Table des matières

[Introduction Générale 1](#_Toc1964498434)

[Chapitre 1 1](#_Toc1060978371)

[Introduction : 1](#_Toc1964668764)

[Présentation de l’entreprise d’accueil 1](#_Toc744882486)

[2.1 test 1](#_Toc1076150849)

[Présentation du projet 1](#_Toc513835583)

[Etude de l’existant 1](#_Toc1123696696)

[Conclusion 2](#_Toc2001033089)

[Chapitre 2 2](#_Toc869756490)

[Introduction 2](#_Toc1105979546)

[Norme IEC-62 304 2](#_Toc1562948656)

[Push-pull-commit (méthodologie de travail) 3](#_Toc1993713534)

[Conclusion 3](#_Toc332793042)

[Chapitre 3 Analyse et spécification de besoins 3](#_Toc970714243)

[1- Introduction 3](#_Toc1520202734)

[2- Spécification des besoins 3](#_Toc1483727872)

[3- Architecture globale « Software –Hardware » 4](#_Toc1320693849)

[3-1 Architecture Hardware : 4](#_Toc494163973)

[3.2 Architecture Software : 4](#_Toc1854329444)

[Chapitre 4 : Conception et réalisation 4](#_Toc1005823397)

[1- Introduction 4](#_Toc1556021012)

[2- OS kernel 4](#_Toc1132789228)

[2-1 Middleware (FreeRTOS) 4](#_Toc1944545520)

[2-2 Tâches Et Queues (File d’attente) 4](#_Toc728590515)

[3- Moteur pas à pas 4](#_Toc814107027)

[3-1 L6474 Driver / L6474.C : 4](#_Toc1194025915)

[3-2 Interruptions 4](#_Toc2067270120)

[3-3 Flux de données : 4](#_Toc1721588702)

[3-4 Calcule 4](#_Toc541533001)

[4- Interface homme machine « IHM » 4](#_Toc913976442)

[Capteurs et mesures 4](#_Toc2031364990)

[Connectivité 4](#_Toc2121534542)

[Taches en cours de développement 4](#_Toc1981133229)

[Conclusion 4](#_Toc1571193278)

Liste des figures

[Figure 1 : Architecture "Pousse seringue" 9](#_Toc103110483)

[Figure 2 : Carte STM32H7 9](file:///C:\Users\ASUS-MSi\Desktop\aziz%20majdi\rapppfe%20aziz_maj.docx#_Toc103110484)

[Figure 3: Moteur PAS à PAS (NEMAI17) 10](file:///C:\Users\ASUS-MSi\Desktop\aziz%20majdi\rapppfe%20aziz_maj.docx#_Toc103110485)

[Figure 4 : DRIVER L6474 10](file:///C:\Users\ASUS-MSi\Desktop\aziz%20majdi\rapppfe%20aziz_maj.docx#_Toc103110486)

[Figure 5 : LCD TFT 4.3 10](file:///C:\Users\ASUS-MSi\Desktop\aziz%20majdi\rapppfe%20aziz_maj.docx#_Toc103110487)

[Figure 6 : Node MCU ESP8266 11](file:///C:\Users\ASUS-MSi\Desktop\aziz%20majdi\rapppfe%20aziz_maj.docx#_Toc103110488)

[Figure 7 : Capteur de position (3590S-2-103L) 11](file:///C:\Users\ASUS-MSi\Desktop\aziz%20majdi\rapppfe%20aziz_maj.docx#_Toc103110489)

[Figure 8 : Capteur de diamètre (PTL01-15W0-103B1) 11](file:///C:\Users\ASUS-MSi\Desktop\aziz%20majdi\rapppfe%20aziz_maj.docx#_Toc103110490)

[Figure 9 : Capteur de proximité (OPB745WZ) 11](file:///C:\Users\ASUS-MSi\Desktop\aziz%20majdi\rapppfe%20aziz_maj.docx#_Toc103110491)

[Figure 10 : Capteur jauge de contrainte (FSS020WNST) 12](file:///C:\Users\ASUS-MSi\Desktop\aziz%20majdi\rapppfe%20aziz_maj.docx#_Toc103110492)

[Figure 11 : Adaptation Analogique(AD620) 12](file:///C:\Users\ASUS-MSi\Desktop\aziz%20majdi\rapppfe%20aziz_maj.docx#_Toc103110493)

[Figure 12: Module alimentation (KMF0.2113.11) 12](file:///C:\Users\ASUS-MSi\Desktop\aziz%20majdi\rapppfe%20aziz_maj.docx#_Toc103110494)

[Figure 13 : Batterie (RRC2054) 12](file:///C:\Users\ASUS-MSi\Desktop\aziz%20majdi\rapppfe%20aziz_maj.docx#_Toc103110495)

[Figure 14: Buzzer (458-1402-ND) 13](file:///C:\Users\ASUS-MSi\Desktop\aziz%20majdi\rapppfe%20aziz_maj.docx#_Toc103110496)

[Figure 15 : Memory Card Connector ( SD-RSMT-2-MQ-WF ) 13](file:///C:\Users\ASUS-MSi\Desktop\aziz%20majdi\rapppfe%20aziz_maj.docx#_Toc103110497)

[Figure 16 : état des taches 19](file:///C:\Users\ASUS-MSi\Desktop\aziz%20majdi\rapppfe%20aziz_maj.docx#_Toc103110498)

[Figure 17 : l'utilisation de Queuse RTOS 20](file:///C:\Users\ASUS-MSi\Desktop\aziz%20majdi\rapppfe%20aziz_maj.docx#_Toc103110499)

[Figure 18 : Modèle de conception modèle-vue-présentateur 29](file:///C:\Users\ASUS-MSi\Desktop\aziz%20majdi\rapppfe%20aziz_maj.docx#_Toc103110500)

[Figure 19 : Modèle-Vue-Présentateur et communication externe 30](file:///C:\Users\ASUS-MSi\Desktop\aziz%20majdi\rapppfe%20aziz_maj.docx#_Toc103110501)

Liste des tableaux

[Tableau 1 : liste des composants 7](#_Toc103091289)

# Introduction Générale

# Chapitre 1

## **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

## **Introduction**

[place your text here ]

## **Norme IEC-62 304**

La norme internationale IEC 62304 – logiciels de dispositifs médicaux – processus du cycle de vie des logiciels est une norme qui spécifie les exigences du cycle de vie pour le développement de logiciels médicaux et de logiciels au sein des dispositifs médicaux. Il est harmonisé par l'Union européenne et les États-Unis et peut donc être utilisé comme référence pour se conformer aux exigences réglementaires de ces deux marchés. La norme est composée d’une exigence générale, et de 5 processus, dont seul le processus de développement du logiciel nous concerne [10].

#### **Exigence générale**

L’exigence contient sert à identifier le système de management de qualité, le système de gestion de risque a utilisé, et de classifier le niveau de sécurité du logiciel.

Les deux premiers, intéressent l’ingénieur qualité plus que l’ingénieur développement, mais la troisième étape est nécessaire pour avancer dans le développement vu que le niveau de sécurité attribué affecte directement le processus de développement. Ci-dessous les niveaux de sécurité possible et leur conséquence au cas d’une défaillance du système :

* Classe A : Aucune blessure ou atteinte à la santé n'est possible.
* Classe B : Une BLESSURE NON GRAVE est possible.
* Classe C : La mort ou une BLESSURE GRAVE est possible.

La pousse seringue électrique à un niveau de sécurité variable selon l’utilisation. Par exemple d’un département pédiatrie ou l’utilisation fréquente est l’infusion des compléments alimentaire la classification sera B, par contre dans une chambre d’urgence, ou soins intensifs, ou la plupart des médicaments utilisé sont dangereux en haute concentration. Une erreur de dosage peut provoquer la mort du patient, la pousse seringue est classifié C.

Vue que notre, produit vise toute utilisation possible de la pousse seringue nous avons attribué le niveau de sécurité C au processus de développement logiciel.

