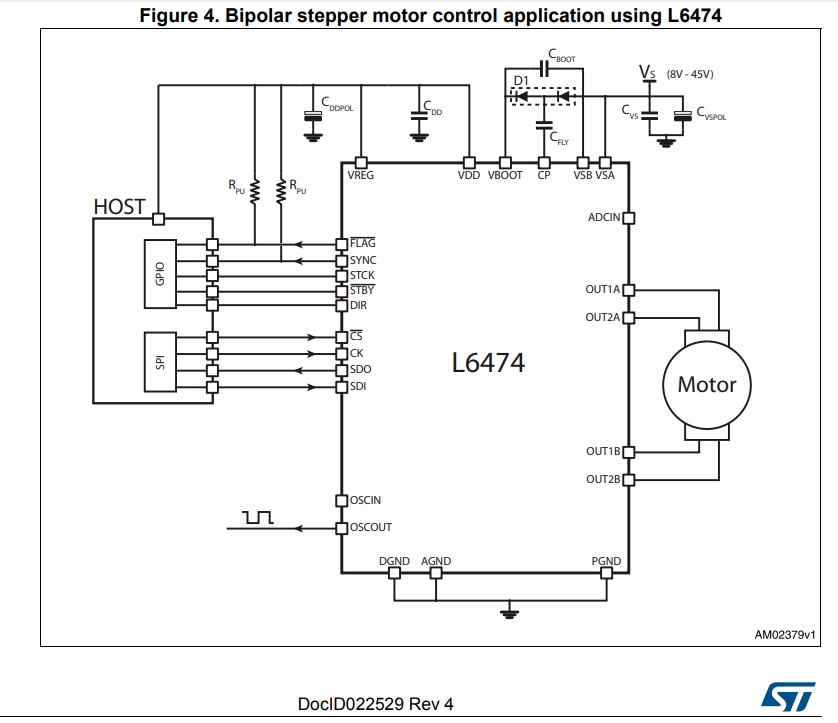
La fonction “osMessageQueueGet” lit le contenu de la file d’attente (queue) dont l’id est passé en paramètre, si la queue est vide (pas de messages) et elle a dépassé le délai maximal d’attente “timeout”, la fonction retourne “osErrorTimeout” si non elle retourne “osOK”.



## **Moteur pas à pas**

* 1. **L6474 Driver / L6474.C:**

Afin d'utiliser un moteur pas à pas il est nécessaire d'utiliser un "driver". Ces drivers permettent de transmettre la puissance électrique au moteur afin de le faire tourner selon nos besoins.

