



# Portage de FreeRTOS sur le calculateur NanOpral

Par  
**Quentin Comte-Gaz**

Stage réalisé à UXP  
(Seyssinet Pariset (France, 38))

Version numérique (en couleur) :  
[http://quentin.comte-gaz.com/stages/rapport\\_stage\\_uxp\\_2013\\_v2.pdf](http://quentin.comte-gaz.com/stages/rapport_stage_uxp_2013_v2.pdf)

## Remerciements

---

Je tiens tout particulièrement à remercier mon maître de stage M. Guillaume Bossard pour la confiance et l'aide qu'il m'a accordé avant et pendant le stage. Je tiens également à le remercier pour m'avoir permis de réaliser ce stage à la fois intéressant, enrichissant et en lien direct avec mes ambitions de poursuite d'étude (spécialisation dans les systèmes embarqués).

Je souhaite également remercier M. Jay Robert, Président d'UXP, pour m'avoir accueilli au sein de son entreprise.

Pour conclure, j'aimerais remercier et témoigner ma gratitude à toute l'équipe d'UXP, pour l'accueil très humain et convivial qu'elle m'a réservé.

# Sommaire

---

Introduction.....	4
<b>PARTIE 1 : PRESENTATION DE L'ENTREPRISE .....</b>	<b>5</b>
<b>I) Fiche d'identité.....</b>	<b>5</b>
<b>II) Historique.....</b>	<b>6</b>
<b>III) Partenaires .....</b>	<b>7</b>
<b>IV) Clients .....</b>	<b>8</b>
<b>PARTIE 2 : PRESENTATION DU SUJET ET DE MON RÔLE LORS DU STAGE .....</b>	<b>9</b>
<b>I) Le domaine d'étude.....</b>	<b>9</b>
<b>II) Présentation du groupe de travail.....</b>	<b>9</b>
<b>III) Explication détaillée du fonctionnement des RTOS.....</b>	<b>11</b>
<b>IV) Les outils mis à ma disposition.....</b>	<b>15</b>
<b>V) Mon rôle .....</b>	<b>18</b>
<b>PARTIE 3 : LES APPORTS PERSONNELS DE CE STAGE .....</b>	<b>26</b>
<b>I) Enrichissement personnel grâce au stage .....</b>	<b>26</b>
<b>II) Enrichissement personnel grâce à l'ouverture d'esprit de l'équipe d'UXP.....</b>	<b>26</b>
<b>CONCLUSION .....</b>	<b>27</b>
<b>ANNEXE A : Exemple en C de gestion multitâche sur FreeRTOS .....</b>	<b>28</b>
<b>ANNEXE B : Extrait du code de la librairie créée durant le stage.....</b>	<b>31</b>
<b>ANNEXE C : Documentation technique du NanOpral .....</b>	<b>33</b>
<b>ANNEXE D : Comparaison entre FreeRTOS et CoOS .....</b>	<b>35</b>
<b>ANNEXE E : Utilisation de Doxygen .....</b>	<b>41</b>
<b>ANNEXE F : Notion de protection des ressources.....</b>	<b>46</b>
<b>ANNEXE G : Documentation Doxygen finale (Javadoc).....</b>	<b>47</b>
<b>ANNEXE H : Environnement de développement pour le client final .....</b>	<b>48</b>
<b>ANNEXE I : Evaluation de stage .....</b>	<b>49</b>

# Introduction

---

Dans le cadre de mon cursus de deuxième année d'école d'ingénieur à l'ENSEA implanté à Cergy, j'ai eu l'opportunité d'effectuer un stage dans une PME du nom d'UXP pour une période de deux mois. Ce dernier avait pour objectif de faire une synthèse de mes connaissances théoriques, d'améliorer mon niveau dans le domaine des microcontrôleurs (tout particulièrement dans le domaine des systèmes embarqués), de découvrir la façon de travailler dans une petite équipe et enfin de participer à l'étude d'un problème concret avec un réel impact sur le marché.

Pour ce stage de deuxième année, j'ai trouvé plus intéressant d'orienter mon choix dans un domaine des systèmes embarqués dans lequel je vais me spécialiser l'année prochaine. En effet, ce domaine occupe aujourd'hui une place qu'il n'est plus possible d'ignorer. Ce domaine qui consiste à concevoir des systèmes électronique-informatique consommant peu d'énergie a de l'avenir car il est en pleine expansion (exemples d'application : téléphone portable, véhicule, tablette, capteur sans fil, ...). Mon intérêt pour le domaine de l'électronique-informatique a vu le jour depuis de nombreuses années. Cependant, mon intérêt pour les systèmes embarqués n'a germé que depuis cette année avec le projet d'alarme sans fil que j'ai réalisé ainsi qu'avec les cours de microprocesseur de l'école d'ingénieur.

Afin de vous faire partager mon expérience, je débiterai par la présentation de l'entreprise UXP où j'ai travaillé, puis je vous présenterai mon sujet, mon rôle et mes responsabilités ainsi qu'une explication détaillée du travail effectué, enfin je terminerai par un récapitulatif des apports et sentiments liés au stage suivi d'une rapide conclusion.

# PRESENTATION DE L'ENTREPRISE

---

## **1) Fiche d'identité :**

### *Nom de l'entreprise :*

UXP (UniX et Productique)

### *Secteur d'activité :*

UXP est concepteur, développeur et intégrateur de technologies et solutions avancées dans les domaines de **l'automatisation, de l'informatique industrielle, des systèmes embarqués et de l'électricité industrielle**. UXP développe des solutions produits (logicielles et matérielles) pour bâtir des solutions métiers ouvertes dans les domaines de l'Industrie, l'Energie, l'Embarqué, le BTP et la Montagne. UXP intervient comme support (interventions à distance ou sur place), formation (automatisme et informatique industrielle), assistance (cahier des charges, développement et accompagnement) ainsi qu'en conseil et expertise (performance industrielle et audit).

### *Implantation de l'entreprise :*

12 avenue Pierre de Coubertin  
ZI Percevalière  
38170 SEYSSINET PARISET

### *Nombre d'employés :*

Environ 10 (PME)

### *Date de création :*

1992

### *Contact :*

(+33)4.76.84.28.80

### *Site internet :*

<http://www.uxp.fr>

## II) Historique :

UXP est une SA créée en 1992 dans la région de Grenoble en tant que SSII offrant des **Services et Solutions en Informatique Industrielle, Automatisation, systèmes embarqués et électricité industrielle.**

Le développement d'UXP a bénéficié du partenariat initialisé dès 1992 avec EMR, filiale du groupe Bic. A cette époque, EMR/Bic confie à UXP la responsabilité de la commercialisation de l'atelier logiciel d'automatisme « ALOGRAF » développé à l'origine pour ses besoins internes.

Ce partenariat de haut niveau confirme la maîtrise d'UXP dans l'utilisation des nouvelles technologies pour l'automatisation industrielle et lui permet de se forger rapidement des références prestigieuses.

En avril 2001, le groupe BIC se recentre sur ses activités et cède à UXP la propriété de l'atelier d'automatisme ALOGRAF, donnant un nouvel essor à la société.

UXP mène une démarche Qualité qui confirme sa volonté d'assurer la pérennité de ses solutions.

Aujourd'hui, après une expérience de plus de 10 ans de la société, de plus de 15 ans de ces dirigeants, UXP continue sa démarche de mise en place de **partenariats** ainsi que son développement sur le marché de l'**Informatique Industrielle** tout en élargissant son domaine d'interventions et de compétences. On pourra en effet noter la conception de nombreux produits matériels pouvant utiliser l'outil ALOGRAF et étant destiné aux industriels (NanOpral, NanOstop, OpralBox, ...).



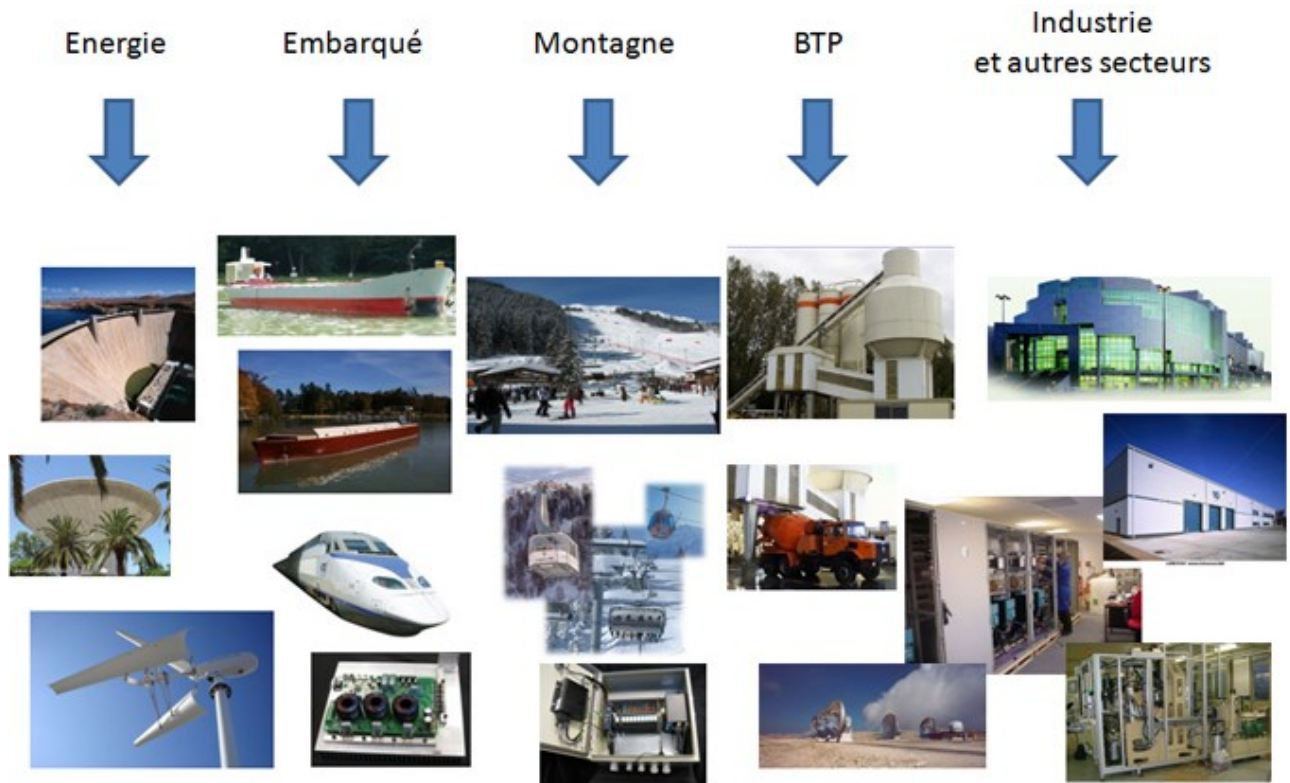
### III) Les Partenaires :

✚ Les 10 principaux partenaires en collaboration avec UXP :

	<p>ADAPTECH (spécialisée dans l'analyse et la conception de systèmes de régulation numérique)</p>
	<p>ADEUNIS RF (spécialisé dans la fabrication et la commercialisation de systèmes de transmission exclusivement sans fil)</p>
	<p>BELIMO (Leader mondial de la technologie des servomoteurs et des vannes).</p>
	<p>EUROLINK (spécialisé dans les liaisons numériques hertziennes à longue portée)</p>
	<p>Kerlink (développe et commercialise des solutions machine to machine pour l'interconnexion d'équipements distants)</p>
	<p>PROGANTEK (partenariat pour des projets de mesure de performance et d'optimisation d'outils de production)</p>
	<p>RESoluCOM (spécialisé dans les solutions pour réseaux de communication industrielle et bus de terrain)</p>
	<p>SELIAtec (bureau d'étude spécialisé en électronique, commande, bancs de tests, mesure et contrôle).</p>
	<p>SIEM (spécialisé dans les machines spéciales, l'informatique industrielle et l'automatisme)</p>
	<p>VALDATA (partenaire exclusif d'UXP, assure les développements et la mise en œuvre des solutions métier basées sur la technologie d'UXP)</p>

#### IV) Les Clients :

UXP possède des clients dans de nombreux secteurs d'activités tels que l'aéronautique, l'agroalimentaire, l'automatisme & le contrôle, l'automobile, le BTP, la chimie, la culture, la défense, l'énergie, GTB/GTC, l'industrie, l'informatique, l'intégrateurs OEM, le manufacturier, le packaging, la recherche & l'enseignement, le sanitaire, les télécoms ainsi que le transport.



*Exemples de projets où UXP a participé*

Voici une petite liste des clients d'UXP :





# PRESENTATION DU SUJET ET DE MON RÔLE LORS DU STAGE

---

## I) Le domaine d'étude

Voici l'intitulé du stage tel qu'il m'a été proposé :

---

**Sujet :** Portage de FREERTOS sur le calculateur d'automatisme NanOpral.

**Description :** NanOpral est architecturé autour du microcontrôleur LPC1768 de chez NXP.

FREERTOS supporte officiellement la gamme LPC17XX, l'objectif du stage consiste à adapter FREERTOS à nos besoins et à développer les bibliothèques d'accès aux différents périphériques (UART, CAN ...).

**Compétences/expériences :** C, programmation de microcontrôleur, Éclipse CDT, GNU/Linux

---

Concrètement, le projet se basait sur un calculateur d'automatisme NanOpral conçu par UXP. Le but était d'intégrer et de comparer deux systèmes d'exploitations en temps réel (RTOS) sur le microcontrôleur du calculateur (LPC1768) ainsi que de concevoir entièrement une bibliothèque pour ce calculateur qui soit également compatible avec le RTOS. Le projet consistait donc à comparer les performances réelles des deux RTOS, créer une bibliothèque ainsi qu'à fournir des exemples concrets d'utilisations pour les clients et le personnel de l'entreprise.

