Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université Mouloud MAMMERI de Tizi-Ouzou
Faculté de Génie Electrique et d'Informatique
Département d'informatique

MEMOIRE

DE FIN D'ETUDE

En vue de l'obtention du diplôme de Master Académique en informatique

Option: Système Informatique (SI).

THEME : *Conception d'un disjoncteur électronique pour la protection des équipements et des personnes.*

Encadré par :
Mr H. ACHOUR

Réalisé par:
Mr BELAIDI Hamid.

Membre de jury :
- *Mr. DAOUI Mehammed*
- *ZERMI Rachid*

Promotion 2019 – 2020

Table des figures

Table des matières

Sommaire

Introduction

Depuis quelques années on utilise de plus en plus d'équipements contenant des systèmes électroniques embarqués qui en facilite l'usage et ce dans tous les domaines tels l'automobile, l'aviation, les équipements médicaux,...

Ces systèmes embarqués sont des systèmes électroniques et informatiques autonomes ne possédant pas des entrées/sorties standards comme un clavier ou un écran d'ordinateur. La machine et le logiciel sont ainsi intimement liés et ne sont pas aussi facilement discernables comme dans un environnement de travail classique de type PC.

Certains de ces systèmes embarqués sont dit temps réel quand leur temps de réponse est soumis à des contraintes plus ou moins strictes selon le système piloté. Ainsi un système embarqué pilotant le freinage d'une automobile est soumis à des contraintes dures ou strictes.

Les systèmes d'exploitation utilisés dans ce genre d'équipements sont appelés des RTOS (Real Time Operating System). Il en existe plusieurs dont certain sont open source (Libre).

FreeRTOS est une des solutions proposée par l'écosystème Open Source. Porté sur plus d'une dizaine d'architectures différentes, FreeRTOS propose un OS temps réel performant, facile d'utilisation, d'empreinte mémoire faible et robuste.

Le réseau électrique Basse Tension Algérien est sujet à plusieurs problèmes, tels les: Coupures intempestives qui arrivent par temps d'orage (microcoupures), surtension... Ces disfonctionnements sont l'une des causes des défaillances des équipements domestiques,

De plus selon les normes Algériennes, une installation domestique ne possède comme organe de protection qu'un disjoncteur différentiel de 300 MA, ce qui est loin d'être une protection suffisante pour la protection des personnes.

Dans le cadre de notre projet, nous nous proposons d'embarqué cet OS sur une carte STM32 de type Nucléo-F411RE et d'élaborer un système de surveillance du réseau électrique domestique pour la protection des équipements et des personnes. En plus des fonctions du disjoncteur différentiel, il aura la fonction de protection contre les surtensions (ou sous-tensions) et les microcoupures.

A cet effet, nous allons commencer par un premier chapitre sur les systèmes embarqués et temps réel (Généralités). Dans le deuxième chapitre on abordera les systèmes d'exploitation temps réel open sources existants et nous ferons un petit comparatif

entre eux. La mise en œuvre d'un systèmes FreeRTOS sur microcontrôleur STM32 sera abordée dans le troisième chapitre et on terminera par un quatrième chapitre qui portera sur la conception de notre application.

Chapitre I -Généralités sur les systèmes embarqués:

1- Généralités

1-1 Définition des systèmes embarqués:

- Un système embarqué est un système électronique et informatique autonome, souvent temps réel, spécialisé dans une tâche bien précise. Ses ressources sont généralement limitées [1].

- Il n'a pas réellement de clavier standard (Bouton Poussoir, clavier matriciel...).

- L'affichage est limité (écran LCD...) ou n'existe pas du tout.

- Il ne possède généralement pas des entrées/sorties standards et classiques comme un clavier ou un écran d'ordinateur.

Il comprend 02 parties:

Une partie matérielle: Micro-processeur, microcontrôleurs, coprocesseurs, DSP, Mémoires, ASIC, Interfaces d'entrées/sorties et une autre logicielle.

90% des microprocesseurs sont fabriqués comme des composantes de systèmes embarqués.