#### **Processus de développement logiciel**

Le processus de développement du logiciel représente des étapes à suivre pour qu’elle soit conforme à la norme 62304. Les étapes à suivre sont régit par le niveau de sécurité attribué. Le tableau 4 décrit les étapes à suivre selon le niveau sécurité [11].

*Tableau 4 : etape du precessus de developpement logiciel*

|  |  |  |  |
| --- | --- | --- | --- |
| Etape | Classe A | Classe B | Classe C |
| Planification du développement logiciel | X | X | X |
| Analyse des exigences logicielles | X | X | X |
| Conception architecturale du logiciel |  | X | X |
| Conception détaillée du logiciel |  |  | X |
| Implémentation de l'unité logicielle | X | X | X |
| Vérification de l'unité logicielle |  | X | X |
| Intégration logicielle et tests d'intégration |  | X | X |
| Test du système logiciel | X | X | X |
| Version du logiciel | X | X | X |

Le but de nos études est d’arriver à la phase Vérification de l’unité logicielle.

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

[place your text here ]

## **Conclusion**

[place your text here ]

# **Chapitre 3 Analyse et spécification de besoins**

## **Introduction**

## **Spécification des besoins**

|  |  |  |
| --- | --- | --- |
| Nom des composantes | Références | Rôle |
| Carte STM32/Core 743I | H743IIT6 |  |
| Moteur pas à pas | NEMA17 42HS |  |
| Driver | L6474 |  |
| Carte ESP | 8266 |  |
| Clavier à membrane | \*\*\*\*\* |  |
| Capteur de position | 3590S-2-103L |  |
| Capteur de proximité | OPB745WZ |  |
| Capteur de diamètre | PTL01-15W0-103B1 |  |
| Jauge de contrainte | FSS020WNST |  |
| Adaptation Analogique | AD620 |  |
| Module alimentation | KMF0.2113.11 |  |
| Batterie | RRC2054 |  |
| LCD TFT 4.3 | AFY480272-4.3INTH-R |  |
| Buzzer | 458-1402-ND |  |
| Led RGY | \*\*\*\*\*\* |  |
| Memory card Connector | SD-RSMT-2-MQ-WF |  |
|  |  |  |

Tableau 1 : liste des composants

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

### **Architecture Hardware :**

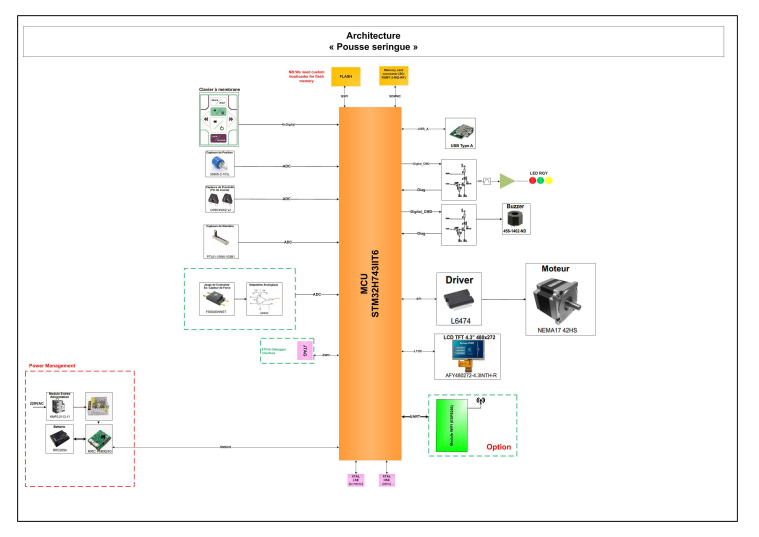


Figure 1 : Architecture "Pousse seringue"

#### **Matériels utilisée :**

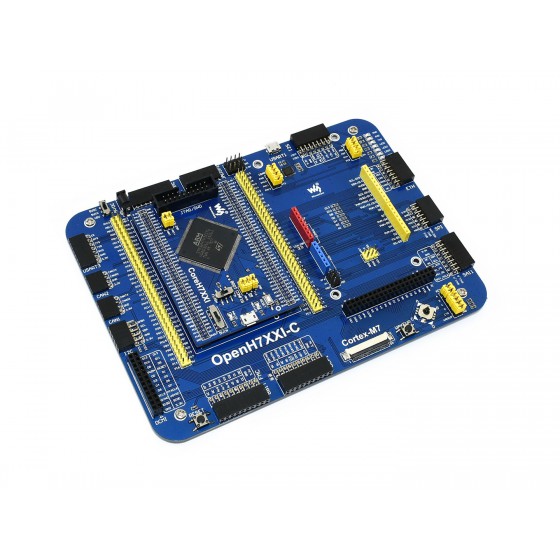
* **STM32H7 :**

Figure 2 : Carte STM32H7

* **MOTEUR PAS à PAS (NEMA17) :**



Figure 3: Moteur PAS à PAS (NEMAI17)

* **DRIVER L6474 :**

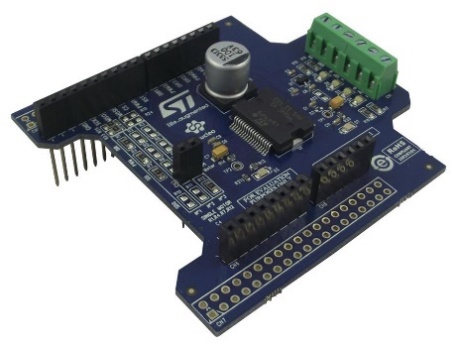
****

Figure 4 : DRIVER L6474

* **LCD TFT 4.3 :**

Figure 5 : LCD TFT 4.3

* **ESP8266 :**

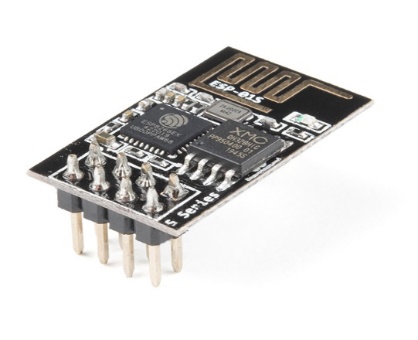
****

Figure 6 : Node MCU ESP8266

* **Capteur de position (3590S-2-103L) :**

****

Figure 7 : Capteur de position (3590S-2-103L)

* **Capteur de diamètre (PTL01-15W0-103B1) :**

****

Figure 8 : Capteur de diamètre (PTL01-15W0-103B1)

* **Capteur de proximité (OPB745WZ) :**

****

Figure 9 : Capteur de proximité (OPB745WZ)

* **Capteur jauge de contrainte (FSS020WNST) :**

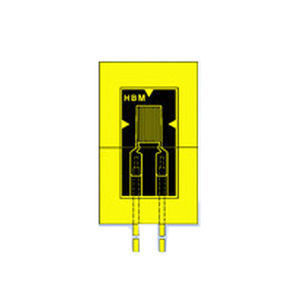
****

Figure 10 : Capteur jauge de contrainte (FSS020WNST)

* **Batterie (RRC2054)**



Figure 13 : Batterie (RRC2054)

* **Buzzer (458-1402-ND) :**



Figure 14: Buzzer (458-1402-ND)

* 1. 3.2 Architecture Software :
* **Logiciels utilisés**
* **STM32 CUBEIDE :**



* **TOUCHGFX :**



* **ARDUINO IDE :**



# **Chapitre 4 : Conception et réalisation**

## **Introduction**

## **OS kernel**

### **2-1 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 maximale.

* **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.