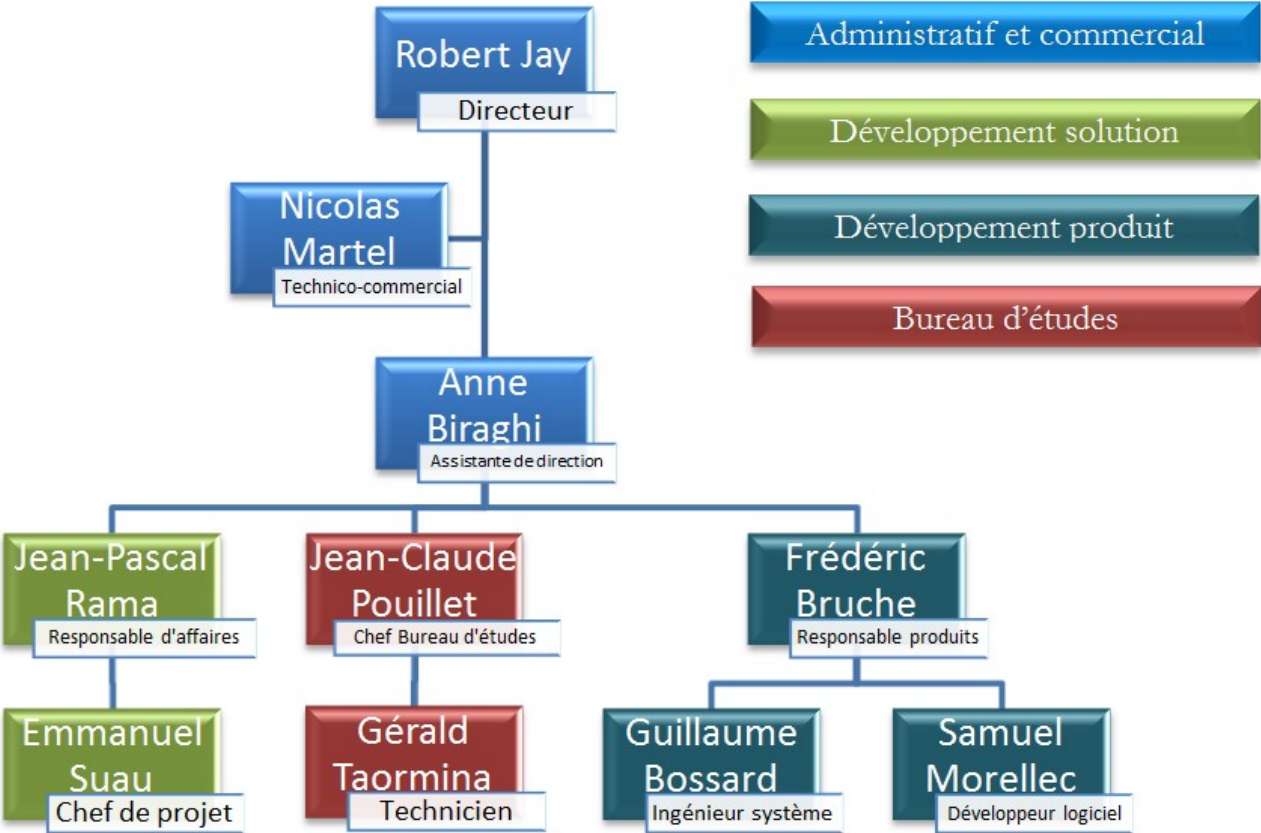
Ce projet demandait des connaissances basiques en électronique, des connaissances poussées en programmation bas niveau (C et assembleur) ainsi qu'en mise en place de tests.

## II) Présentation du groupe de travail

Le stage s'est déroulé dans le pôle « Développement produit » de l'entreprise.

Le groupe de travail était composé de mon maître de stage M. Guillaume Bossard ainsi que de moi-même. J'étais cependant le seul à m'occuper de l'intégration d'un système en temps réel sur le NanOpral.

Voici un organigramme représentant la structure hiérarchique que l'entreprise :



### III) Explication détaillée du fonctionnement des RTOS

Avant de vous présenter mon rôle durant mon stage, je n'ai pas d'autre choix que de vous expliquer ce qu'est un RTOS et vous présenter les principes de bases de ces OS :

Un RTOS, qui signifie **Real Time Operating System (système d'exploitation en temps réel)**, est un système d'exploitation qui a pour but premier de gérer des applications en temps réel.

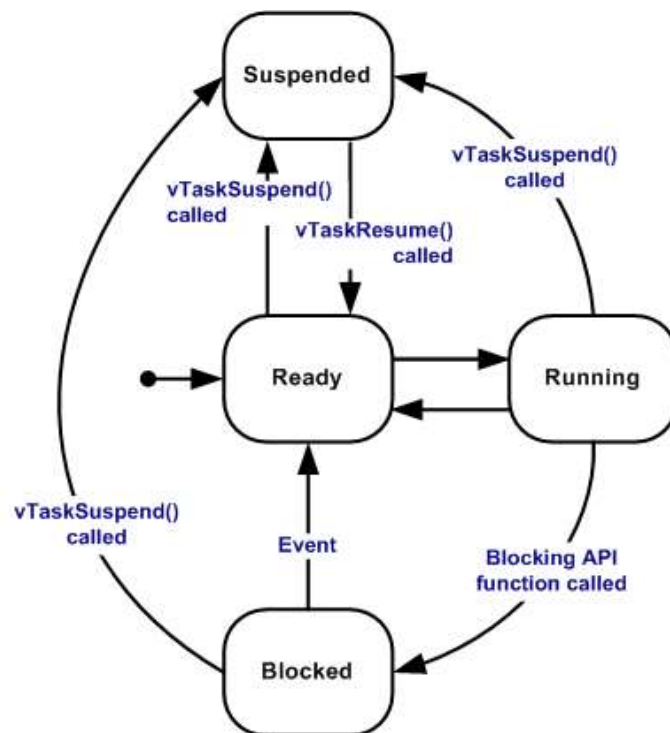
Cela signifie qu'il doit pouvoir gérer des tâches en parallèles tout en les traitant de manière **rapide** et **efficace**.

Cette notion de tâches est très importante car dès lors que l'on lance le RTOS, seuls les tâches ainsi que les interruptions matérielles peuvent se lancer.

Chaque tâche possède une priorité permettant de connaître l'importance de celle-ci. La tâche ayant la priorité la plus élevée se lancera en priorité. Dès que celle-ci a fini, elle laissera la place à une tâche ayant une priorité plus faible et ainsi de suite.

Si deux tâches ont la même priorité, une des deux tâches se lancera. **Après un temps configuré dans l'OS (SYS\_TICK), une interruption propre à l'OS se lance automatiquement (interruption par TICK ou interruption de gestion de tâches) qui lance un gestionnaire de tâche qui va déterminer quelle tâche doit maintenant se lancer.** Dans notre cas, il va passer à la tâche suivante. Et ainsi de suite.

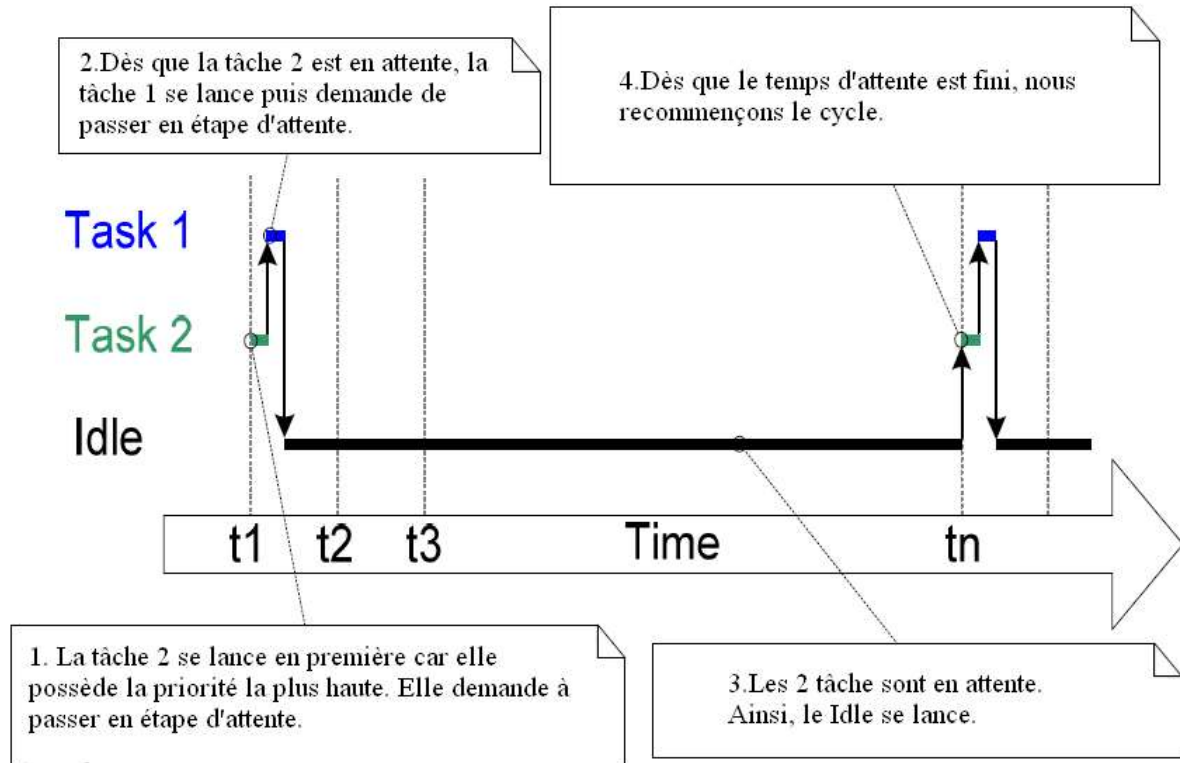
Une tâche est soit en fonctionnement (une seule tâche peut être dans cet état), soit en état d'attente de fonctionnement, soit en état bloqué (en attente d'un événement pour être en attente de fonctionnement), soit en état suspendu (en attente de se faire réactiver).



*Schéma des états et des actions pour passer d'un état à un autre*

Si plus aucune tâche n'est en état d'attente de fonctionnement ou en fonctionnement, alors le RTOS lance une tâche « de très basse priorité » que l'on nomme **Idle**.

Voici un exemple où l'Idle se lance :

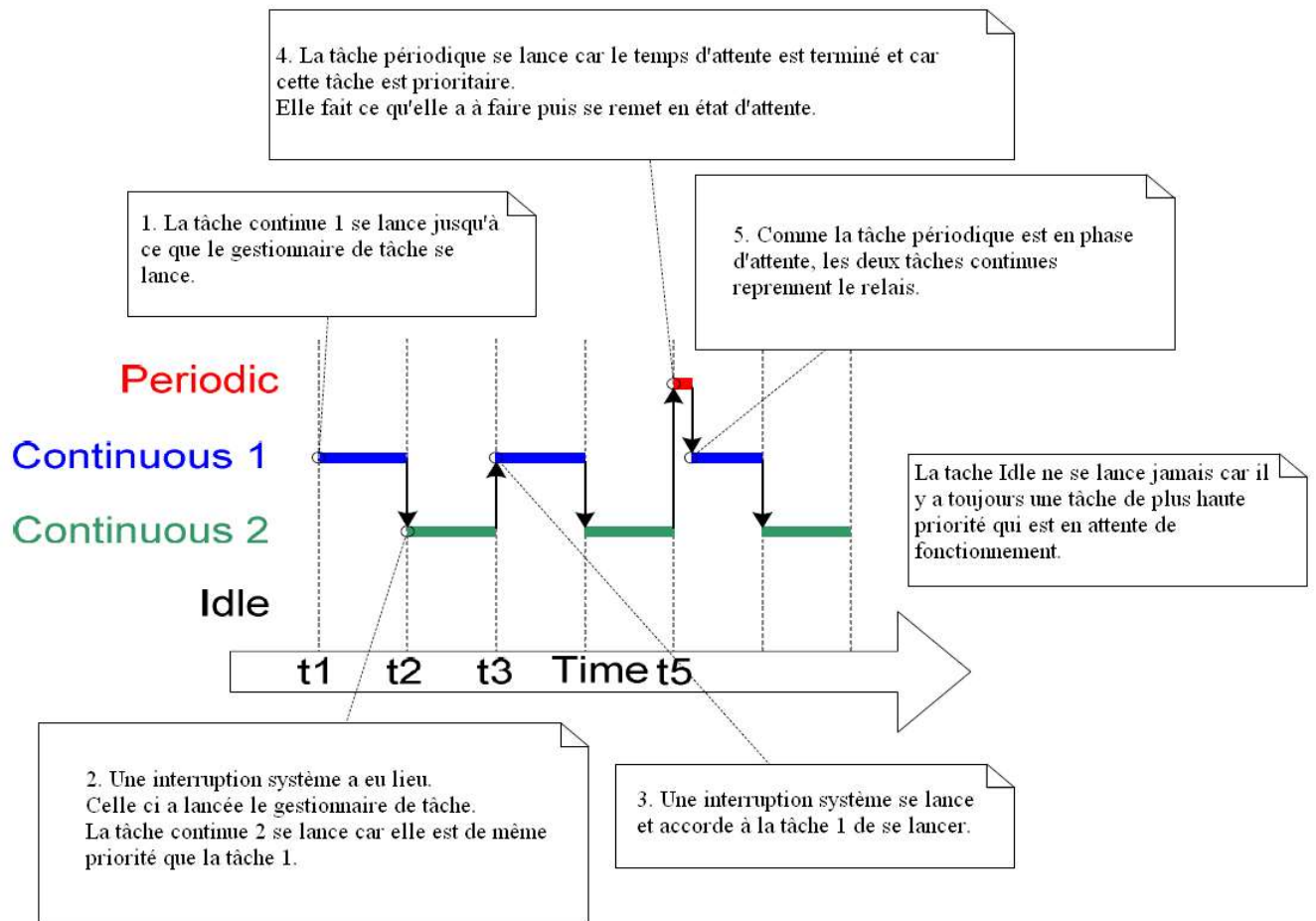


*Diagramme de fonctionnement des tâches dans le cas où les tâches 1 et 2 demandent de s'arrêter*

Il est possible de lancer des tâches de manière périodique (toutes les x ms, on lance la tâche) ou continue (dès que la tâche a le droit de se lancer, elle se lance et s'arrête quand le gestionnaire de tâches se lance).

Pour illustrer ce concept, prenons l'exemple d'un RTOS ayant 3 tâches dont deux de même priorité basse et une autre de priorité haute. Les deux tâches de priorités basses ne demandent jamais à s'arrêter. La tâche de priorité haute fait ce qu'elle a à faire puis demande à s'arrêter.