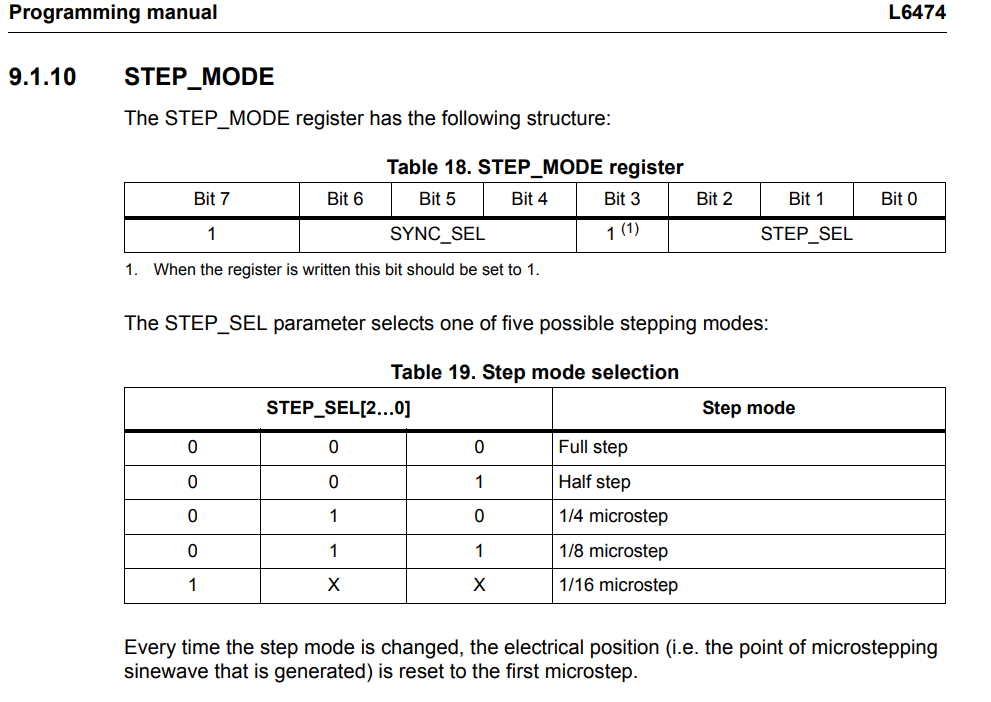


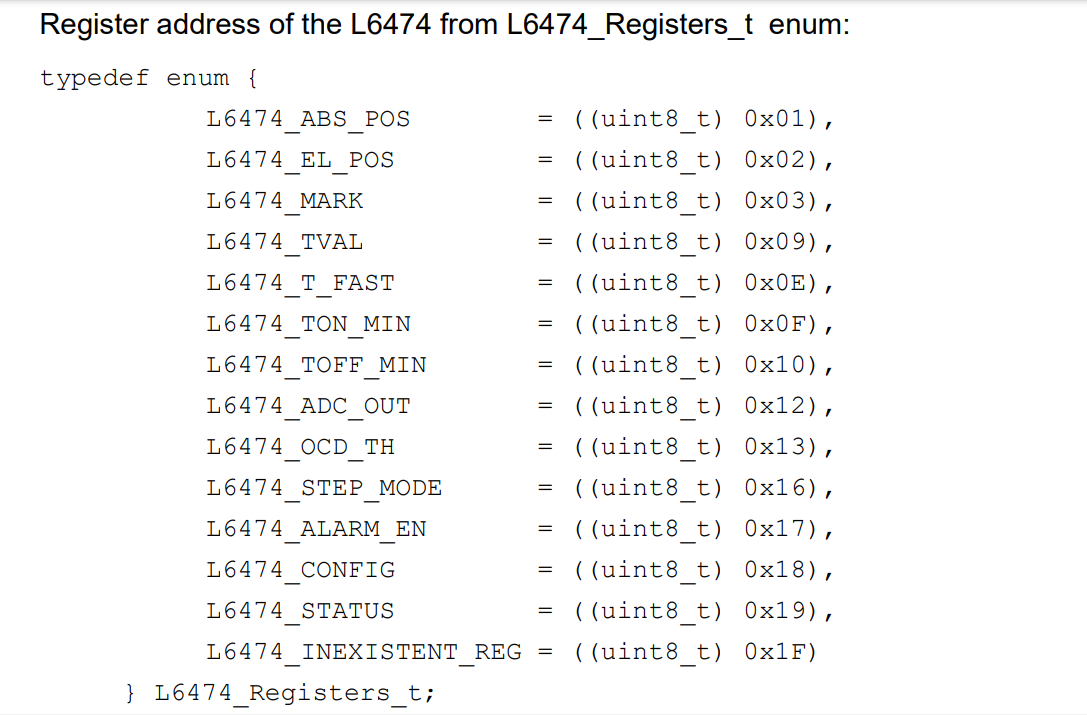
Nous travaillons avec la carte d’expansion **x-nucleo-ihm01a1** basée sur le L6474 driver et le moteur pas à pas **Nema 17**.

La communication entre notre carte et le driver est à travers le protocole **SPI** 8bit (Serial Peripheral Interface) où Le microcontrôleur représente le Master or que le driver est l’esclave.