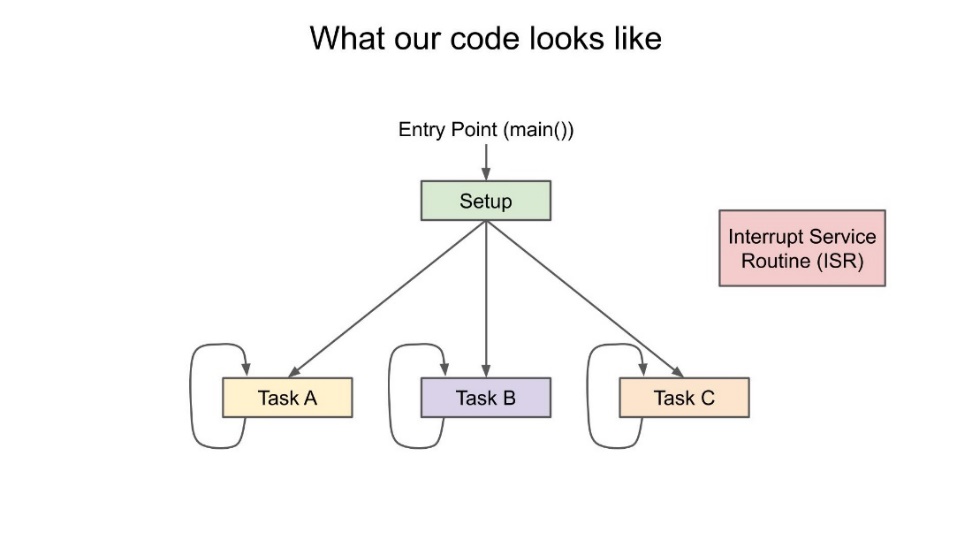
RTOS garantis la performance maximale du processeur et la bonne gestion de la mémoire ainsi que le fonctionnement sans erreur (Error-Free) offert par ces types de systèmes.

* **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.



### **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 \*paramètres)**

**{**

**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**

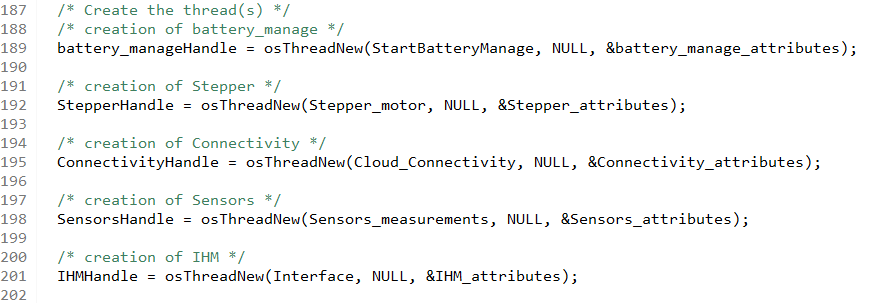
La connectivité admet le même degré de priorité que la tache IHMHandle osPriorityAboveNormal, toutes les données nécessaires sont envoyées à travers cette tache vers le Cloud afin de les récupérer en temps réels par une application de supervision.

* **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**.**

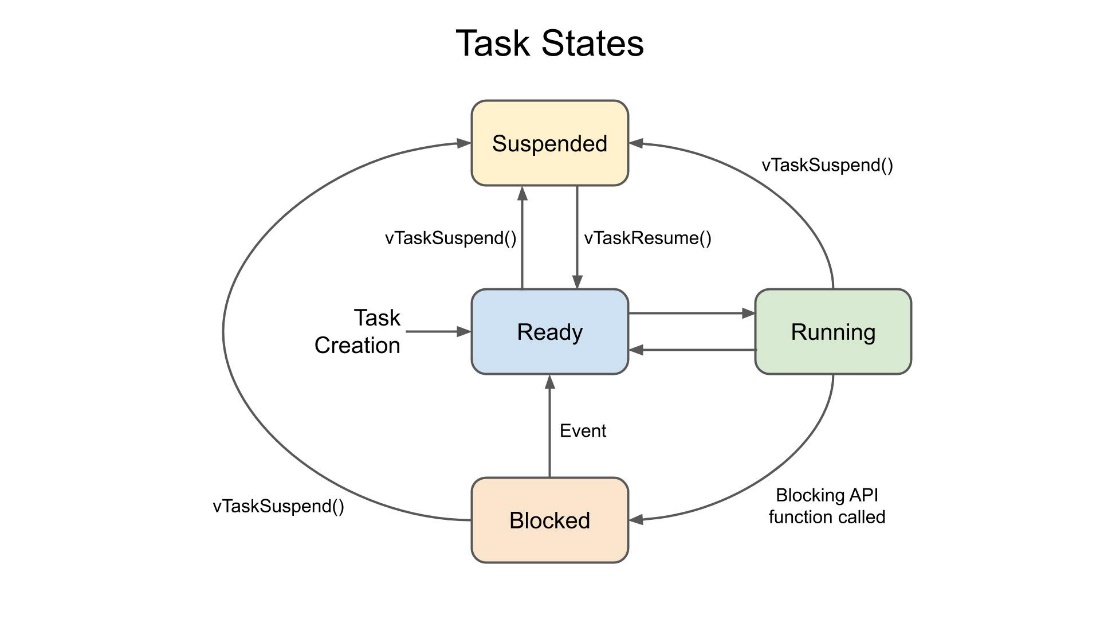


Figure 16 : état des taches

* **Queues (File 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 c – à - d 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.

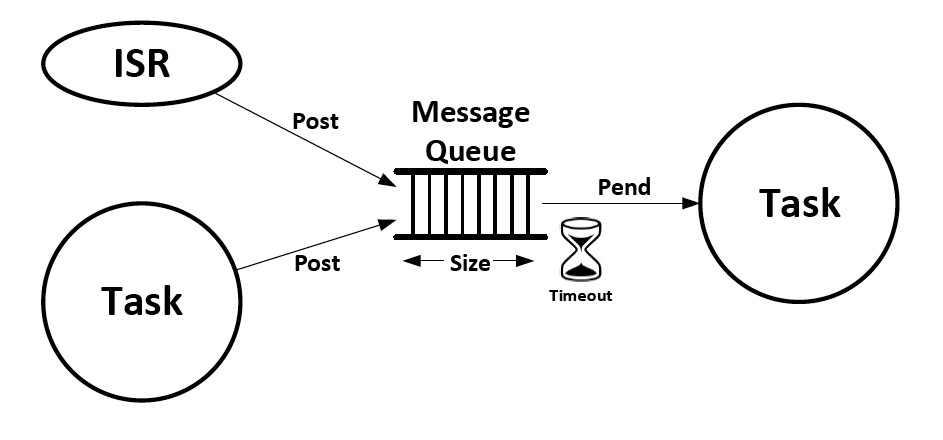
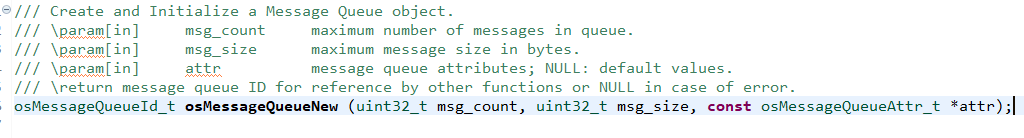


Figure 17 : l'utilisation de Queuse RTOS

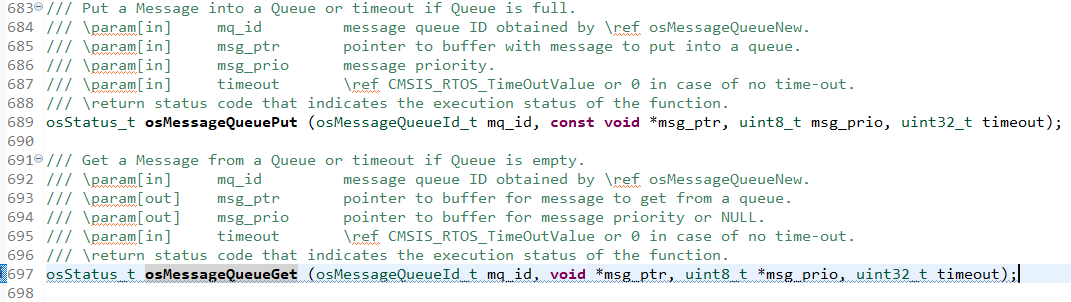
La bibliothèque “Cmsis\_Os2” dispose les fonctions nécessaires pour créer, Lire et écrire dans les files d’attente (queues).