Voici le diagramme de fonctionnement de ce qu'il se passe dans le cas présenté précédemment :



Voici deux exemples concrets d'utilisation d'un RTOS :

- Nous avons trois tâches (récupérer des données, les traiter et envoyer des instructions). Il faut que le RTOS soit capable de passer d'une tâche à une autre de manière logique et rapide.
- Nous avons deux tâches ayant la même priorité. L'une reçoit des données, l'autre les traite très rapidement puis s'arrête pendant un certain temps. La tâche récupérant les données s'exécute pendant le temps SYS\_TICK (temps entre deux interruptions système) puis laisse la place à la tâche qui va traiter les données puis s'arrêter directement. Nous revenons alors à la tâche de réception pour refaire le cycle à l'infini.

**Ainsi, un RTOS est dit « multitâches » du fait qu'il peut gérer différentes tâches en parallèle.**

Notez également que les différentes tâches peuvent communiquer entre elles via des « queues ».

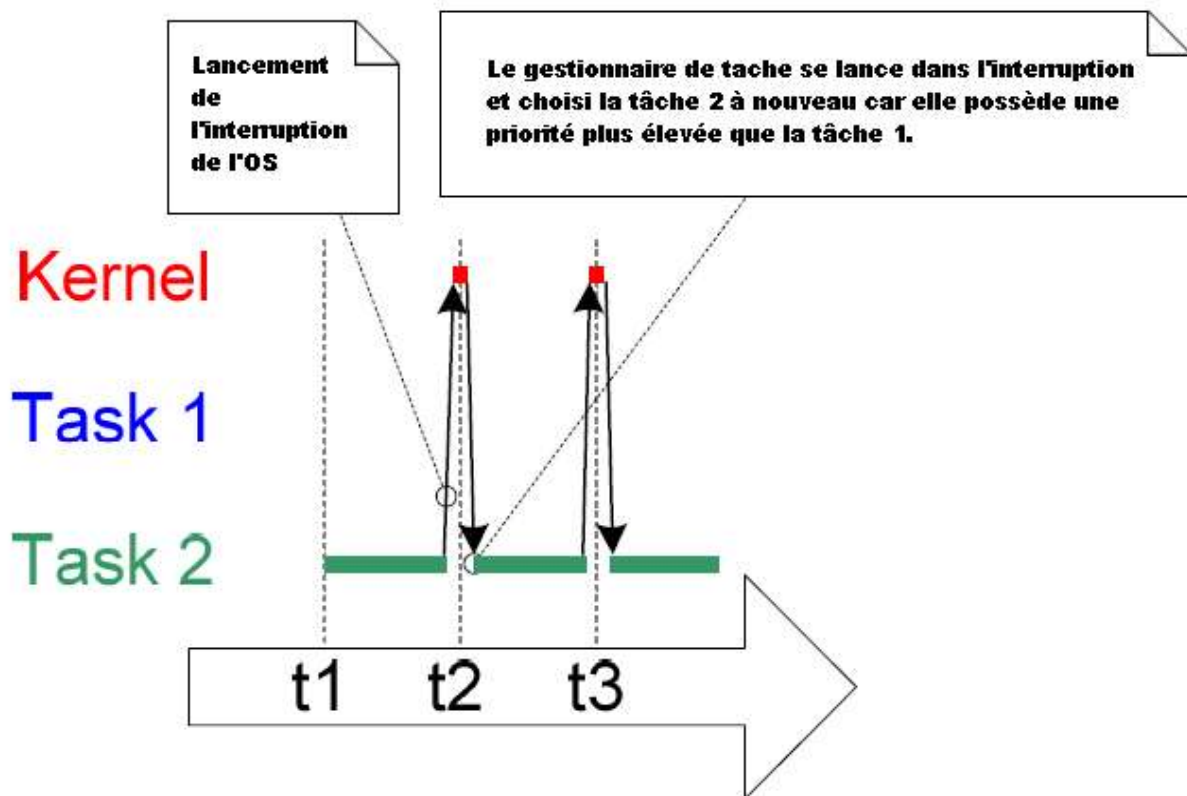
Il est également possible d'obliger le RTOS à rester dans une tâche de basse priorité (il faut cependant faire attention en utilisant cela).

Avant de finir ma présentation sur les RTOS, il faut se rendre compte que les RTOS engendrent certains problèmes.

Un de ces nombreux problèmes est le phénomène de « **famine** » :

Imaginons que nous avons 2 tâches de priorités différentes. Si la tâche de priorité élevée ne s'arrête jamais, alors la tâche de priorité basse ne se lancera jamais.

Voici l'illustration de ce cas (« Task 1 » de priorité inférieure à « Task 2 ») :



*Diagramme illustrant le problème de famine*

**Vous connaissez maintenant les bases du fonctionnement d'un RTOS.**

#### IV) Les outils mis à ma disposition

- Le logiciel **Eclipse IDE Indigo& Kepler** (environnement de développement)



- Une application en JAVA créée par UXP permettant d'envoyer le fichier compilé par Eclipse sur la carte NanOpral. Cette application fait également office de débogueur.

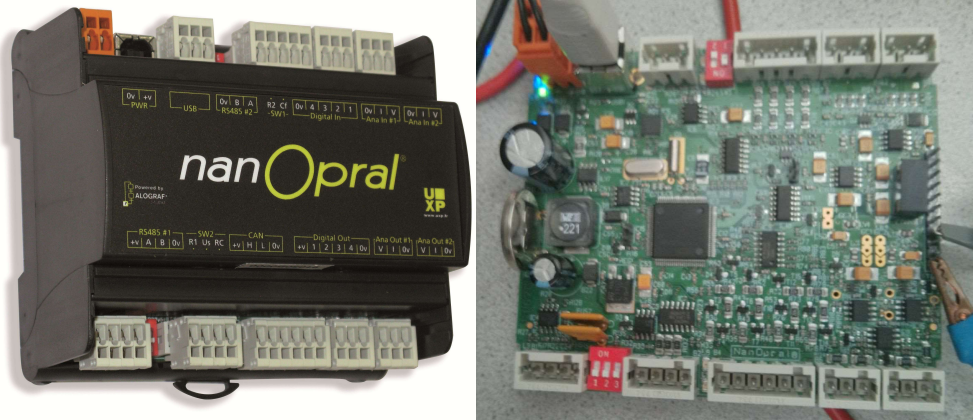


Cette application est très pratique pour visualiser rapidement si notre programme est juste ou non.

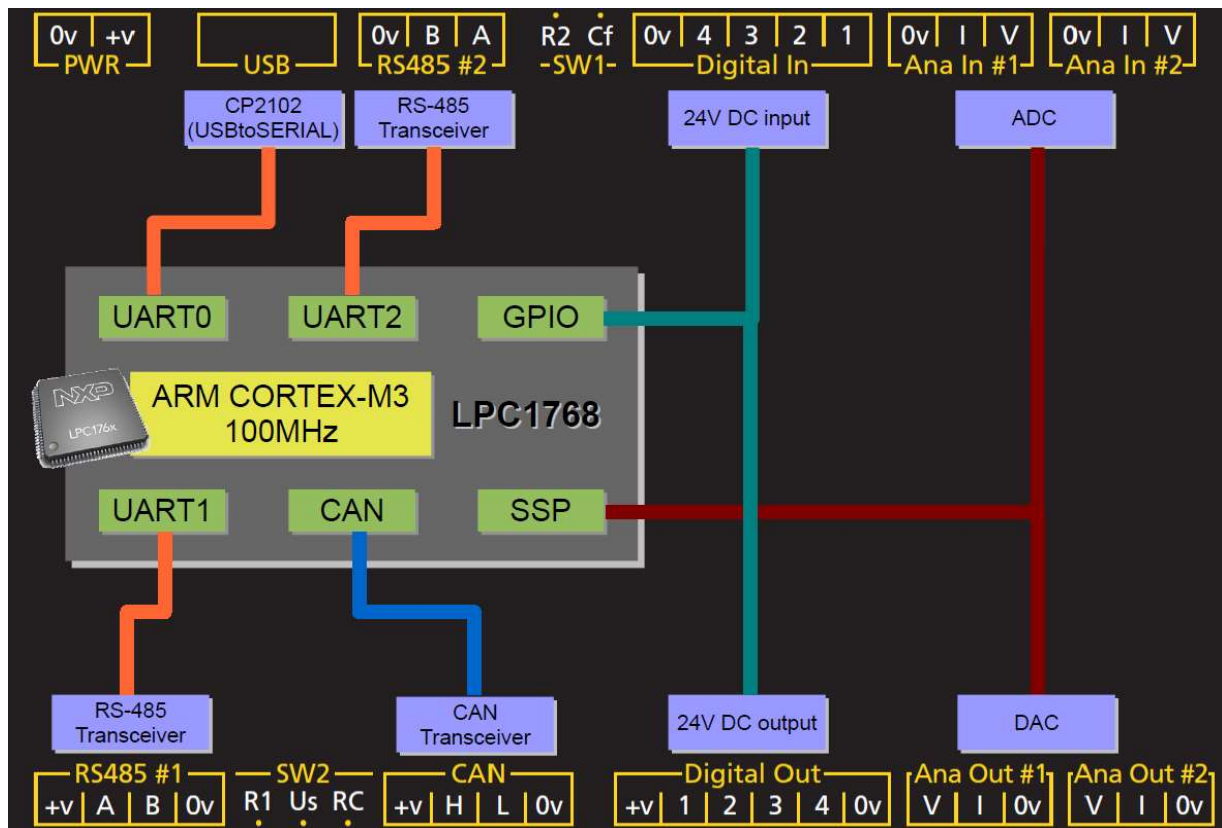




- o Une **carte NanOpral** (produit d'UXP) contenant un microcontrôleur NXP LPC 1768 à base de Cortex-M3. C'est sur ce produit que j'ai travaillé durant ces deux mois de stage.



*Le NanOpral sous sa forme commerciale (à gauche) et sans la coque (à droite)*



*Description technique du NanOpral*

Cette carte électronique permet de gérer des communications (CAN, RS-485, USB), elle permet également de traiter et d'envoyer des informations analogiques ou numériques via les différents ports de communication. Enfin, cette carte permet d'effectuer des calculs pour les différents composants connectés.

Vous trouverez une documentation plus détaillée de ce produit en annexe.

## V) Mon rôle

### 1)Présentation générale avec diagramme de Gantt

N'ayant jamais étudié les RTOS, la notion de système en temps réel (RTOS) m'était inconnue. J'ai donc étudié ces systèmes afin d'en comprendre leurs fonctionnements, leurs avantages, inconvénients,...

J'ai tout particulièrement étudié les systèmes en temps réels « CoOS » ainsi que « FreeRTOS ». Il a fallu étudier la documentation technique ainsi que comprendre les enjeux de l'implantation de ces RTOS.

Après avoir étudié ces RTOS, il m'a fallu faire des tests sur ces OS afin de les comparer pour en choisir un. Un point important est qu'il a fallu augmenter la réactivité des systèmes alors que cela n'était pas prévu par ceux-ci. Il a donc fallu ruser et quantifier les pertes de performances associées.

Après deux semaines de tests, FreeRTOS s'est imposé tout naturellement.

Il était à présent nécessaire de créer une librairie/driver compatible avec FreeRTOS pour le calculateur à base de LPC1768 (vous trouverez plus de détails sur cette « compatibilité » dans les explications détaillées). Un point important est qu'il fallait créer une librairie/driver de A à Z (c'est-à-dire sans les librairies de base de NXP, fabricant du LPC1768)). Cette librairie/driver devait servir de base pour les clients voulant utiliser le calculateur. Ainsi, les fonctions devaient être à la fois simple et performantes. Il a donc été nécessaire de créer des fonctions du type « initialisation », « envoie » et « réception » pour le CAN/USB/RS485#1/RS485#2/Entrées-Sorties numériques et analogiques, des vérifications d'erreurs/overflow pour les buffers associés, gérer correctement le « temps système », faciliter la création future d'interruptions, ...

Notons également que la librairie devait être livrée avec un document explicatif complet (explication de base de la librairie ainsi qu'une explication détaillée de chaque fonction). Il a donc été nécessaire que j'apprenne le « langage Javadoc » qui permet à la fois de commenter proprement son code et de générer une documentation complète de la librairie/driver.

Ainsi, pour résumer, durant ce stage, j'ai testé deux RTOS afin de sélectionner le plus performant pour le calculateur et j'ai créé intégralement la librairie de ce calculateur qui devait être compatible avec FreeRTOS.

Voici un diagramme de Gantt décrivant globalement le travail effectué durant ce stage:



## 2)Présentation détaillée du contenu du stage

### ○ Découverte du NanOpral (LPC1768)

Lors des premiers jours du stage, j'ai réalisé des tests sur la carte de développement LPC1766-STK (fichier à compiler avec un hello world sur l'écran de la carte de développement ainsi que quelques tests sur les pins) afin de me familiariser avec ce type de microprocesseur.

J'ai ensuite étudié en détail le fonctionnement électronique du calculateur d'automatisme NanOpral qui sera par la suite le microcontrôleur que j'utiliserais.

### ○ Intégration basique de deux RTOS (FreeRTOS et CoOS)

Suite à l'étude du NanOpral, il a fallu que je me familiarise avec la notion de RTOS que je vous ai présenté plus haut (cf p11). Cela a pris quelques jours car certaines notions ne sont pas forcément évidente à appréhender.

Après avoir compris le fonctionnement général des RTOS, j'ai étudié en détail deux d'entre eux (FreeRTOS et CoOS) en lisant leurs documentations utilisateurs ainsi que des exemples d'applications.

C'est ainsi que j'ai appris les notions de tâches, priorités, préemption, mise d'éléments dans la file, passage d'une tâche à une autre par tick système, par attente de tâche ou encore par mise d'élément dans la file, ...

Ensuite, j'ai intégré ces deux RTOS sur la carte NanOpral afin de faire des tests basiques comme le clignotement d'une led par le biais de tâches afin de valider le fonctionnement des RTOS sur la carte NanOpral.

### ○ Test de performance des RTOS

Afin de ne garder que le meilleur de ces deux RTOS pour la carte NanOpral, il a fallu rechercher les possibilités de chacun ainsi que leurs avantages/inconvénients.