“Une liaison SPI (pour Serial Peripheral Interface) est un bus de données série synchrone baptisé ainsi par Motorola, au milieu des années 1980 1qui opère en mode full-duplex. Les circuits communiquent selon un schéma maître-esclave, où le maître contrôle la communication. Plusieurs esclaves peuvent coexister sur un même bus, dans ce cas, la sélection du destinataire se fait par une ligne dédiée entre le maître et l'esclave appelée « Slave Select (SS).”, **Wikipédia**

Le L6474 driver admet plusieurs registres qui sont responsable à convertir la commande SPI en une commande analogique du moteur pas à pas. Chaque registre admet une adresse bien déterminée, par exemple le registre **STEP\_MODE** admet l’adresse **0X16** est responsable à changer le mode du micro-pas (**Microstepping**).



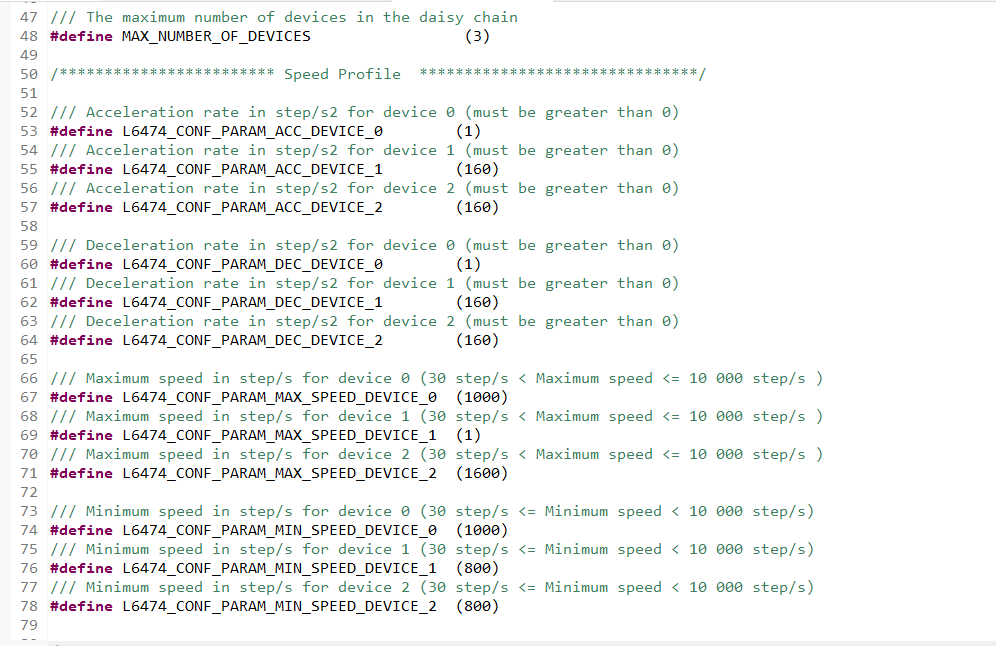


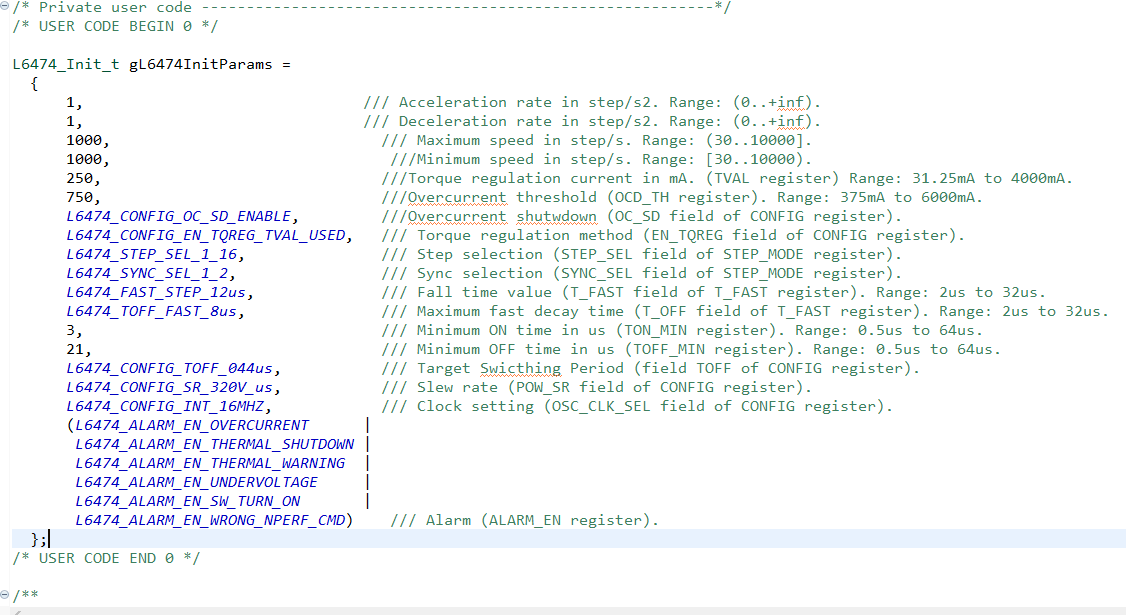
La bibliothèque fournit par ST “**X-CUBE-SPN1**“ peut gérer tous les commandes bas niveau à travers des fonctions prédéfinit qui envoient des trames bien déterminés contenant l’adresse du registre ainsi que le code commande correspondant et des arguments si nécessaire, mais le problème que cette bibliothèque est compatible qu’avec les Nucléo F4, F3, F0, L0.