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.

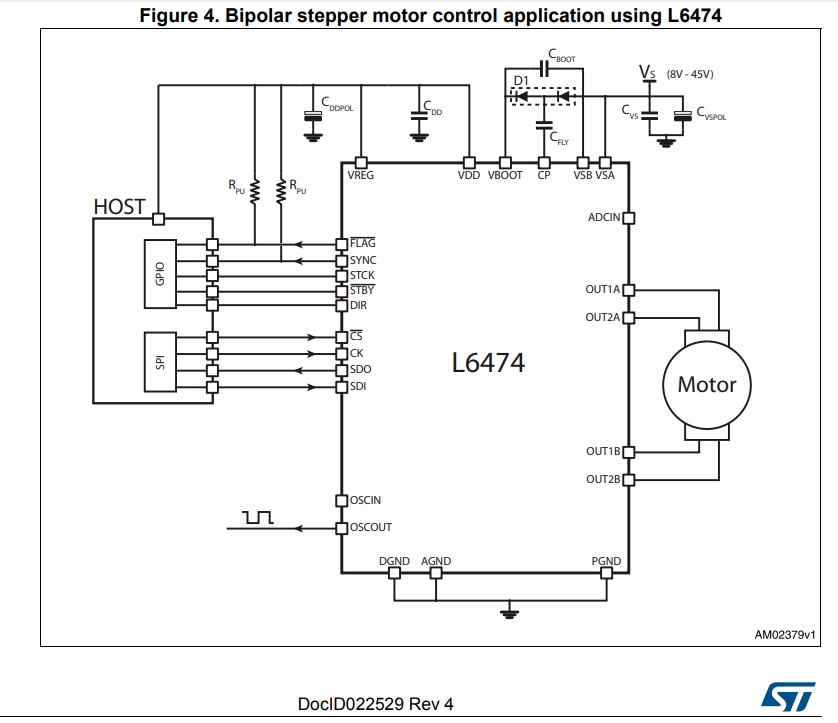
La fonction “osMessageQueueGet” lit le contenu de la file d’attente (queue) dont l’id est passé en paramètre, si la queue est vide (pas de messages) et elle a dépassé le délai maximal d’attente “timeout”, la fonction retourne “osErrorTimeout” si non elle retourne “osOK”.



## **Moteur pas à pas**

### **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.

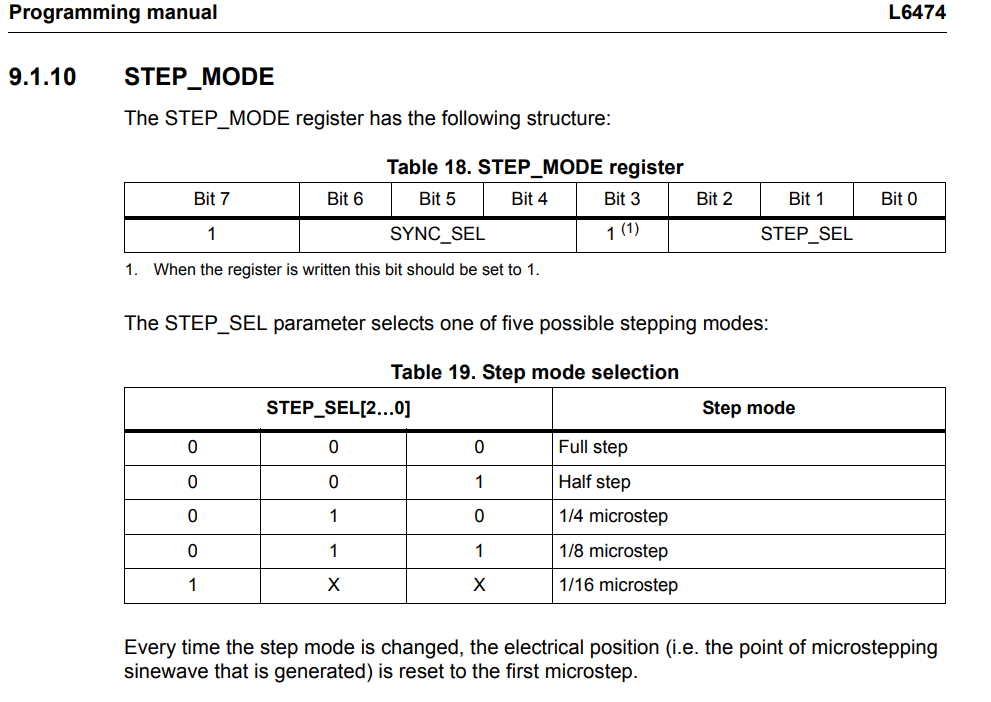


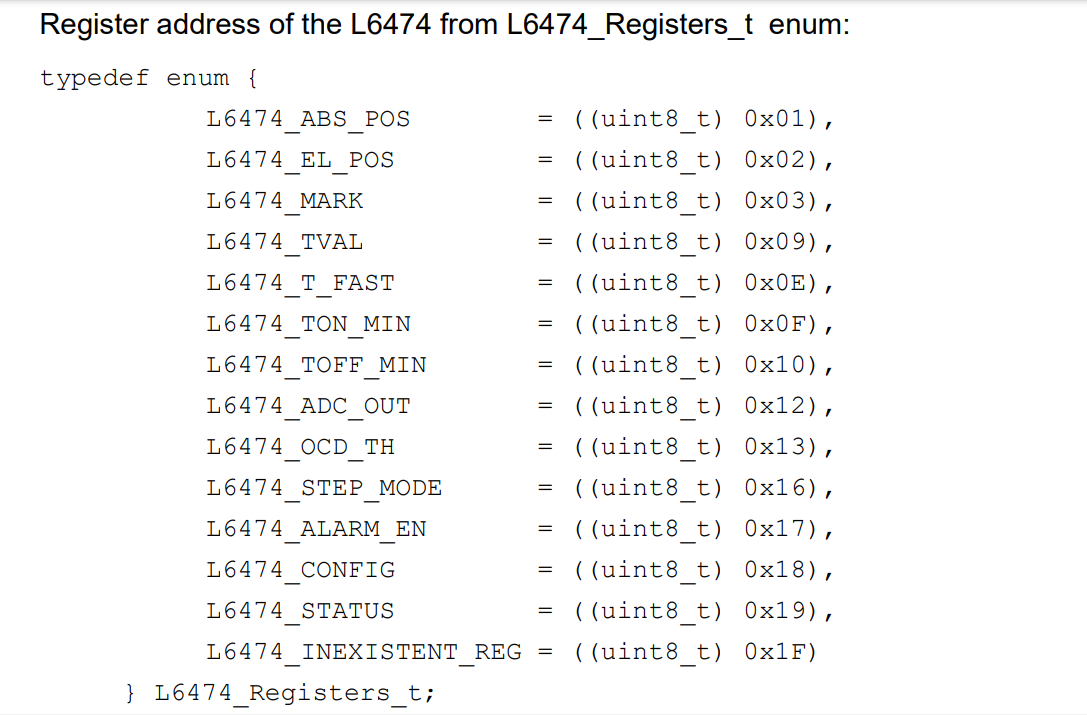
Nous travaillons avec la carte d’expansion **x-nucleo-ihm01a1** basée sur le L6474 Driver et le moteur pas à pas **Nema 17**.

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**).