Pour cela il a fallu faire des recherches sur internet ainsi que dans le code source lui-même. Cette étape m'a permis de comprendre en profondeur le fonctionnement réel des RTOS (implémentation des interruptions propres au RTOS, les ticks, ...).

Après cette étape « bibliographique », il a fallu passer à la réalisation des différents tests pour comparer les performances des deux RTOS tout en augmentant leurs réactivités. Ce besoin de réactivité vient du fait que les applications d'UXP doivent le plus souvent être très réactives, mais ne durent pas longtemps (réception, envoi, vérification, ...).

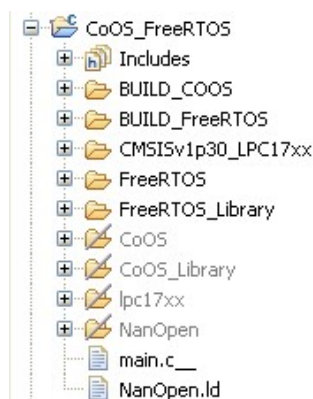
J'ai donc pour cela augmenté la fréquence des ticks au-delà des fréquences conseillées par les concepteurs des RTOS et j'en ai observé les conséquences. Pour vérifier la véracité de mes

observations, j'ai également regardé le fichier assembleur afin de compter le nombre d'instructions appelées (pour les comparer aux observations sur l'oscilloscope). Le but était donc de trouver le bon compromis réactivité (temps entre deux lancements d'une même tâche) et la performance en terme de temps de calcul.

J'ai ensuite fait des tests d'autres temps caractéristiques (création d'une queue/sémaphore, mise en place d'un élément dans la queue, demande de sémaphore, ...).

À la fin des tests, il est apparu évident que FreeRTOS était bien plus performant que CoOS pour NanOpral.

Vous trouverez en annexe D un rapport complet de cette comparaison entre FreeRTOS et CoOS.



*Architecture du projet sous Eclipse permettant de comparer les deux RTOS :  
Les deux RTOS utilisant les mêmes fichiers, la comparaison est plus fiable*

- **Création de la nouvelle librairie ainsi que de l'environnement de développement (Eclipse Kepler)**

Après avoir fait les différents tests sur les RTOS, il a fallu créer intégralement une librairie « NanOpen 2.0 » n'utilisant pas les librairies de base LPC17xx. En effet, la librairie constructeur contient de nombreuses erreurs qui rendait les applications non fiables.

Ainsi, le fait de refaire complètement la librairie élimine ce problème.

Cette étape demande donc énormément de travail et ne sera fini qu'à la fin du stage.

Il a fallu que j'apprenne en profondeur le langage assembleur et que je réfléchisse à une librairie pouvant être utilisée avec et sans RTOS.

Cette nouvelle librairie doit être simple d'utilisation pour l'utilisateur (INIT/READ/WRITE pour l'USB/RS485/CAN, ...).

De nombreux impératifs viennent s'ajouter à la création de cette librairie :

- Utiliser un nouveau compilateur GCC, l'ancien étant obsolète : <https://launchpad.net/gcc-arm-embedded>
- Tester et utiliser la nouvelle version d'Eclipse (Kepler)
- Créer un Makefile et un Load file afin de pouvoir lancer la compilation sans Eclipse en cas de problème. Cela a pour but de proposer aux clients un environnement de travail

plus performant et polyvalent que l'ancien.

Voici les principaux périphériques intégrés dans cette nouvelle librairie :

- Timer (permet de connaître une durée entre deux instants avec une précision en us/ms/sec/heure sur 400 ans).
- UART RS485/USB/CAN (permet de transmettre et recevoir des informations/instructions par le biais de ces bus réseaux à la vitesse que l'on veut avec des buffers et des systèmes de détection d'overflow, cette partie fut la plus longue à concevoir).
- LEDs (permet de visualiser si le programme fonctionne correctement ou non).
- Entrées-Sorties numériques et analogiques (permet de récupérer des données provenant de capteurs ou d'autres composants électroniques).
- Pin de débogage (permet au personnel d'UXP de faire des tests avec un oscilloscope afin de faire un débogage précis).

Un problème s'est posé avec le nouveau compilateur : Celui-ci avait un comportement étrange. Il s'est avéré qu'il ne s'agissait en fait que d'un oubli de ma part (je ne connaissais pas la notion de « volatile » qui permet d'empêcher une optimisation d'une variable par le compilateur).

Voici la macro utilisée ainsi que ce que cela donne dans le code source afin d'éviter des erreurs de compilation :

```
#define p_ISERO 0xE000E100 //Adresse du registre ISERO
#define ADDRESS(x) (*(volatile uint32_t *) (x)) //Macro permettant de pointer vers le registre x

//Dans le code source :
ADDRESS(p_ISERO) |= (1<<2) + (1 << 1) ; //Mettre un 1 décalé de 2 et de 1 dans le registre
ISERO (registre ISERO : xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx11x)
ADDRESS(p_ISERO) &= (1<<4) + (1 << 5) ; //Mettre un zéro décalé de 4 et de 5 dans le registre
ISERO (registre ISERO : xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx00x11x)
```

Un autre problème a été rencontré dans cette partie : Mon maître de stage s'est absenté durant deux semaines. Cela m'a un peu ralenti dans la conception de la librairie. Cela m'a en contrepartie permis de travailler de manière entièrement autonome.

La création de la librairie du bus CAN fut l'une des étapes les plus longues car ce bus est tout particulièrement compliqué à tester sans avoir au préalable complétement conçu l'envoi et la réception.

Notons également que c'est à ce moment là que mon record personnel de nombre de lignes de code en une journée a été battu : 800 lignes de code en une journée !

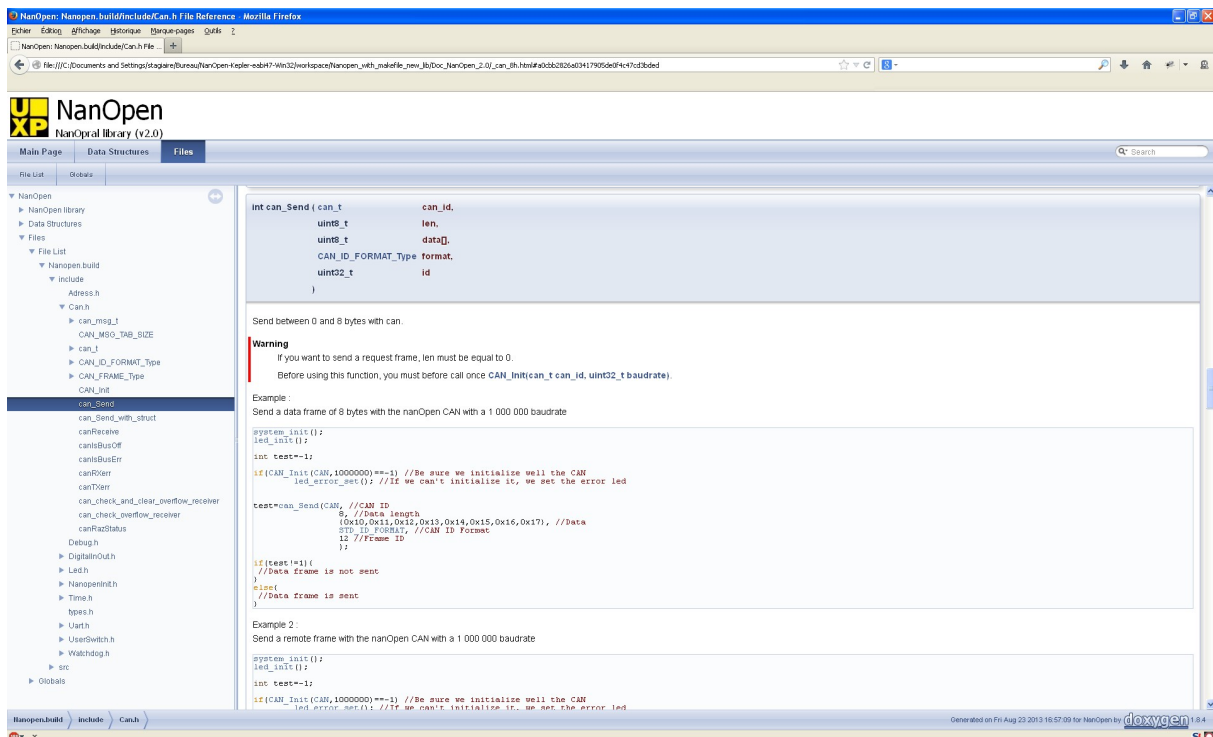
## ○ Documentation Javadoc (Doxygen)

Afin de rendre le code compréhensible par les clients d'UXP, il est nécessaire de bien commenter la librairie.

Pour cela, il existe un langage de documentation qui se nomme Javadoc/Doxygen.

Cela permet, en plus de la clarté du code, de générer une documentation précise des possibilités qu'offre la librairie.

Voici un petit exemple du résultat :



Vous trouverez un guide d'utilisation de Doxygen en annexe E ainsi que d'autres exemples en annexe G.

- **Faire coexister la nouvelle librairie avec FreeRTOS**

Le but de cette partie est de protéger les parties critiques de la librairie (Uart-tx/Uart-rx/Can-rx-tx/...). Pour cela il a fallu utiliser des mutex.

Voici un exemple où un mutex est indispensable :

Ce que l'on doit envoyer :

tâche 1 :

00000000000000000000\n"

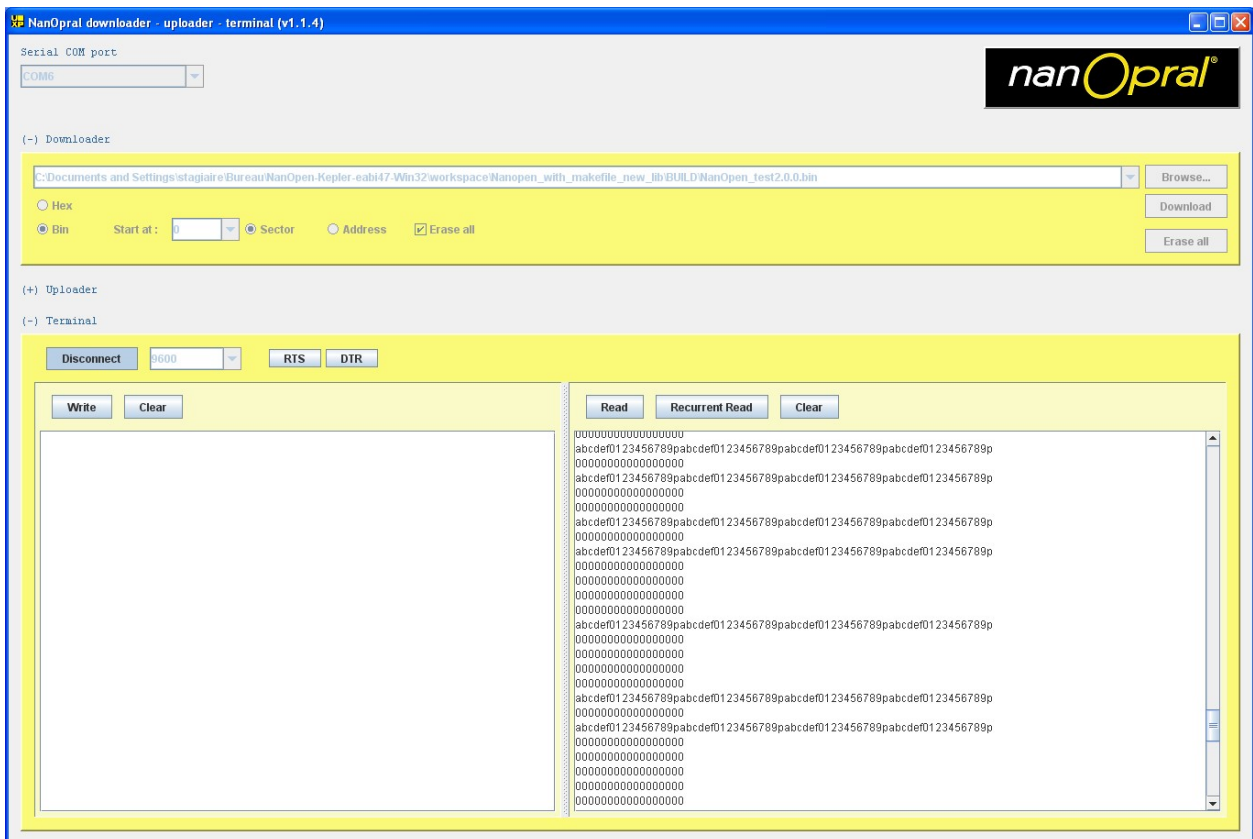
tâche 2 :

abcdef0123456789pabcdef0123456789pabcdef0123456789pabcdef0123456789p\n

En utilisant un mutex dans la librairie, cela oblige le système à envoyer les données complètes.

Ainsi, sans le mutex, la tâche 1 envoie ses données en plein milieu de l'envoi de la tâche 2 (des 00000000000000000000 s'insèrent dans les "abcdef0123456789p").

Cela prouve l'utilité des mutex et qu'ils fonctionnent bien dans la librairie créée.



*Envoie de données avec mutex*





# LES APPORTS PERSONNELS DE CE STAGE

---

## I) Enrichissement personnel grâce au stage

D'un point de vue professionnel, ce stage m'a permis de découvrir de nouvelles **méthodes de travail**, de **travailler en équipe et surtout en autonomie**, d'**approfondir mes connaissances dans le domaine de la programmation bas niveau** et donc des **systèmes embarqués** qui me tient tout particulièrement à cœur.

J'ai particulièrement apprécié la communication et le dynamisme du groupe.

J'ai également apprécié le côté **convivial** de ce stage. En effet, toute l'équipe d'UXP était ouverte et prête à aider en cas de problème.

Il est cependant vrai que mon maître de stage, ayant une vue plus globale sur le projet, était d'une aide précieuse pour pouvoir accomplir mon stage à terme.

Enfin ce stage m'a permis de **découvrir comment créer et élaborer des projets informatiques proprement pour être à la fois efficace et permettre aux futurs développeurs/clients une bonne compréhension du travail déjà accompli**.