Dans ce cas, la première étape était d’adapté les fichiers “.h” (header files) avec notre carte (OpenH7). Les timers et leurs channels, les brochages des pins SPI (MISO – MOSI – CLK – NSS(CS)), le pin de flags ainsi que de la remise à zéro, tous été modifiés …

Pour l’initialisation du driver on pourrait choisir entre utiliser le fichier des valeurs par défaut des registres “**l6474\_target\_config.h**”, ou bien de déclarer une variable de type Structure C “**L6474\_Init\_t**” où on spécifie tous les paramètres à initialiser comme la vitesse du moteur maximale et minimale (pas/s), l’accélération et la décélération (pas/s^2), les paramètres relatifs au courant, les alarmes...

Dans notre cas nous avons modifié le fichier des paramètres par défaut selon nos besoins.



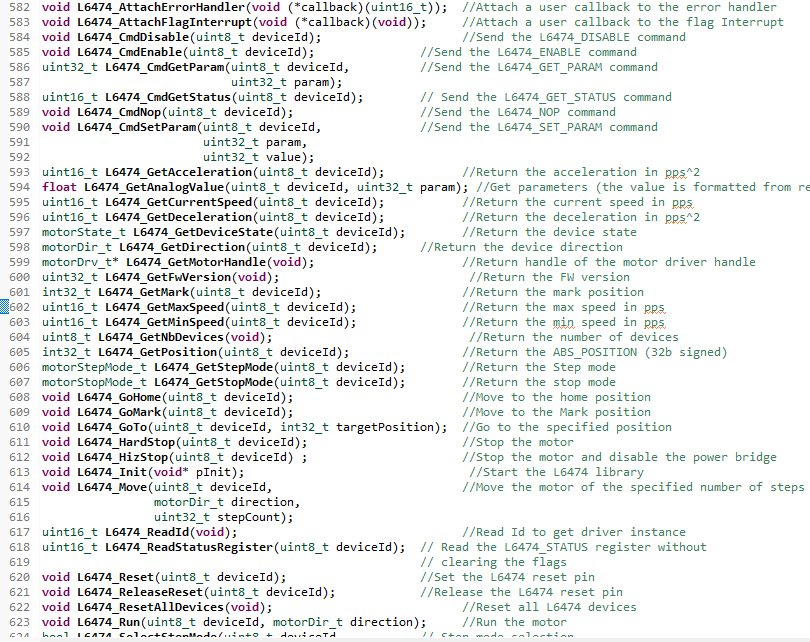


Pour avoir une vitesse constante toute au long de l’opération de l’injection, l’accélération et la décélération sont initialisés à 1 (sans accélération / décélération).