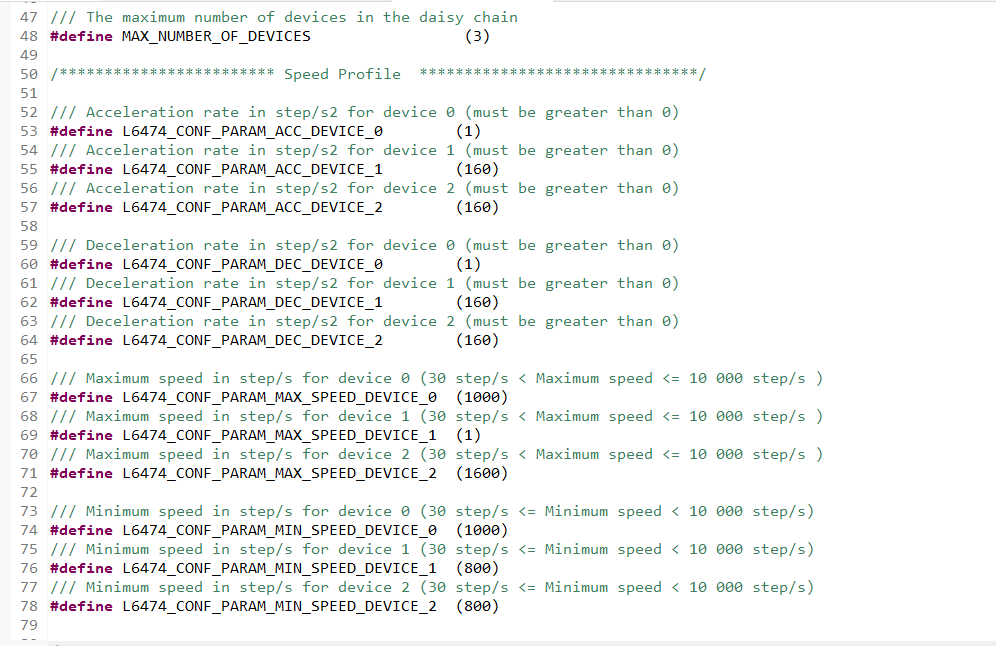


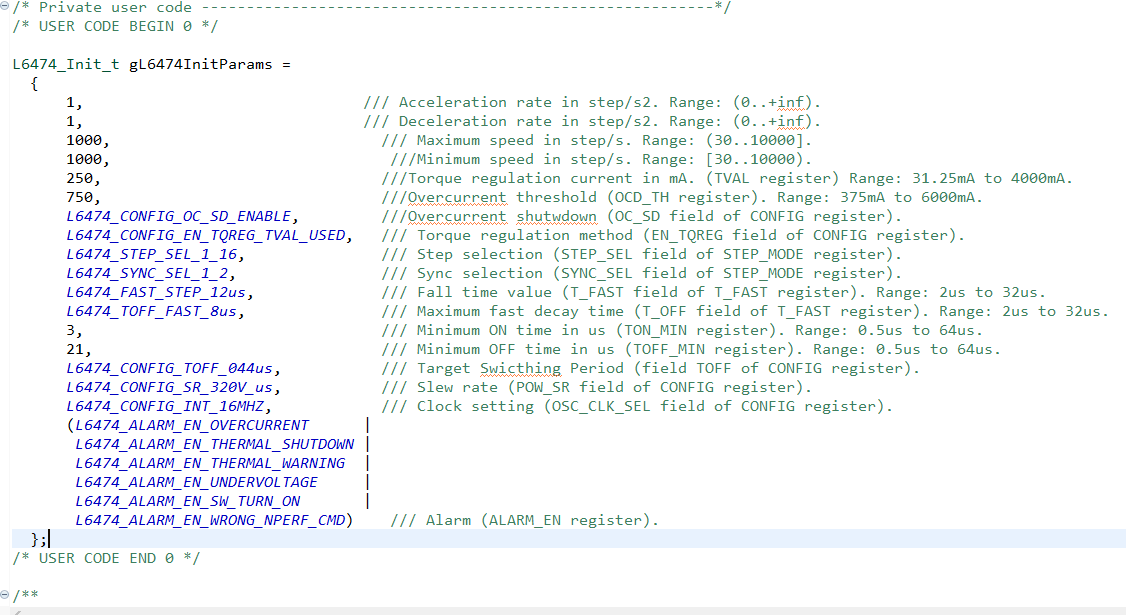
La bibliothèque fournit par ST “**X-CUBE-SPN1**“ peut gérer tous les commandes bas niveau à travers des fonctions prédéfinit qui envoient des trames bien déterminés contenant l’adresse du registre ainsi que le code commande correspondant et des arguments si nécessaire, mais le problème que cette bibliothèque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



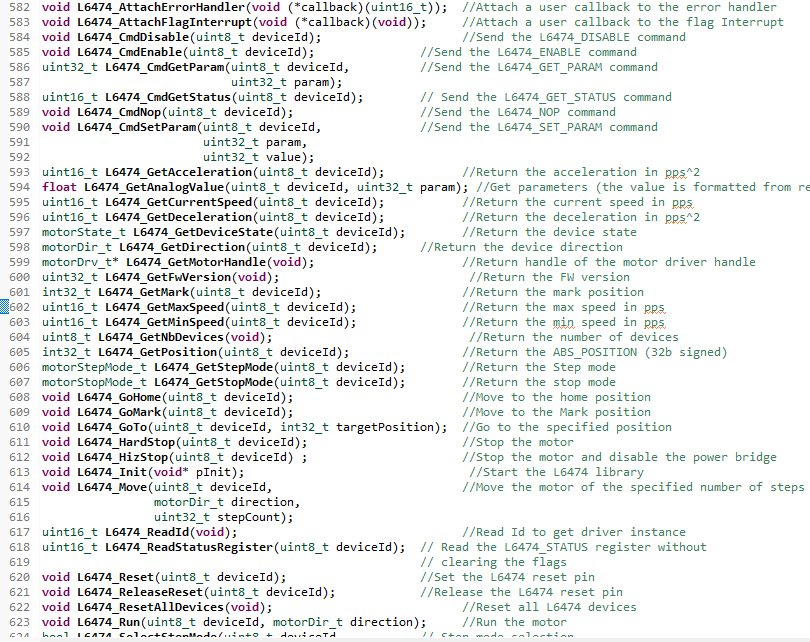


Pour avoir une vitesse constante toute au long de l’opération de l’injection, l’accélération et la décélération sont initialisés à 1 (sans accélération / décélération).

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

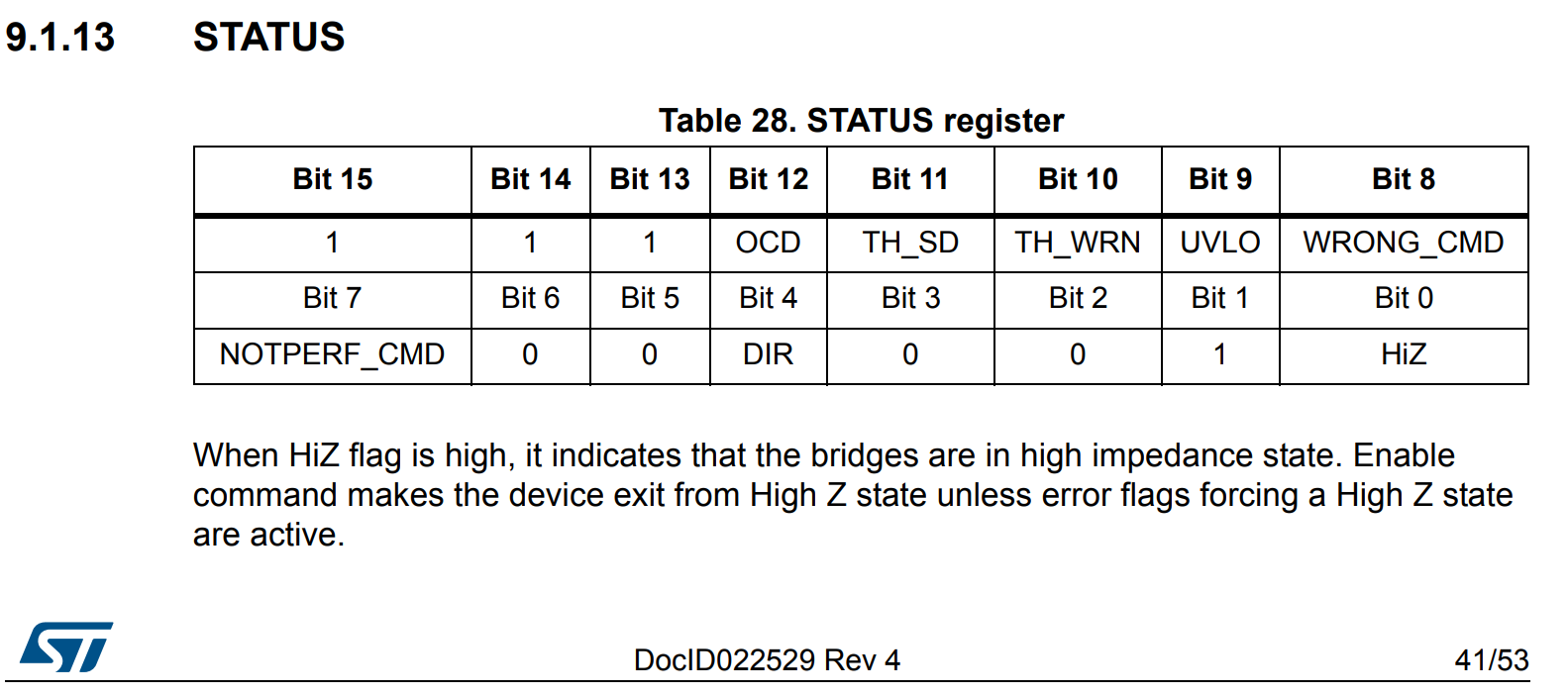
**“uint16\_t L6474\_SetMaxSpeed(uint8\_t deviceId, uint16\_t newMaxSpeed)”**