Pour ce qui est de l'enrichissement de mes connaissances personnelles, voici ce que le stage m'a apporté :

- **Familiarisation avec un nouveau environnement de développement C**  
Compréhension poussée (Eclipse Indigo et Kepler)
- Approfondissement et application de mes connaissances en **C bas niveau ainsi qu'en assembleur**
- Apprentissage du « langage » **Javadoc** (et du logiciel **Doxygen**)
- Création de **Makefile** et de **Load files**
- Compréhension poussée du fonctionnement des **UARTs, bus CAN, modbus, RS485, DMA, DAC, ADC, Timers, ...**

## II) Enrichissement personnel grâce à l'ouverture d'esprit de l'équipe d'UXP

Une des expériences intéressantes fut de pouvoir discuter des projets personnels de chacun. En effet, UXP est, en partie, constituée de personnes ayant une philosophie se rapprochant du « NERD », c'est-à-dire des personnes passionnées par les nouvelles technologies.

Cette ouverture d'esprit m'a entre autre motivé pour **concevoir ma première application sous Android** (une intelligence artificielle sous forme d'un petit robot répondant aux questions de l'utilisateur et pouvant renvoyer sur des articles Wikipédia si l'utilisateur a une question trop spécifique).

Grâce à l'équipe d'UXP, je me suis également motivé pour essayer la **programmation sur Arduino** afin de la comparer avec celle sous **stm32**.

Certains membres de l'équipe font également de la **course à pied**. Cela m'a donc motivé à en faire avec eux. Je ne suis pas devenu un athlète dans ce domaine pour autant ...

# CONCLUSION

---

Ce stage à UXP fut une expérience enrichissante à la fois d'un point de vue professionnel-méthodes de travail et social.

Au niveau personnel, ce stage était l'occasion rêvée de confirmer mon choix de poursuite d'étude dans les systèmes embarqués ainsi que l'occasion d'étudier ce domaine avant d'avoir les cours théoriques.

Ce stage m'a également fait prendre conscience que mon niveau en programmation bas niveau n'était pas si élevé que je le pensais. En effet, les connaissances apprises à l'ENSEA sont certes de qualité mais manquent cruellement d'approfondissement (sans doute par manque de temps). Cela ne m'a cependant pas empêché de progresser très rapidement et de rendre le projet final à temps.

Voici les différents problèmes rencontrés durant ce stage :

- Problèmes techniques : Fréquence du CPU, premier test des RTOS non cohérent, optimisation O2 du nouveau compilateur.
- Manque de connaissances en C bas niveau ainsi que sur les ports USB/RS485/CAN : Cela a un peu ralenti la conception de la librairie car il a été nécessaire que je me forme dans ce domaine. Cela a également entraîné de petits problèmes lors de la conception de la librairie (mauvaise configuration de certains registres et une mauvaise compréhension du mode « lecture-écriture » d'un registre en particulier).
- Autonomie : Le fait que mon maître de stage n'était pas présent pendant deux semaines n'a pas réellement été un problème même si cela m'a un peu ralenti dans la conception de la librairie.

Tous les problèmes précédents ont été résolus sans réelles difficultés et m'ont même été bénéfiques.

# ANNEXE A : Exemple en C de gestion multitâches sur FreeRTOS

---

Le code suivant permet de faire clignoter deux leds toutes les secondes (tâches périodiques) ainsi que de recevoir des données via l'USB, le RS485#1 et 2 ainsi que via le bus CAN :

```
/* Task Handles (used to identify tasks) : */

static xTaskHandle xTaskHandle1;
static xTaskHandle xTaskHandle2;
static xTaskHandle xTaskHandle3;

/* Tasks : */

void task_led_error(void* p) // This task starts once every 1 sec to
change the state of the LED ERROR (periodic task)
{
    //Put here definition of the necessary variables for
task_led_error (no blocking function) :
    unsigned long long nombre_lederror_toggle=0;

    while(1)
    {
        // Put here what should periodically be run in
task_led_error :

        led_error_toggle(); // Changes the state of the ERROR LED
on each call of the task
        nombre_lederror_toggle++; // Variable increased by 1 on
each call of the task
        led_error_toggle();
        vTaskDelay(1000000/portTICK_RATE_US); //Time the task
will sleep : 1sec (1000000Us=1sec)
    }
}

void task_led_run(void* p) // This task starts once every 1 sec to
change the state of the LED RUN (periodic task)
{
    //Put here definition of the necessary variables for
task_led_run :
    unsigned long long nombre_ledrun_toggle=0;

    while(1)
    {
        // Put here what should periodically be run in
task_led_run (no blocking function) :

        led_run_toggle(); // Changes the state of the RUN LED on
each call of the task
        nombre_ledrun_toggle++; // Variable increased by 1 on each call
of the task

        vTaskDelay(1000000/portTICK_RATE_US); //Time the task will
sleep : 1sec (1000000Us=1sec)
    }
}
```

```

void rx_task(void* p){ // Receive data task

    can_msg_t pMessage;
    char bufrx[18+17+17+17+2];
    int test=-1;
    int i;

    while(1){

        test=canReceive(CAN, &pMessage);

        if(test!=-1){
            // Do something if we received a CAN message
        }

        test=uart_receive_bytes(UART_RS485_2,0,18+17+17+17+2,bufrx);
        if(test!=-1){
            // Do something if we received a RS485#2 message
        }

        test=uart_receive_bytes(UART_RS485_1,0,18+17+17+17+2,bufrx);
        if(test!=-1){
            // Do something if we received a RS485#1 message
        }
        test=uart_receive_bytes(UART_USB,0,18+17+17+17+2,bufrx);
        if(test!=-1){
            // Do something if we received a USB message
        }
    }
}

// MAIN PROGRAM
int main(){

    /* ----- INITIALIZATION ----- */
    system_init(); // Don't delete it !

    led_init(); //If you need to use red/blue led
    time_init(); //If you need to use NanOpen time (if you only use
FreeRTOS time, don't use time_init())
    Digital_in_out_init(); //If you want to use Digital outputs (if you
only need digital inputs, no need to do Digital_in_out_init())
    if(!uart_init(UART_USB,921600,NO_PARITY,1)) //If you need USB with a
921600 baudrate, without parity and with one stop bit
        while(1)
            led_error_set();

    if(!uart_init(UART_RS485_1,460800,NO_PARITY,1)) //If you need RS485#1
with a 460800 baudrate, without parity and with one stop bit
        while(1)
            led_error_set();

    if(!uart_init(UART_RS485_2,460800,NO_PARITY,1)) //If you need RS485#2
with a 460800 baudrate, without parity and with one stop bit
        while(1)
            led_error_set();

    //Pin_Debug_Init(); //Only for UXP staff !!

```

```

    if(CAN_Init(CAN,1000000)==-1)
        while(1)
            led_error_set();

    /* ----- Creating tasks & launching FreeRTOS ----- */
    #ifdef USE_FREERTOS
        xTaskCreate(task_led_run, (signed char*)"t1",
        FreeRTOS_MINIMAL_STACK_SIZE, 0, 2, &xTaskHandle1); //Task task_led_run
        (priority 2)
        xTaskCreate(task_led_error, (signed char*)"t2",
        FreeRTOS_MINIMAL_STACK_SIZE, 0, 2, &xTaskHandle2); //Task task_led_error
        (priority 2)
        xTaskCreate(uart_tx_task, (signed char*)"t3",
        FreeRTOS_MINIMAL_STACK_SIZE+100, 0, 1, &xTaskHandle3); //Task
        continuous_task (priority 1 (lowest))
        xTaskCreate(can_tx_task, (signed char*)"t3",
        FreeRTOS_MINIMAL_STACK_SIZE+100, 0, 1, &xTaskHandle3); //Task
        continuous_task (priority 1 (lowest))
        xTaskCreate(rx_task, (signed char*)"t4",
        FreeRTOS_MINIMAL_STACK_SIZE+200, 0, 1, &xTaskHandle4); //Task
        continuous_task (priority 1 (lowest))

        vTaskStartScheduler(); //Launching FreeRTOS (create all needed
        tasks before launching FreeRTOS)
    #endif

    /* ----- END ----- */

    //You should not go there if you use FreeRTOS (if you go there, maybe
    you didn't allocate enough heap (go check "Configuration.h"))
    led_error_set();
    while(1);

    return 0;
}

```

## ANNEXE B : Extrait du code de la librairie créée durant le stage

Je vous présente ici la fonction de d'envoi de bytes via les UARTS (USB/RS485#1/RS485#2). Cette fonction est protégée par des mutex suivant les différents UARTs. Elle va remplir un buffer de transmission puis envoyer un élément afin d'amorcer les interruptions d'envois de bytes.

```
int uart_send_bytes(UART_id uart,Wait_send_id wait_send, const char *
bytes_array , uint16_t bytes_array_len)
{

    //Return 0 if wrong UART or if we don't have enough space in buffer
to send data (-->overflow is not possible):
    if((!test_UART(uart)) ||
((UART_TX_buffer[uart].uart_tx_nbr_elements)+bytes_array_len)>UART_TX_BUFF
ER_SIZE) || (bytes_array_len==0))
        return (-1);

    #ifdef USE_FREERTOS //Protect the needed UART
        if((uart==UART_RS485_1) || (uart==UART_RS485_2)){
            if(xSemaphoreTake(xMutex_UART[uart],0) != pdTRUE)
                return (-1);
        }
        else if(xSemaphoreTake(xMutex_UART_TX[uart],0) != pdTRUE)
            return (-1);
    #endif

    end_uart_send[uart]=0;

    //Else we fill the buffer as asked :

    ADDRESS(uart_adress[uart].UxIER) &=~(0x02); //Stop Uart send interrupt
(THRE)

    volatile uint16_t index = 0;

    for( index = 0 ; index < bytes_array_len ; index++ )
    {
        //fill the buffer of one byte

        UART_TX_buffer[uart].uart_tx_buffer[UART_TX_buffer[uart].uart_tx_end_
in_buffer]=bytes_array[index];

        //Go to the next byte to fill

        if(UART_TX_buffer[uart].uart_tx_end_in_buffer<UART_TX_BUFFER_SIZE-1)
            UART_TX_buffer[uart].uart_tx_end_in_buffer++;
        else
            UART_TX_buffer[uart].uart_tx_end_in_buffer=0;
    }

    UART_TX_buffer[uart].uart_tx_nbr_elements+=bytes_array_len;

    //If rs485#2, we need to activate "send mode"
    if(uart==UART_RS485_2)
        ADDRESS(p_FIO0SET) = (1ul << UART2_DIR_COM_PIN);
```

```

        ADDRESS(uart_address[uart].UxIER) |=0x02; //Activate Uart send
interrupt (THRE)

        //We need to send 1 byte to begin THRE interrupts :
        ADDRESS(uart_address[uart].UxTHR)=
UART_TX_buffer[uart].uart_tx_buffer[UART_TX_buffer[uart].uart_tx_first_in_b
uffer];

        if (UART_TX_buffer[uart].uart_tx_first_in_buffer<UART_TX_BUFFER_SIZE-
1)
            UART_TX_buffer[uart].uart_tx_first_in_buffer++;
        else
            UART_TX_buffer[uart].uart_tx_first_in_buffer=0;
            UART_TX_buffer[uart].uart_tx_nbr_elements--;

#ifdef USE_FREERTOS //You must wait if you don't use FreeRTOS, you can
choose if you use it
        if(wait_send==WAIT_SEND_DATA) // Wait here that we send all bytes
#endif
            while(end_uart_send[uart]==0);

        //Return 1 if all bytes have been put in the buffer.
        return 1;
}

```



# ANNEXE C : Documentation technique du NanOpral

# **nanOpral<sup>®</sup>**

**Calculateur d'Automatisme Programmable**  
**pour le Contrôle Commande d'Equipements Industriels**  
**Applications : Energie, Embarqué, Industrie, BTP et Montagne**



*Fabrication française*

Ce calculateur d'automatisme est une nouvelle génération de matériel à faible consommation d'énergie, très compact et performant, pouvant fonctionner seul ou s'intégrer dans tout type d'architecture. Robuste, conçu aux normes de l'Industrie, ce calculateur dispose de ports de communication filaire adaptés au milieu industriel (Ports série RS485, bus CAN, protocoles Jbus, Modbus RTU et CAN Open, ...). Il se présente sous la forme d'un boîtier modulaire standard à monter sur rail DIN.

**NanOpral** est idéal pour vos applications de Contrôle et Optimisation d'Energie, Commande d'équipements et machines fixes ou embarqués, Régulations de systèmes et Asservissements de moteurs, ainsi que pour Collecter, Calculer et Transmettre les indicateurs de production pertinents, etc.

Il offre 4 entrées numériques pouvant acquérir des états de contacts, des compteurs, ... ainsi que 4 sorties numériques, 4 entrées analogiques (2 en courant, 2 en tension) et 2 sorties analogiques (courant + tension). Des extensions E/S sont possibles sur bus CAN ou RS485.

Bâti autour d'un microcontrôleur populaire standard, ce calculateur est programmé de manière très conviviale selon les normes IEC 61131 (Grafcet, Ladder, ST, FBD en langage C) et IEC 60848 (déterminisme temporel et sûreté de fonctionnement), grâce à l'atelier d'automatisme ALOGRAF Studio, logiciel largement éprouvé depuis plus de 25 ans pour les applications industrielles les plus exigeantes.



UXP SA - 12, avenue Pierre de Coubertin - ZI Percevalière - 38170 SEYSSINET  
Tel : 04 76 84 28 80 - Fax : 04 76 84 02 47 - web : [www.uxp.fr](http://www.uxp.fr)


## Calculateur d'automatisme programmable

# nanOpral®

Nom de la gamme de produits :	NanOpral
Type d'équipement :	Calculateur d'automatisme programmable
Microcontrôleur :	100 Mhz, 32 bits
Nombre et type de ports :	2 x Séries RS485 2 fils, 1 x CANbus, 1 x port système format USB
Autres caractéristiques :	Horloge temps réel sauvegardée LEDs activité ports séries et port CAN
E/S locales :	4 entrées numériques 0/40V NPN (entrée TOR ou comptage rapide ou entrée codeur incrémental) 4 sorties numériques 0/40V PNP, protégées 4 entrées analogiques 12 bits ( 2 en courant 0/20mA, 2 en tension 0/10V) 2 sorties analogiques 12 bits (sortant toutes 2 en courant 0/20mA et tension 0/10V) Connecteurs débrochables à pinces, au pas de 3,5
Protocoles :	Modbus RTU, Jbus sur les 2 ports RS485 CANopen sur port CAN
Extensions, options :	Ethernet avec protocole MODBUS TCP, Communication sans fil : ZigBee, Bluetooth, ...
Caractéristiques de Débit :	Liaisons RS485 et bus CAN configurables jusqu'à 1 Mb/s
Modes de configuration et d'administration :	PC via connecteur système USB
Logiciel de développement:	ALOGRAF STUDIO 5,2
Conditions de fonctionnement :	-10 à +70 °C
Alimentation :	10-40VDC
Consommation typique :	1 Watt
Dimensions et poids (avec boîtier) :	105*91*60 mm (6 modules d'un tableau électrique standard). Poids 150 gr.