La vitesse maximale et minimale va être modifier dans le code selon le débit d’injection à l‘aide des fonctions

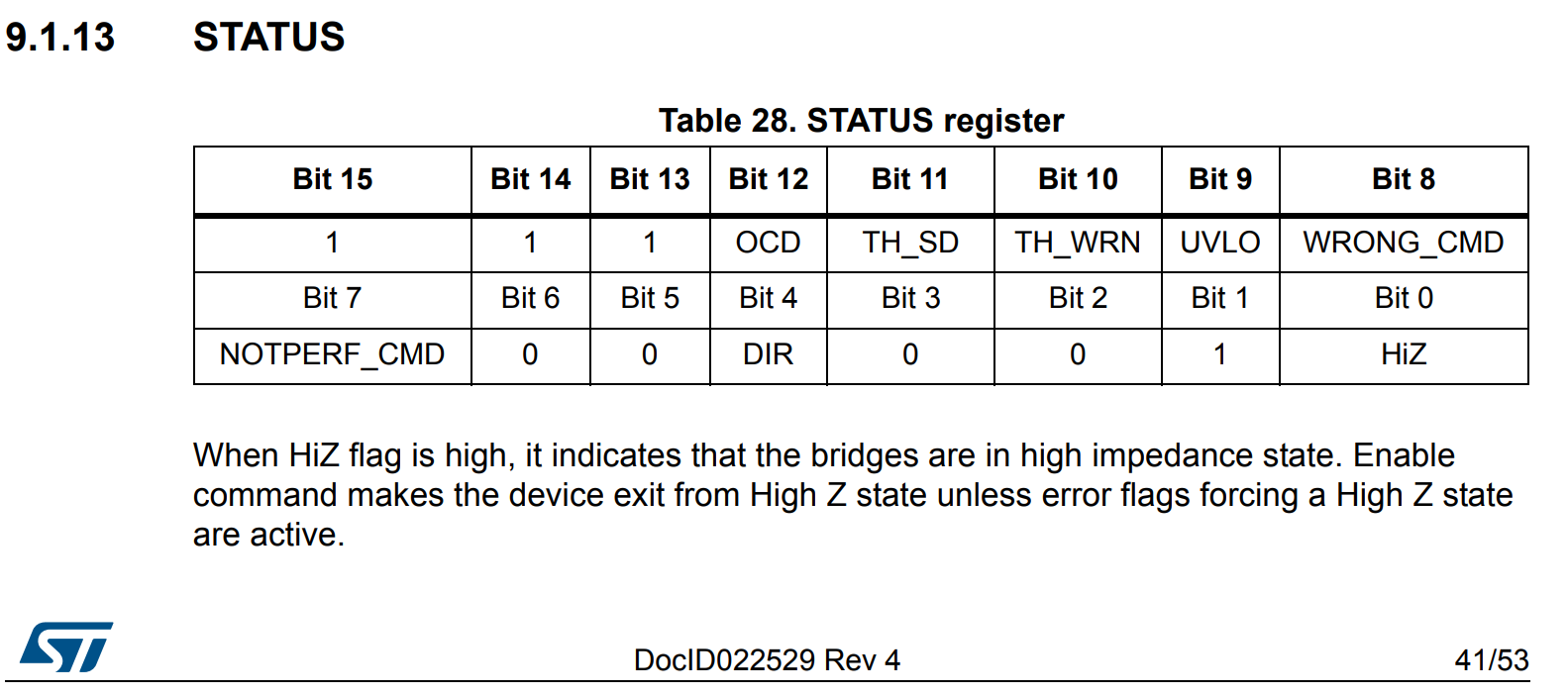
**“uint16\_t L6474\_SetMaxSpeed(uint8\_t deviceId, uint16\_t newMaxSpeed)”**