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

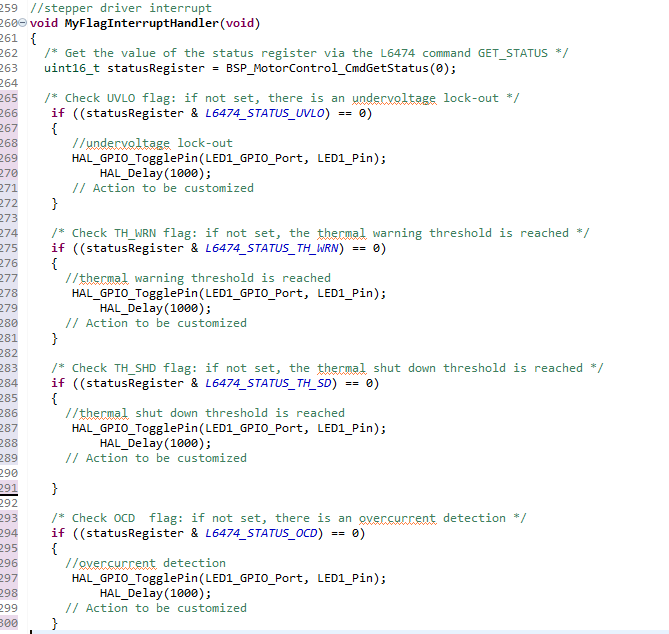


### **Interruptions**

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.



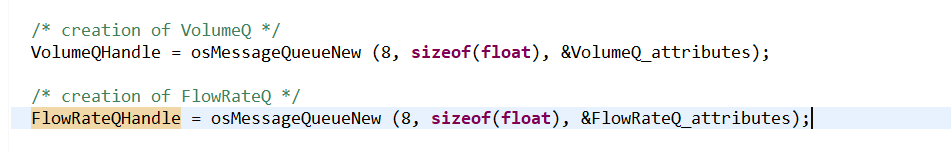
Lors de l’initialisation on définit une limite pour chaque grandeur, si elle est dépassée une interruption est lancée dans notre programme, elle dispose toujours la priorité maximale. En cas d’interruption, une alarme est activée en fonction de l’état du driver. Pour le moment les alarme s’agissent des 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 paramètres 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.

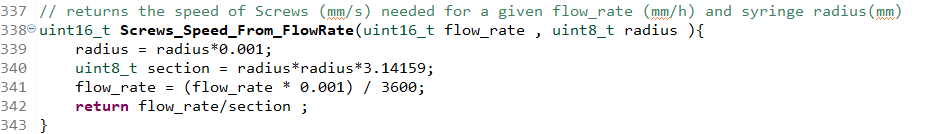
On commencera avec le terme “**débit volumique**” qui désigne la quantité de liquide qui circule dans une canalisation durant un laps de temps déterminé. Exprimé en litres par seconde (L/s), litres par minute (L/mn) ou en mètres-cubes par heure (m3/h). Dans notre cas, le liquide c’est le médicament à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

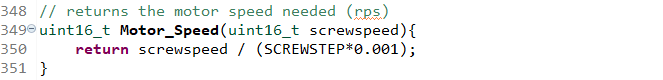
Pour une section d’un canal donnée, plus la vitesse de passage est grande, plus le débit d'écoulement sera important :

**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.



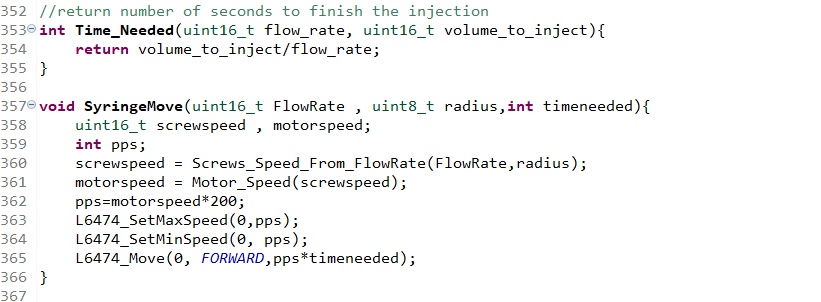
L’écrou fait le déplacement d’un pas quand le moteur pas à pas termine une tour complète (une révolution). Donc la vitesse du moteur (en tours/seconde) est :



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.**



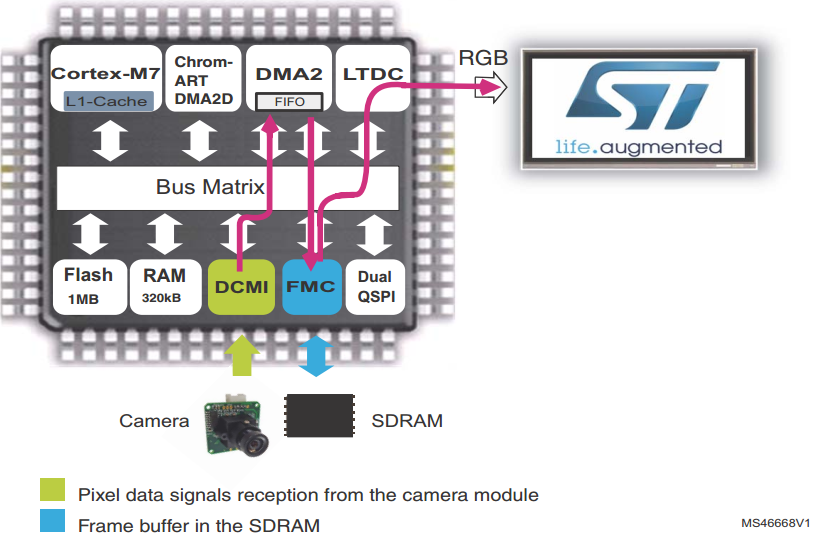
## Interface homme machine « IHM**»**

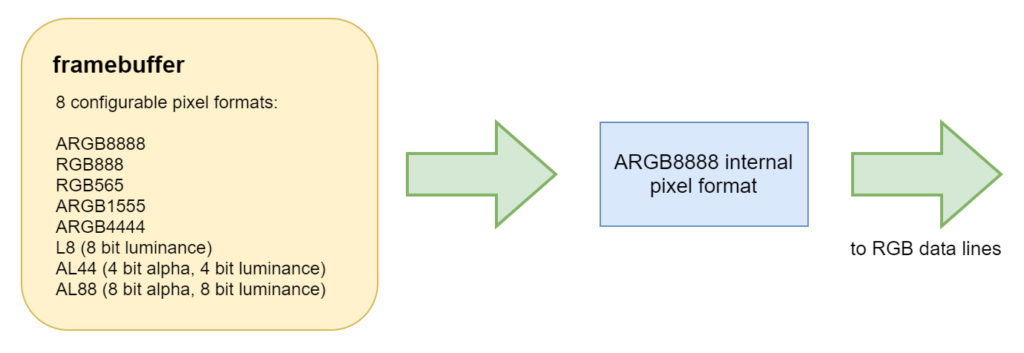
L'écran tactile (4.3inch 480x272 TFT) va assurer l’interaction entre le médecin et la pousse seringue et vice versa, en fait toutes les données relatives à l’injection sont tapées à travers le médecin. Or que les données d’avancement et de l’état de la seringue ainsi que les alarmes sont affichées à travers l’écran.