Exemple d'intégration dans un tableau électrique standard

	<b>UXP offre des <i>Services et Solutions</i> <i>Ouvertes pour l'Automatisme</i> :</b>	<b>nanOpral® est un Produit UXP</b> distribué par :
	Marchés ENERGIE, EMBARQUE, INDUSTRIE, BTP, MONTAGNE Contrôle en production, Automatismes rapides, Régulation, Asservissements, Communication d'atelier, Réseaux de terrain, Télégestion, Serveurs Web Services embarqués, Dialogue homme- machine et Supervision, ...	

ALOGRAF, OPRAL, MICROPRAL, NANOPRAL, MODBUS, JBUS, WINDOWS, sont des marques déposées.  
Ce document n'est pas contractuel, UXP se réservant toute modification.  
Maj : Avril 2011

UXP SA - 12, avenue Pierre de Coubertin - ZI Percevalière - 38170 SEYSSINET  
Tel : 04 76 84 28 80 - Fax : 04 76 84 02 47 - web : [www.uxp.fr](http://www.uxp.fr)

## ANNEXE D : Comparaison entre FreeRTOS et CoOS

---

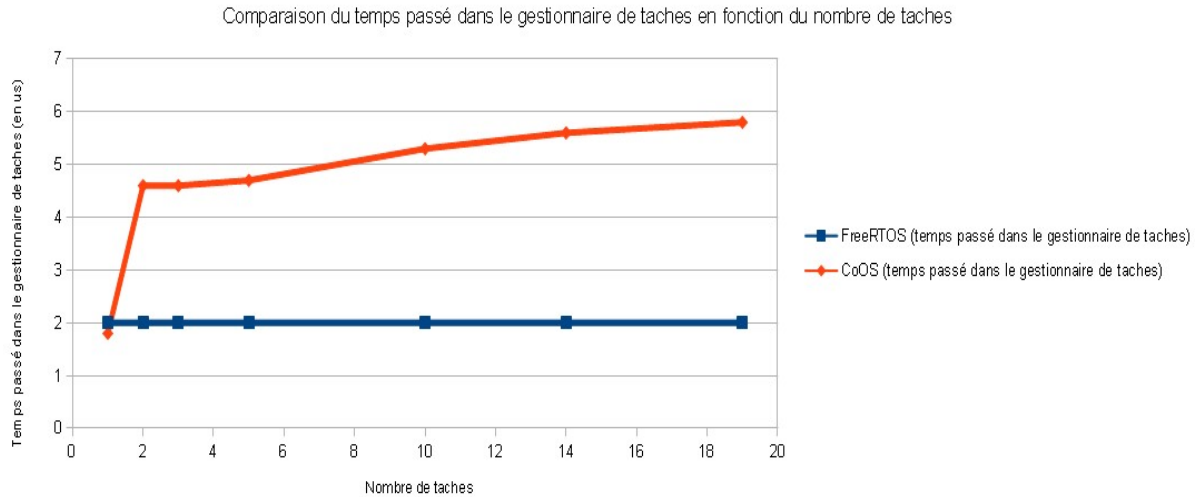


Voici une liste des principaux points communs et des différences entre les deux RTOS :

- **Politique** : Utilisation commerciale sans frais pour l'entreprise, il faut cependant mentionner que l'on utilise le RTOS, dans le cas de FreeRTOS, il faut également préciser quand on touche au noyau du système.
- **Intégration sur les différents microcontrôleurs** :
  - FreeRTOS : Possible sur de très nombreux microcontrôleurs (dont le Cortex M3)
  - CoOS : Possible uniquement sur la gamme Cortex M-x (dont le Cortex M-3)
- **Gestion des priorités** : Identique pour les 2 RTOS (la tâche ayant la plus haute priorité se lancera (notion de « preemptive priority »), cependant, si 2 tâches ont la même priorité, on change de tâche à intervalles réguliers (notion de « Round-Robin »)
- **Gestion de la mémoire** : Statique et dynamique pour les 2 RTOS
- **Echange de données tâche-tâche et tâche-interruption** :
  - FreeRTOS : Possibilité d'échange de messages (dans des queues) dans les tâches et dans les interruptions.
  - CoOS : Possibilité d'échange de messages (dans des queues et équivalents) dans les tâches et interruption vers tâche ... Il n'est cependant pas possible d'échanger d'une tâche vers une interruption (utiliser une variable statique pour cette échange peut être une solution ... quoi que peu élégant) !
- **Changer la priorité d'une tâche dans une tâche** : Possible dans les 2 RTOS
- **Possibilité de rester dans une tâche même si la tâche a une priorité inférieure** : Possible dans les 2 RTOS (cependant, les interruptions matérielles auront toujours lieu)
- **Gestion des tâches critiques** (avec un mutex/sémaphore) : Possible dans les 2 RTOS

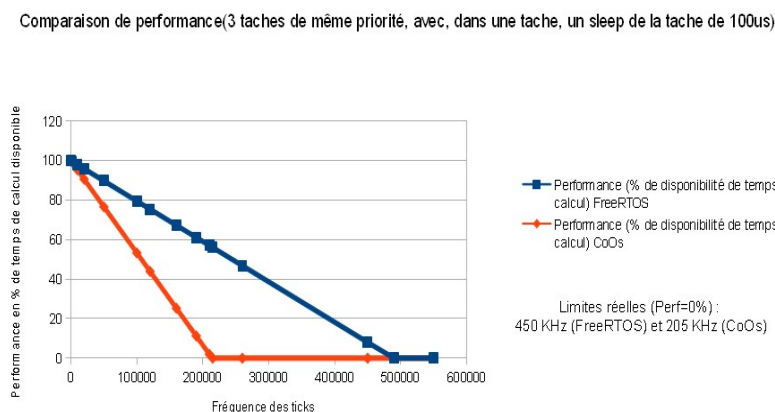
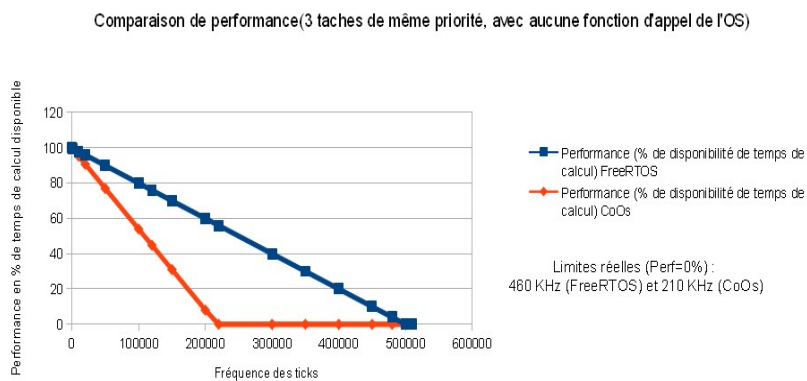
## Les performances réelles des 2 RTOS :

- **Comparaison du temps passé (en us) dans le gestionnaire de tâches en fonction du nombre de tâches :**



Le temps passé dans le gestionnaire de tâches est indépendant du nombre de tâches pour FreeRTOS alors qu'il ne l'est pas pour CoOS. FreeRTOS semble être plus performant dans ce domaine.

- **Comparaison des performances en % de disponibilité de temps de calcul pour les tâches :**



Nous remarquons que plus nous augmentons la fréquence des ticks (cad la réactivité), plus nous baissons les performances du système. Pour un système devant être très réactif mais ne demandant que peu de temps de calcul, il faut se placer à une performance de l'ordre de 30%-40% (entre 100 et 150Hz pour CoOS, entre 300 et 350KHz pour FreeRTOS). Dans le cas contraire, il faut se placer à une performance de l'ordre de 80%-100%.

**Il est ici très clair que FreeRTOS est bien plus performant que CoOS (par exemple, si on veut utiliser une fréquence système de 100KHz, FreeRTOS a 80% de performance contre 54% pour CoOS).**

– **Test d'utilisation de 20 tâches en parallèles :**

Aucun problème à signaler sur les deux RTOS (il faut cependant faire attention à la taille des ressources que l'on alloue à chaque tâche ... plus celle-ci est importante, moins on pourra mettre de tâches en parallèle).

– **Comparaison de différents temps caractéristiques :**

	FreeRTOS	CoOS
Fréquence dans les taches (en MHz, normalement 100MHz)	100MHz	100MHz
Temps de création d'une tache (en us)	3,4	3,3
Temps de suppression d'une tache (en us)	2	2,5
Temps de création d'une file (en us)	8	3,3
Temps de suppression d'une file (en us)	5	2,5
Changement de tache par tick système (3 taches en parrallèles, action automatique)	2	4,6
Changement de tache par mise en place d'un element dans la file (en us)	7	7,1
Changement de tache par mise en place d'une attente (en us)	4,4	7
Changement de tache par mise en place d'un element dans une « mailbox » (càd un élément) (en us)	7	6,3
(remarque : une mailbox sous FreeRTOS est une file avec un element)		

Le temps de création/suppression des tâches et des files n'est présent qu'à titre indicatif car ils n'ont pas une incidence majeure sur les performance du système.

Nous pouvons remarquer que les temps caractéristiques de changement de tâches sont assez similaires entre les deux RTOS. Il faut cependant noter que le temps de passage d'une tâche à une autre par un tick système est bien plus faible sur FreeRTOS que sur CoOS (cf premier test)

Notons que l'ajout de fonctions systèmes (attente, mise en place d'un élément dans une queue ...) utilise un temps de calcul non négligeable. Les plus gourmand étant les fonctions pour changer de tâche de manière régulière (exemple : 2 taches de même priorité dont une contenant une attente , alors, il y aura une baisse de 10% de performance si la fréquence des ticks est la même que la fréquence d'une attente (une fois sur deux, on va lancer la fonction d'attente au lieu de lancer directement l'interruption système); Si cependant, la fréquence des ticks est 10 fois supérieure à celle de l'attente, alors la baisse de performance ne sera plus que de 2% (la fonction attente se lancera une fois sur 20)).

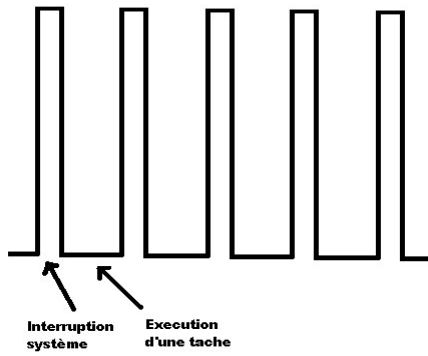
Il faut donc faire très attention à ce détail non négligeable qui peut entrainer à un disfonctionnement du système (0% de performance).

**En résumé : il faut que les temps caractéristiques des différentes fonctions systèmes permettant de changer de tâche soient appelées à des fréquences plus faibles que la fréquence des ticks système.**

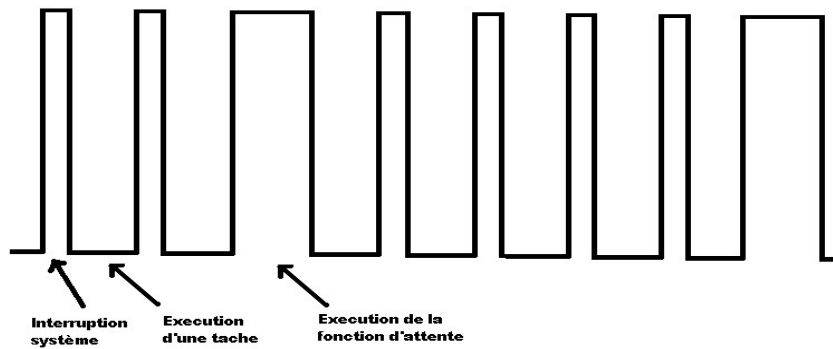
## Méthodes pour déterminer les performances des RTOS :

- **Temps passé (en us) dans le gestionnaire de tâches en fonction du nombre de tâches :**
  - Editer les différentes constantes pour pouvoir utiliser le bon nombre de tâches (le nombre de tâche dans CoOS, la place mémoire totale dans FreeRTOS)
  - Ajout des tâches ainsi que toutes les variables correspondantes (id des tâches, priorités (identiques), pointeurs vers les tâches)
  - Création de deux fonctions permettant soit de mettre à l'état bas un pin du uP (`prvValue_to_LOW_7`), soit de le mettre à l'état haut (`prvValue_to_HIGH_7`).
  - Dans les tâches, mettre une boucle infinie contenant uniquement `prvValue_to_LOW_7`
  - Placer `prvValue_to_HIGH_7` à l'entrée de l'interruption système.
  - Mesurer le temps passé à l'état haut (temps perdu) à l'aide d'un oscilloscope
  - Changer le nombre de tâches pour regarder le nouveau temps perdu
  
- **Performance en % de disponibilité de temps de calcul pour les tâches :**
  - Utiliser 3 tâches (en configurant les paramètres correspondants, comme pour le test précédent)
  - Placer de la même façon que le test précédent les fonctions `prvValue_to_LOW_7` et `prvValue_to_HIGH_7`
  - Editer la fréquence (en supprimant les restrictions que l'OS impose ainsi qu'en faisant attention de ne pas utiliser les constantes devenues obsolètes (`portTICK_RATE_MS` pour FreeRTOS) et créer l'équivalent (`portTICK_RATE_US`))
  - Visualiser avec un oscilloscope le temps passé à l'état haut (temps perdu). En déduire un pourcentage (regarder par exemple le temps perdu sur 100us, le pourcentage de disponibilité est alors  $100(\text{en us}) - \text{temps\_perdu}(\text{en us})$ ).
  - Changer la fréquence pour visualiser le nouveau temps perdu.