**“uint16\_t L6474\_SetMinSpeed(uint8\_t deviceId, uint16\_t newMinSpeed)”**

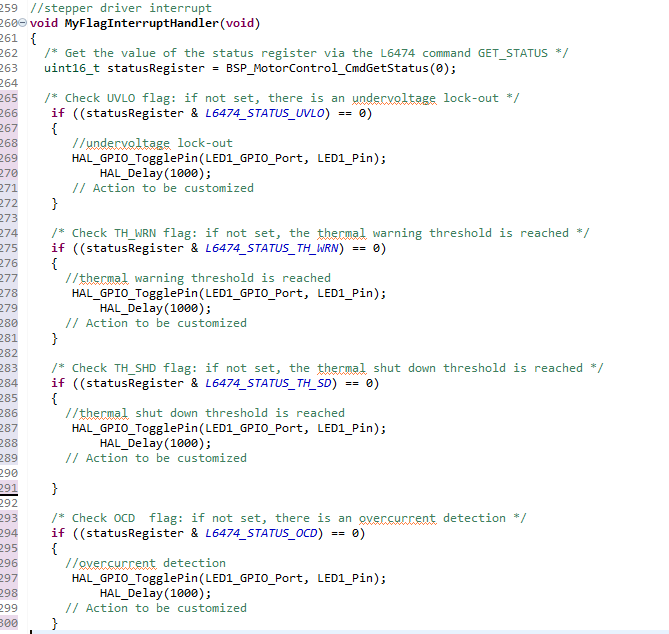


* 1. **Interruptions :**

Le L6474 contient Un ensemble très riche de protections (thermique, faible tension de bus, surintensité …) On peut détecter ces irrégularités à travers le registre STATUS qui contient des flags indiquant l’état du driver.



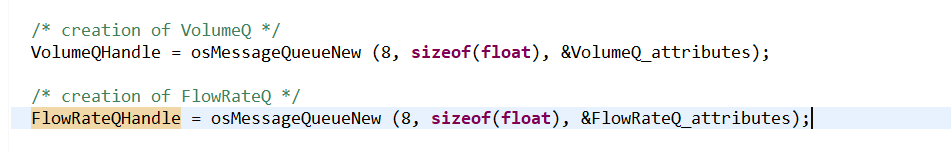
Lors de l’initialisation on définit une limite pour chaque grandeur, si elle est dépassée une interruption est lancée dans notre programme, elle dispose toujours la priorité maximale. En cas d’interruption, une alarme est activée en fonction de l’état du driver. Pour le moment les alarmes s’agissent des toggles Led et une notification dans l’application mobile. Nous allons les modifiés aux furs et à mesure avec des alarmes sonores selon les exigences générales des systèmes d’alarmes **(EN 60601-1-8).**



* 1. **Flux de données :**

On trouvera un échange de données entre la tâche **StepperHandle** et les autres taches, cet échange est assuré par les queues (files d’attentes).

On reçoit le débit d’injection à l’aide de la file “FlowRateQHandle”, cette queue peut contenir 8 message maximum de type “float”.



Le volume à injecter et le rayon de la seringue sont placés respectivement dans “VolumeQHandle” et “RadiusQHandle”. Selon ces 3 paramètre la vitesse du moteur est calculée comme il est indiqué dans la partie 3.4.

Il y on a évidemment un flux de données sortant de la tache **StepperHandle** vers les tâches qui ont besoin d’informations concernant le moteur. Par exemple, après calculer le temps total d’injection, on dépose cette information dans une queue “TotalTimeQHandle”. Ainsi, on partage le nombre total des pas dans la queue “LastStepQHandle”.