* **LTDC (LCD-TFT display Controller)**

Grace au périphérique LTDC on pourra interfacer l’écran avec la carte STM32, ce périphérique est responsable à transmettre l’ensemble des pixels d’une image sous une format bien déterminé (hors de portée de ce rapport due à sa taille).

Le LTDC consomme beaucoup de ressources que ce soit au niveau des pins (>40 pins) ou au niveau calcules et traitements. C’est pour cela qu’on a utilisé un accélérateur **DMA2D**.





* **Architecture (MVP)**

**On a utilisé le Framework TouchGFX pour développer l’interface homme machine.**

**Le X-CUBE-TOUCHGFX est une expansion développée par ST pour faciliter l’intégration avec CUBE-IDE, en fait grâce à la méthode drag and drop, on peut faire le design de l’ensemble des vues (view) et puis on génère le code en C++ selon une architecture appelé MVP avec l’approche orienté objets.**

Modèle de conception modèle-vue-présentateur

Les interfaces utilisateur TouchGFX suivent un modèle architectural appelé MVP qui est une dérivation du modèle Modèle-Vue-Contrôleur (MVC). Les deux sont largement utilisés pour créer des applications d'interface homme machine.

Dans MVP, les trois classes sont définies comme suit :

* **Le *modèle (Model)*** est une interface de données, elle sert également de lien entre la partie non-UI (Backend system) et la partie UI (User Interface) du projet : c’est le cœur de l’interface graphique.
* **La *vue (View)*** est une interface passive interface passive qui affiche les données et acquiert les informations de l’utilisateur (via les différents widgets de touchgfx ex : zone de texte, image, bouton, menu déroulant, curseur…)
* **Le *présentateur (Presenter****)* est une classe qui agit sur le modèle et la vue. Elle récupère les données du modèle et les formate pour les afficher dans la vue.

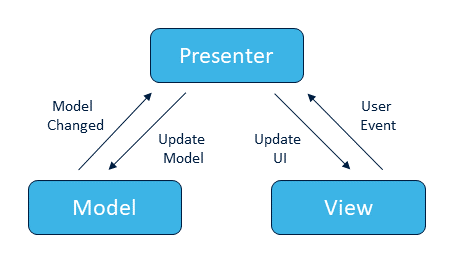


Figure 18 : Modèle de conception modèle-vue-présentateur

* **Les interactions dans TouchGFX**

**Dans TouchGFX Designer, une interaction est constituée d'un trigger et d'une action :**

* **Un trigger est ce qui va démarrer l'interaction - ce qui doit se passer dans notre application pour que l'action ait lieu.**
* **Une action est ce qui va se passer après qu'un déclencheur ait été émis.**

**Un écran vide n'aura que quatre actions disponibles :**

* **Call new virtual function**
* **Change screen**
* **Execute C++ code**
* **Wait for**
* **Flux des données**

Dans TouchGFX, la communication avec la partie non-UI de l'application, appelée ici le **backend**, se fait à partir de la classe Model. Le système backend dans notre cas est FreeRTOS avec tous les taches que nous avons parlé précédemment. Du point de vue TouchGFX, cela n'a pas vraiment d'importance, tant qu'il s'agit d'un composant avec lequel il est capable de communiquer.

Dans notre cas, le protocole de communication entre la partie graphique et le backend est géré à travers les queues (files d’attentes) que nous avons déjà utilisées pour la communication inter-taches.

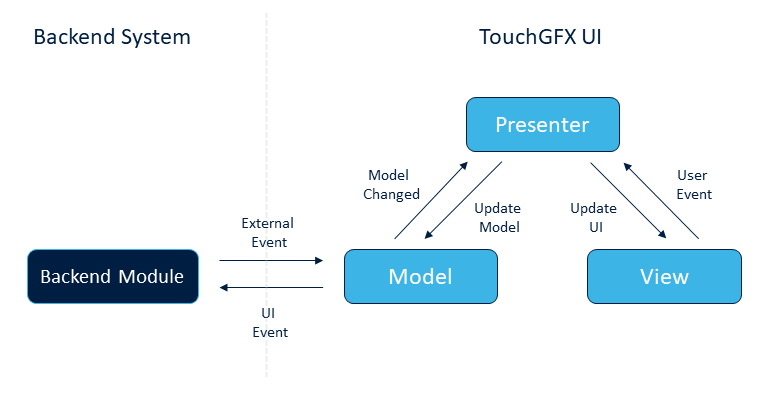
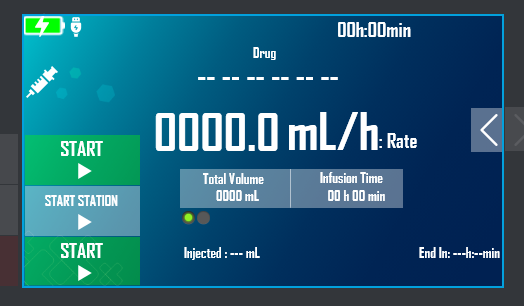
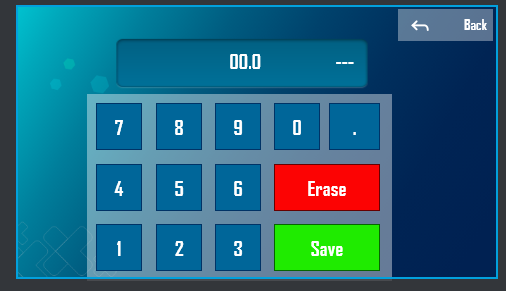


Figure 19 : Modèle-Vue-Présentateur et communication externe

On prend l’exemple du débit d’injection :

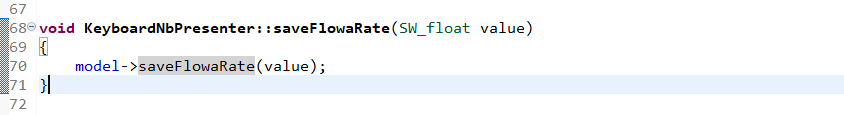
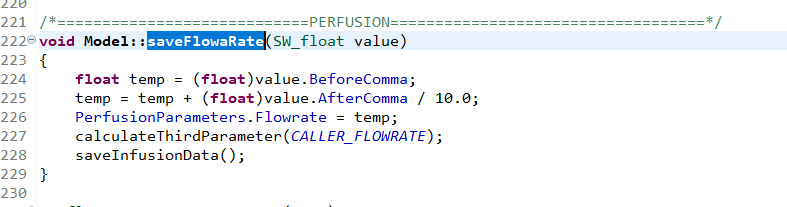
Le Médecin tape sur le bouton “RateBtnBuffer”, et grâce à l’interaction “KeyboardRate” il est dirigé vers une interface pour taper en chiffre le débit. Pour chaque chiffre tapé une interaction eu lieu pour le stocker dans un buffer.