Pour vérifier que ce test est bien cohérent, j'ai mis une fonction toggle dans chacune de tâches (et rien au début de l'interruption système). En faisant varier la fréquence, on remarque bien que la fonction toggle ne fonctionne pas à certains instants (plus la fréquence est élevée, moins cette fonction fonctionne). Cette vérification aux fréquences extrêmes me prouve que mon raisonnement est correct.



*Test de disponibilité de calcul avec uniquement les interruptions systèmes*



*Test de disponibilité de calcul avec les interruptions systèmes et une attente pour une tâche*

– **Différents temps caractéristiques :**

Pour trouver les temps correspondant, le plus simple est de placer la fonction `prvValue_to_HIGH_7` avant la fonction appelée puis mettre des `prvValue_to_LOW_7` dans les différentes tâches.

Ainsi, on visualise un état haut à l'oscilloscope correspondant au temps que la fonction à utiliser.

## **Conclusion :**

Les deux RTOS sont performant et ont des caractéristiques très similaires, cependant, voici leurs points faibles :

- Points faibles de FreeRTOS : Il faut prévenir l'éditeur quand on édite le noyau du système, pas de flag à proprement parler dans FreeRTOS (problème très facile à résoudre en utilisant une queue binaire de 1 élément)
- Points faibles de CoOS : Impossible d'envoyer des données d'une tâche vers une interruption via une queue, interruptions matérielles plus lentes que FreeRTOS (donc moins performant).

Voici les points importants à ne pas oublier quand on utilise un RTOS à une fréquence non conventionnelle :

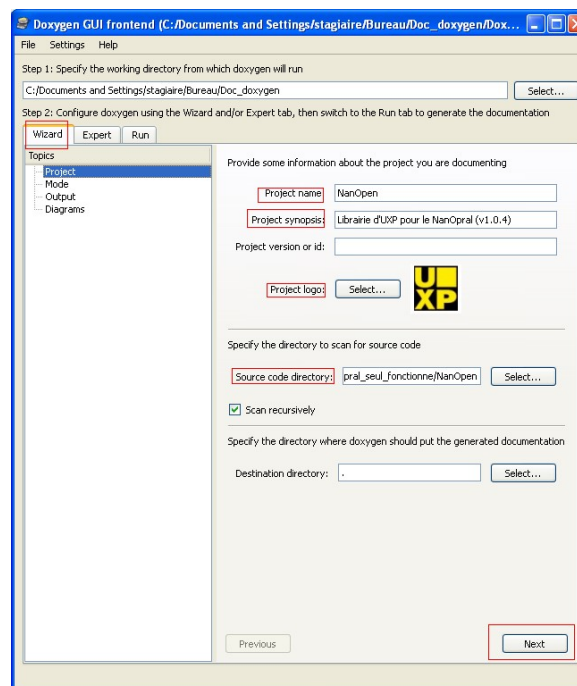
- Etre conscient qu'augmenter la fréquence des ticks (augmentation de la réactivité) fait baisser drastiquement les performances (cf courbe de comparaison des performances pour choisir le bon compromis)
- Réfléchir au temps d'appel des fonctions systèmes (attente, mise en place d'un élément dans une file pour changer de tâche, ...) pour éviter de se retrouver avec une performance désastreuse.
- Faire attention aux constantes que l'on utilise pour créer des attentes par exemple (cf méthodes de test)



# ANNEXE E : Utilisation de Doxygen

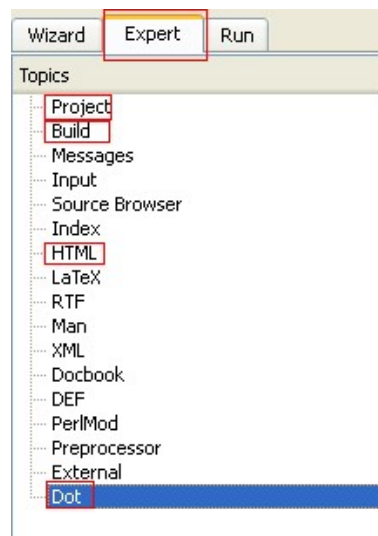
## I) Installation et configuration :

- Logiciels utilisés :
  - Doxygen (génère les documents sans les graphes)
  - Graphviz (génère les graphes, ce logiciel doit être appelé dans Doxygen)
- Installation de Doxygen :
  - Téléchargement via le site <http://www.stack.nl/~dimitri/doxygen/download.html>
  - Installation (suivre les étapes)
  - Configuration :
    - Onglet Wizard : Configuration rapide de Doxygen
      - Nom du projet
      - Description
      - Logo
      - Répertoire du code à utiliser
      - Langage de programmation utilisé
      - Type de diagrammes à utiliser : Il faut cocher toutes les cases (cela ne suffit pas pour afficher les graphes)



Doxygen → Wizard

- Onglet Expert :
  - Project :
    - OUTPUT\_LANGUAGE : French (langue)
    - FULL\_PATH\_NAME : coché
    - STRIP\_FROM\_PATH : Mettre le chemin vers la librairie
  - Build :
    - EXTRACT\_STATIC : coché
    - SHOW\_INCLUDE\_FILES : décoché (car les graphes affichent déjà les inclusions de fichiers).
  - HTML :



- SEARCHENGINE : coché

- Installation de Graphviz :

- Téléchargement : <http://www.graphviz.org/Download..php>
- Installation : Il faut copier le dossier téléchargé (non compressé) dans un endroit connu (par exemple C:\Program Files\Graphviz\)
- Configuration de Graphviz DANS Doxygen : Aller dans Doxygen → Expert → Dot → DOT\_PATH → Mettre le chemin jusqu'au dossier bin de Graphviz (exemple : C:/Program Files/Graphviz/release/bin)

- Lancer la création de la documentation :



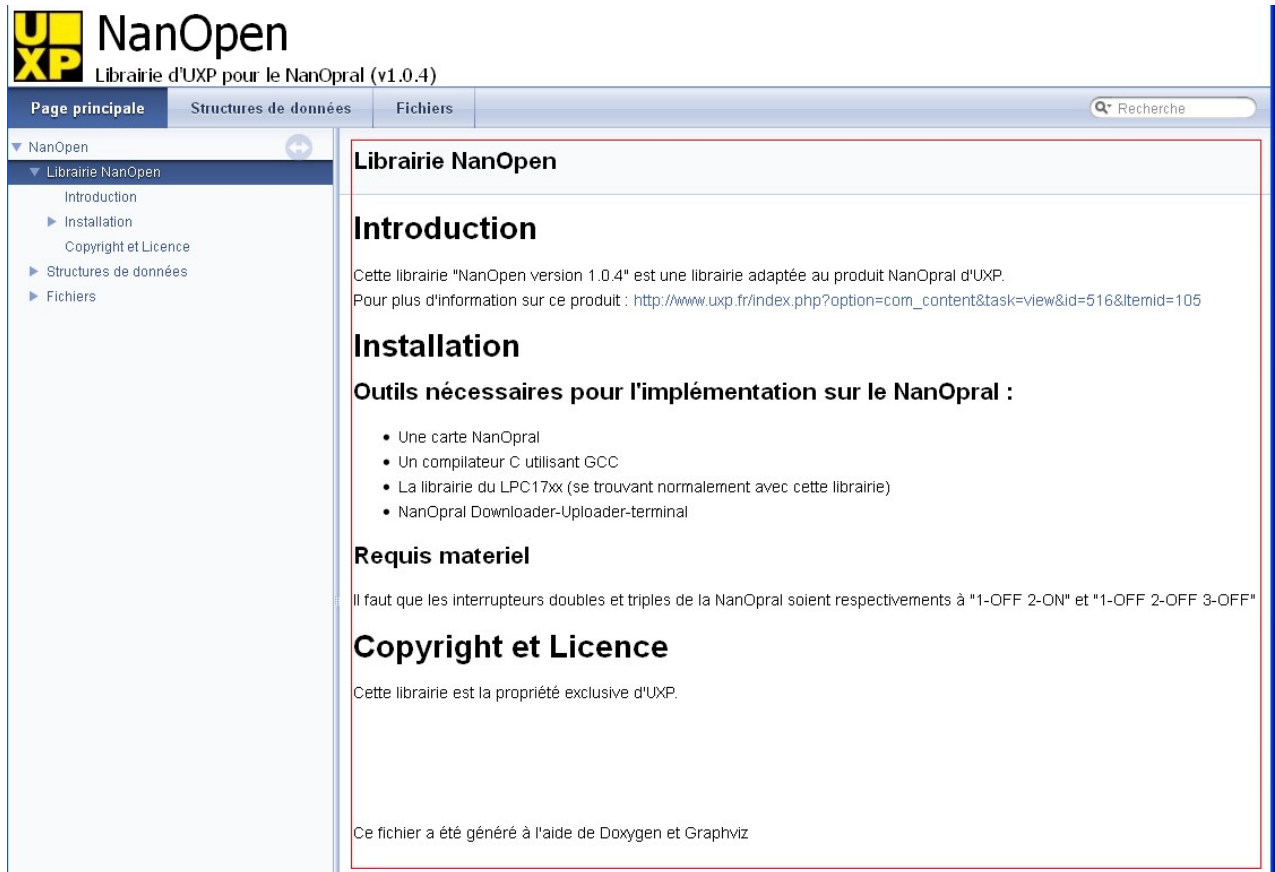
*Onglet Run → Run Doxygen*

- Ouverture/Enregistrement projet Doxygen :

Pour éviter de tout reconfigurer à chaque lancement de Doxygen, il est possible d'enregistrer/ouvrir un projet (File→Save As et File→Open)

– **Commentaires dans les fichiers .h et .c :**

1) Page principale



*Page d'introduction de la documentation Doxygen*

Pour inclure cette page principale dans la documentation, il faut ajouter le code suivant dans n'importe quel fichier .c ou .h :

```
/*! @mainpage Nom de la librairie ici
 *
 * @section intro_sec Introduction
 *
 * Mettre l'introduction ici (pour faire un saut de ligne, utiliser <BR>)
 *
 * \section Nom de la section
 * Mettre le contenu de la section ici
 * \subsection nomsubsection Nom de la subsection
 * Mettre le contenu de la sous-section ici
 *
 * \section Nom de la section
 * Mettre le contenu de la section ici
 * \subsection nomsubsection Nom de la subsection
 * Mettre le contenu de la sous-section ici
 * ... */
```

– Avant de commenter un fichier (obligatoire) :

```
/**
 * @file UART.c
 *
 * @brief Commentaire rapide sur le fichier UART.c
 * @details
 * Commentaire détaillé
 * sur plusieurs
 * lignes !
 * @author Nom de l'auteur du fichier
 * @date Mettre la date de la dernière édition du fichier ici
 * @version 1.0.4 (numéro de la version)
 */
```

– Commenter une fonction :

```
/**
 * @fn int nomfonction(char monparametre)
 * @brief Commentaire rapide sur cette fonction
 * @warning Ceci est une balise pour attirer l'attention (encadré rouge)
 * @param monparametre – Description du paramètre
 * @param monparametre2 – Description du 2ème paramètre
 * @return "0" – Mettre ici ce que signifie un retour de "0"
 * @return "1" - Mettre ici ce que signifie un retour de "1"
 */
int nomfonction(char monparametre, char monparametre2)
{
    (...)
    return 1;
    return 0;
}
```

– Commenter une variable (#define, variable globale, ...):

```
char ma_variable; /*!<Mon commentaire sur la variable ici*/

#define UART_H_ /*!<Mon commentaire sur la variable ici*/
```

5) Commenter une structure :

```
/**
 * @struct nom_de_la_structure
 * @brief Commentaire rapide sur cette structure
 * @details Commentaire détaillé ici (si commentaire sur
 * plusieurs lignes, ne rien mettre derrière @details, mettre
 * la description détaillée sur la ligne suivante, les sauts de
 * lignes seront reconnus directement)
 *
 */
typedef struct
{
```

```
    char variable1; /*!<Mon commentaire sur la variable*/
    char variable2; /*!<Mon commentaire sur la variable*/
}nom_de_la_structure;
```

– Commenter un enum :

```
/**
 * @enum NOM_ENUM
 * @brief Description rapide la l'enum
 * @details Commentaire détaillé ici
 */
typedef enum
{
    NUM_1 =0, /*!< Premier numéro*/
    NUM_2 =1, /*!< Deuxième numéro*/
    (...)
}NOM_ENUM;
```

7) Ajout de code dans un commentaire :

```
/**
 * (mettre ici tout ce qui précède le code à ajouter)
 * @code
 * (Mettre le code source ici)
 * @endcode
 * (Mettre ici tout ce qui suit le code à ajouter)
 */
```

8) Eviter qu'une partie du code soit dans la documentation Doxygen :

```
/**
 * @cond
 */
(Mettre ici le code source qui ne doit pas être dans la documentation Doxygen)
/**
 * @endcond
 */
```

9) Point important :

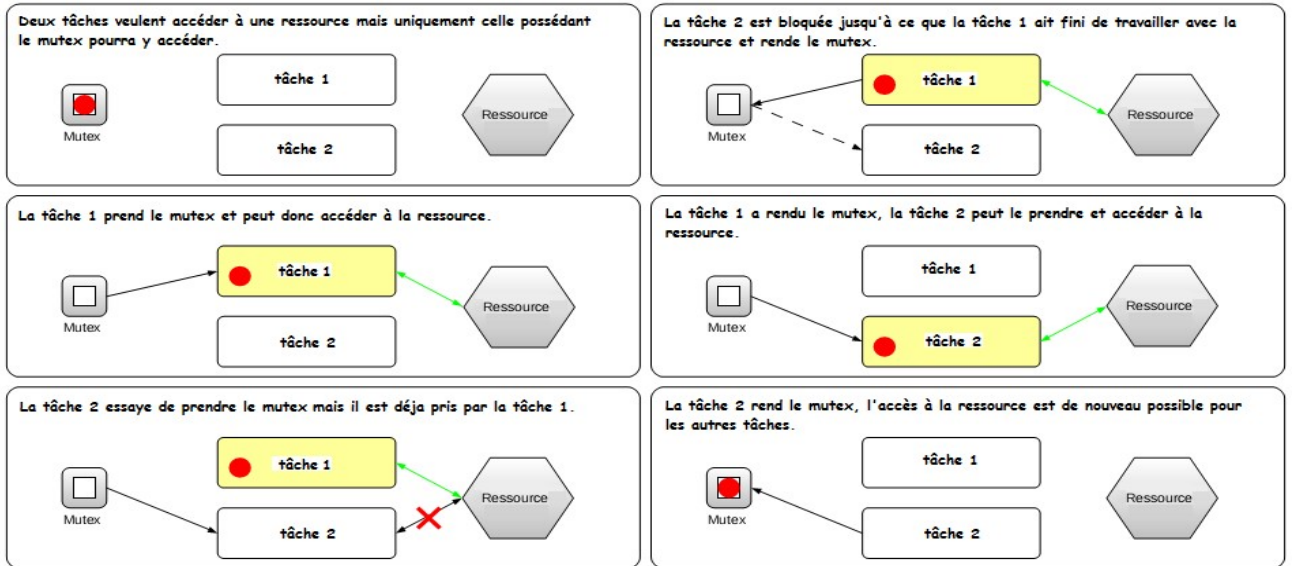
Les fichiers .c et .h commentés en Français doivent être encodés en **UTF-8** (sinon les accents ne seront pas reconnus).