* 1. **Calcule :**

Afin de contrôler le débit d’injection à l’aide d’un moteur pas à pas et un système vis écrou, il faut utiliser la mécanique de fluide.

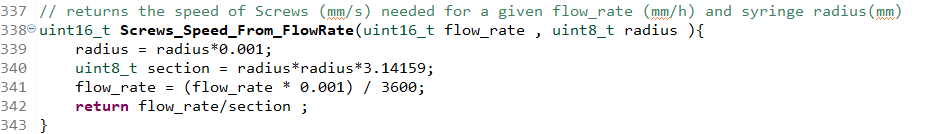
On commencera avec le terme “**débit volumique**” qui désigne la quantité de liquide qui circule dans une canalisation durant un laps de temps déterminé. Exprimé en litres par seconde (L/s), litres par minute (L/mn) ou en mètres-cubes par heure (m3/h). Dans notre cas, le liquide c’est le médicament à injecter et le canal c’est la seringue.

Étant donné que la **viscosité** des médicaments est de même ordre que celle de l’eau, elle influence peu sur le calcul, donc elle va être négligée.

Pour une section d’un canal donnée, plus la vitesse de passage est grande, plus le débit d'écoulement sera important :

***v* =** **qv /** **S**  
**Avec :**  
**q***v : débit volumique en [m³/s]*  
***v*** *: vitesse du fluide en [m/s]*  
***S*** *: section de passage en [m²*]

Pour calculer la section (π\*r2) on obtient le rayon de la seringue à travers la tâche “SensorsHandle”. Le débit est tapé par le médecin dans l’écran TFT. La vitesse du fluide ou la vitesse de déplacement est calculée d’après la formule précédente.



L’écrou fait le déplacement d’un pas quand le moteur pas à pas termine une tour complète (une révolution). Donc la vitesse du moteur (en tours/seconde) est :

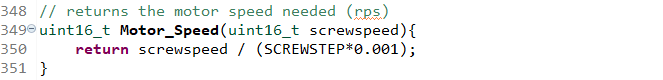
***N=v/p***

***Avec :***

***N :*** *nombre de tours par seconde(tr/s)*

***V :*** *vitesse de déplacement nécessaire(m/s)*

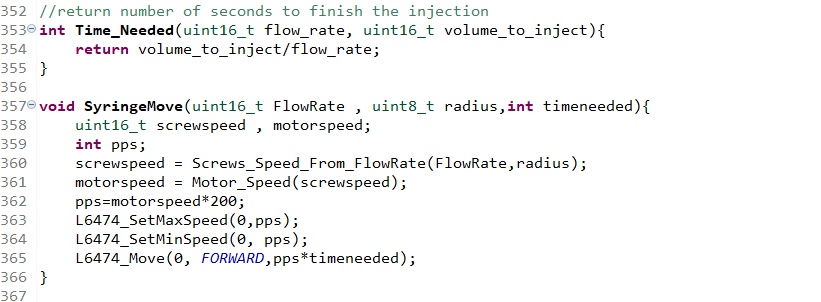
***P :*** *le pas de la vis(m)*



Une révolution représente 360°, or qu’un pas complet du moteur (Full Step) représente 1.8° donc le nombre des pas totale pour une seule révolution est **360°/1.8°=200 pas/révolution.**

La vitesse du moteur dans la bibliothèque est en **pas/seconde** donc la fonction L6474\_SetMaxSpeed prend en paramètre **N\*200.**

Le temps total de l’opération d’injection est le rapport **volume/débit.**



## **Interface homme machine « IHM »**

[place your text here ]

## **Capteurs et mesure**

[place your text here ]

## **Connectivité**

[place your text here ]

## **Taches en cours de développement**

[place your text here]

## **Conclusion**

[place your ext here ]

# **Conclusion générale**

[place your text here ]