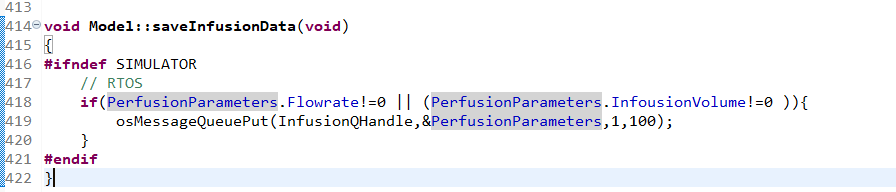
Puis il tape le bouton “Save” qui fait l’appel d’une fonction “SaveData” définit dans la classe View de l’interface “KeyboardNb”. Cette fonction passe la valeur du débit à la fonction “saveFlowaRate” définit dans la classe Presenter de la méme interface.

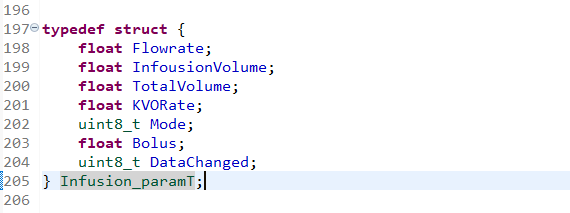
Enfin cette dernière fait l’appel de la fonction “saveFlowaRate” définit dans la classe Model.



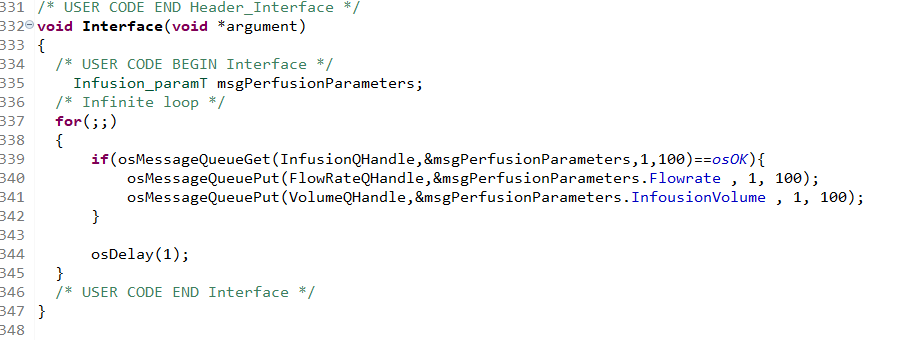
La fonction “saveInfusionData” transmet les données vers le backend à travers une queue ”InfusionQHandle” sous format d’une structure C de type “Infusion\_paramT” qui admet tous les paramètres d’infusion.

La fonction “saveInfusionData” transmet les données vers le backend à travers une queue ”InfusionQHandle” sous format d’une structure C de type “Infusion\_paramT” qui admet tous les paramètres d’infusion.





Dans le backend, la tâche “IHMHandle” reçoit toutes les données envoyées d’après la queue d’infusion et les transmet vers d’autre file selon le besoin.



## **Capteurs et mesures**

* **Capteur de diamètre**

**Solution proposée :**

Le calcule de débit d’injection, comme il est indiqué précédemment, nécessite une connaissance de la section de la seringue. Afin de calculer ce paramètre il faut connaitre le rayon.

Généralement, on a une idée sur les rayons possibles d’après le type de la seringue. Notre solution est de détecter dans quelles marges se trouve le diamètre de la seringue et puis assigner la valeur convenable selon cette marge.

Le capteur s’agit d’un potentiomètre linéaire ou bien “slide potentiometer” qui est lié au vérin de la pince à seringue (syringe clamp). Le potentiomètre va retourner une valeur pour chaque niveau du curseur et selon cette valeur on attribue un diamètre.

Par exemple, on travaille avec les seringues de type x qui admettent ces diamètres possibles :

* 0.34mm pour le volume 5 mm^3
* 0.49mm pour le volume 10 mm^3
* 1.03mm pour le volume 50mm^3
* …

Si le capteur indique x ou y la variable rayon reçoit 0.49mm et puis on fait le calcul de la section.

**Implémentation**

Le potentiomètre est alimenté avec une tension 3.3v qui représente sa valeur maximale et la sortie du potentiomètre linéaire est reliée à une entré ADC (Analogue to digital Converter) du microcontrôleur qui admet une résolution 16 bit, donc les valeurs possibles sont entre 0 (0v) et 65535 (3.3v).

On a activé La fonction de mode continu (continuous mode) qui permet à l'ADC de travailler en arrière-plan. L'ADC convertit les canaux en continu sans aucune intervention du CPU.

.

.

...

* **Capteur de position**

**Solution proposée**

Avec le même principe On a utilisé un potentiomètre rotatif avec des dents pour qu’il puisse suivre la rotation de l’arbre moteur il s’agit d’un feedback pour contrôler l’erreur du moteur et augmenter sa performance. D’après les valeurs retournées par le potentiomètre on peut aussi conclure la position de la seringue ainsi que le volume restant.

**Implémentation**

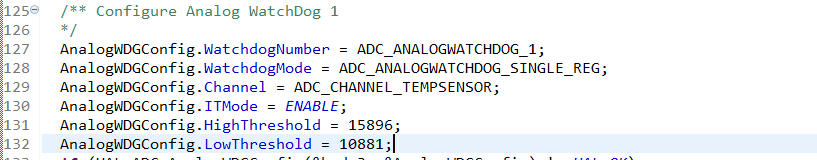
Le potentiomètre rotatif

Le potentiomètre rotatif est interfacé de la même manière que celui utilisé pour capturer le diamètre. La seule chose qui se diffère est la partie de l’interprétation des valeurs retournées par le capteur. En fait, pour détecter la position de la seringue on doit

* **Capteur de température**

Avec toutes les taches que notre carte (Stm32H7) va gérer, il est probable qu’elle surchauffe pour une longue durée d’utilisation, même si elle peut supporter jusqu’à 140°c avec une fréquence de 480MHZ.

Mais pour se protéger et rester dans la zone hors stress, on a utilisé le capteur de température interne qui est connecté directement à l’**ADC3**. La conversion analogique numérique est faite dans le background avec le mode continu (810.5 cycles) et le **watchdog analogique** qui génère une interruption (HAL\_ADC\_LevelOutOfWindowCallback(ADC\_HandleTypeDef \*hadc)) si la conversion est hors la marge spécifiée.



La limite supérieur ( HighThreshold ) et la limite inférieur (LowThreshold ) sont calculées d’après la formule suivante :

tCelsius = (110 - 30) \* (readValue - \*TEMPSENSOR\_CAL1\_ADDR) / (\*TEMPSENSOR\_CAL2\_ADDR - \*TEMPSENSOR\_CAL1\_ADDR) + 30

tcelsius : la température en degré Celsius.

readValue : le résultat de conversion analogique numérique.

\*TEMPSENSOR\_CAL1\_ADDR : contenant l’adresse de la valeur de calibrage pour la température 110°c (stocké dans la mémoire morte de la carte).

\*TEMPSENSOR\_CAL2\_ADDR : contenant l’adresse de la valeur de calibrage pour la température 30°c (stocké dans la mémoire morte de la carte).

L’interruption est appelée quand le résultat de conversion est en dehors de cette fenêtre :

Max : 15896 qui correspond à 100°c

Min : 10881 qui correspond à 0°c

Pour chaque interruption de l’ADC3 on lance une alarme de type “overheating”.



* **Capteur de pression**

## **Connectivité**

La connectivité est une élément clé dans notre projet, vu qu'elle est rarement utilisée dans les pousses seringues existantes dans le marché. Son rôle est de permettre la surveillance en temps réel de l’état d’injection à distance et n’importe où.

**Solution proposée :**

Beaucoup de solution peuvent être implémentées, mais nous avons les contraintes de certifications. Pour cela on a choisi d’envoyé les données au cloud, pour permettre l’application mobile qui (en cours de développement)

## **Taches en cours de développement**

[place your text here]

## **Conclusion**

[place your ext here ]