Pour cela, un moyen simple est d'utiliser :

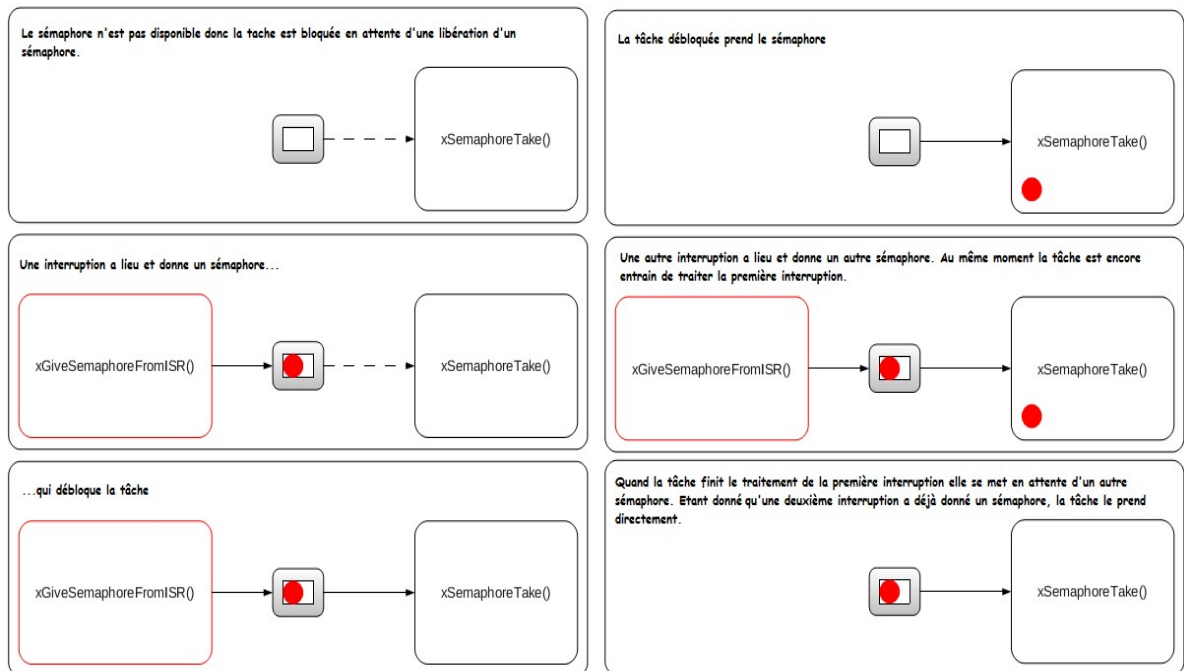
Notepad++ → Encodage → Convertir en UTF-8 → Enregistrer.

# ANNEXE F : Notion de protection des ressources

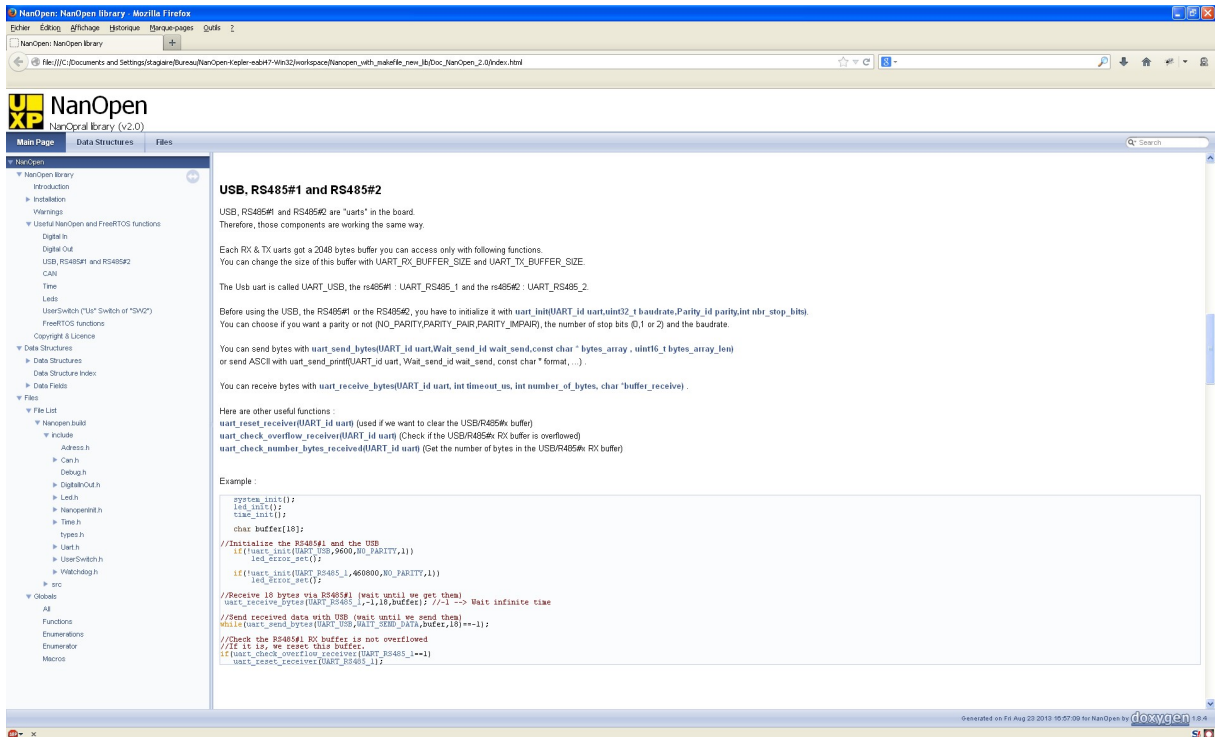
- **Mutex** : PrIMITIVE de synchronisation permettant d'éviter que des ressources partagées (temps du processeur, mémoire vive, périphérique externe ...) d'un système ne soient utilisées en même temps.



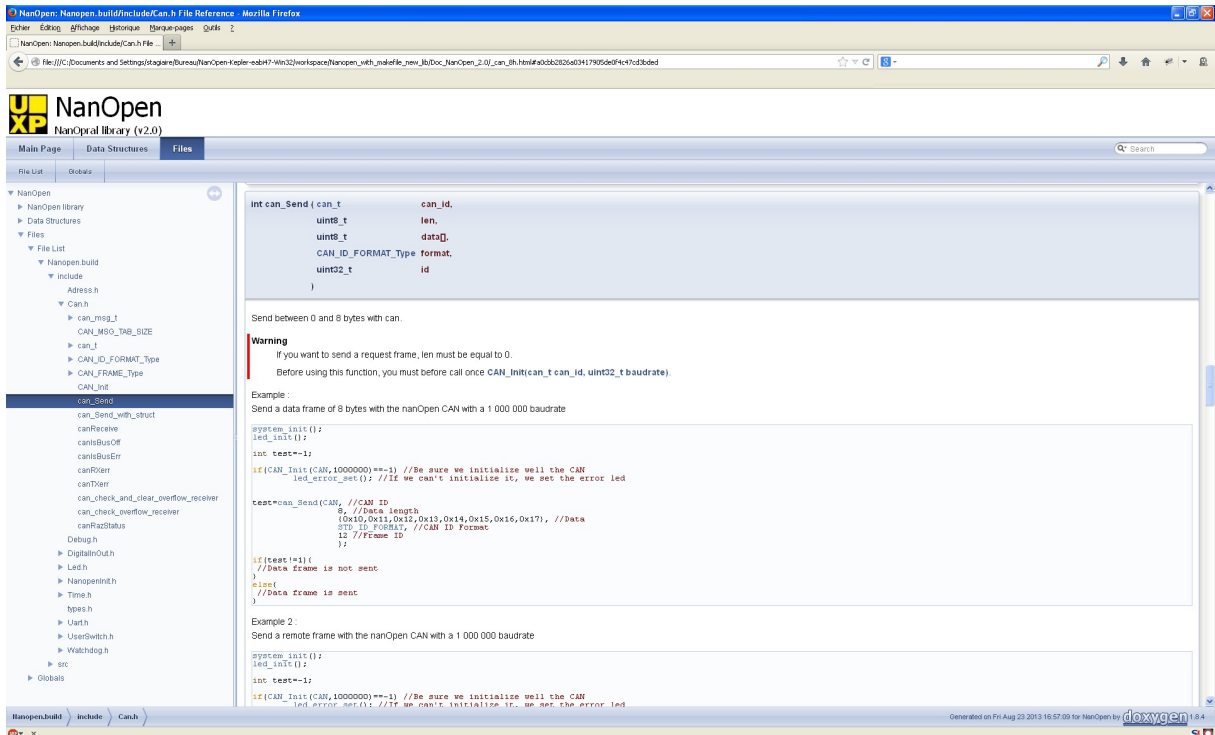
- **Sémaphore** : Comme pour le mutex, un sémaphore binaire ou continue permet de gérer les interruptions et les ressources.



# ANNEXE G : Documentation Doxygen finale (Javadoc)



Introduction de la documentation



Explication d'une fonction dans la documentation

# ANNEXE H : Environnement de développement pour le client final

The screenshot displays the NanOpen development environment. The top-left pane shows a file explorer with the following structure:

- Fichiers complètes**
  - src
    - canic
    - delugnc
    - digitaloutc
    - ledc
    - Nanopenintc
    - seranopc
    - timec
    - ultrc
    - userwatchc
    - watchdogh
    - watchdogh\_
  - src
    - canic
    - delugnc
    - digitaloutc
    - ledc
    - Nanopenintc
    - seranopc
    - timec
    - ultrc
    - userwatchc
    - watchdogh
    - watchdogh\_
- Librairie**  
NanOpen ainsi que FreeRTOS (et le CMSIS dont il a besoin)
- Fichiers utilisateurs**  
(configuration et fichier principal)
  - ProjectBuild
  - include
    - compagnonh
    - manh
  - src
    - manic
- Fichiers de compilation**
  - DocFile
  - DocFile
    - Launch\_Makefile.bat
    - make.exe
    - makefile
  - NanOpen\donloaded\ghosden\terminal-1.1.4.19r
  - NanOpen\donloaded\ghosden\terminal-1.20.0r
  - Recovery.html
  - UVP.mmo
- Documentation**  
doxygen

The main editor window shows the NanOpen library documentation for the `can_receive` function. It includes a list of useful functions and an example code snippet:

```

// Initialize the CAN
if(CAN1_Init(CAN1,1000000)==1) //Be sure we initialize well the CAN
    led_green_led(); //If we can't initialize so, we see the error led
can_msg_t pMessage; //Frame we will send soon
12, //Frame ID
0x0, //Data length
8, //Data length
STD_ID_FORMAT, //CAN ID format
DATA_FRAME //Type of frame
);
can_send_send_with_struct(CAN1,pMessage); //Try to send a frame
if(test==1)
    //Data frame is not sent
else {
    //Data frame is sent
    //Data frame is received a frame from can and that this frame got a ID equal to 12.
    //Data frame is received a frame
    if(test==1) //If we received a frame
        if(pMessage->id==12)
    
```

The bottom status bar indicates the documentation was generated on Fri Aug 23 2013 18:57:09 for NanOpen by doxygen 1.8.4.



# ANNEXE I : Evaluation de stage



Ecole Nationale  
Supérieure  
de l'Electronique  
et de ses Applications

## Evaluation de stage des élèves ingénieurs de l'ENSEA

Nom : **COMTE - GAZ**

Prénom : **Quentin**

Année : **2013**

Dates du stage : du **1/07/2013** au **28/08/2013**

Entreprise : **UXP**

Adresse : **12 rue Pierre de Coubertin 38170 Seyssinet Pariset**

Tuteur Industriel : **Bossard Guillaume**

Coordonnées : **www.uxp.fr -> Contact / +33 4.76.84.28.80**

### Appréciation générale (cochez l'une des cases proposées) :

Etes vous globalement satisfait :

- du comportement du stagiaire  oui  non
- du travail fourni  oui  non
- de ses connaissances  oui  non

Reprendriez-vous un stagiaire de l'ENSEA dans les mêmes conditions ?

oui  non

### Appréciation sur le rapport de stage :

- Clarté de l'expression écrite  bon  moyen  faible
- Esprit d'analyse et de synthèse  bon  moyen  faible

### Remarques générales sur le comportement du stagiaire :

(ne remplir les rubriques que lorsqu'elles ont pu être appréciées durant le stage)

- Autonomie  bon  moyen  faible
- Conscience professionnelle  bon  moyen  faible
- Ténacité au travail  bon  moyen  faible
- Aptitude au travail en équipe  bon  moyen  faible
- Curiosité scientifique et technique  bon  moyen  faible
- Créativité  bon  moyen  faible
- Clarté de l'expression orale  bon  moyen  faible
- Maîtrise des outils utilisés  bon  moyen  faible
- Efficacité dans le travail  bon  moyen  faible

Date et Signature du tuteur industriel :

**23/08/2013**

Tout commentaire jugé utile peut être ajouté ci-dessous