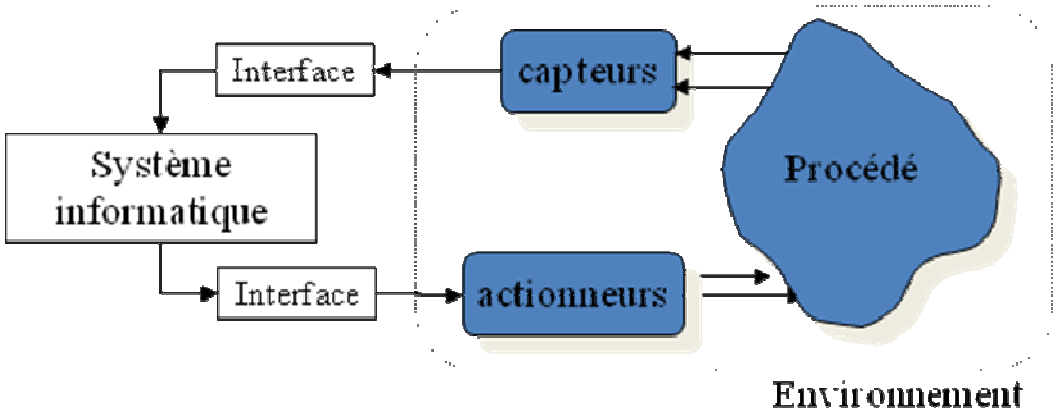

Figure 1 : Schéma d'un système embarqué

Figure 2: Exemple d'un système embarqué

1-2 Caractéristiques générales :

Les caractéristiques générales d'un système embarqué sont [2] :

- Ils sont dédiés à des applications spécifiques

- Coût réduit, maximisation du rapport performance/prix

- Volume restreint (compact, pas modulaire)

- Capacité mémoire adaptée (Mémoire restreinte, souvent pas de disque).

- Capacité de calcul appropriée à l'application

- Contrainte de sécurité et temps réel.

- Consommation d'énergie maitrisée, voir très faible en cas d'utilisation sur batterie

Les systèmes embarqués doivent d'être robuste, simple, fiable et fonctionnel. Le système doit toujours fonctionner correctement surtout si la sécurité des personnes est en jeu.

- Les crash soft sont beaucoup plus graves dans les systèmes embarqués que dans les systèmes de bureau.

- Les systèmes embarqués doivent fonctionner parfois dans des conditions d'environnement extrêmes.

- Exécution temps réel (souvent): Rater une échéance peut causer une erreur de fonctionnement.

1-3 Contraintes :

Les différentes contraintes sont:

- Espace mémoire limité.

- Minimiser la consommation d'énergie au maximum, énergie limitée.

- Coût, il est réduit dans le cas de production de série.

- Puissance de calcul qui épuise l'énergie, dans ce cas il faut utiliser juste ce qu'il faut

- Sureté de fonctionnement, donner des résultats justes et dans les délais sinon y aura des conséquences néfastes soit humaines ou matérielles.

Sécurité: Les informations doivent être confidentielles

1-4 Architecture d'un système embarqué [3] :

-Il est réalisé autour d'un microcontrôleur (µC) qui est un système à processeur dans un seul chip (SoC) qui comprend: processeur, mémoire, GPIO (entrées/sorties simples configurables), contrôleur de bus, contrôleur d'interruption, contrôleur d'écran, USB, Ethernet, …

D'un très bon rapport performance/prix et performance/consommation, c'est un ensemble compact (volume optimisé) qui possède un démarrage autonome du système (boot), mais pas de disque dur mais plutôt de la mémoire flash...

Les systèmes simples peuvent avoir des boutons ou des LED et les plus complexes des écrans tactiles.

Généralement, ils ne possèdent pas d'extension possible et leur construction est non modulaire.

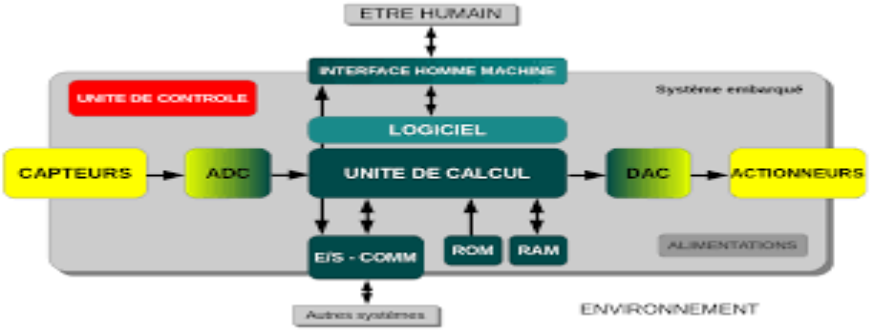

Figure 3: Architecture d'un système embarqué

1-5 Classification des systèmes embarqués :

Il existe plusieurs critères de classification pour les systèmes embarqués, comme par exemple le gabarit, le type de calcul, ou le segment du marché économique dans le quel l'appareil figure. Ce dernier critère est le plus utilisé dans les systèmes embarqués.

1-6 Types des systèmes embarqués :

Les différents types sont :

Calcul général: Jeu vidéo

Contrôle de systèmes en temps réel: Systèmes de navigation aérien

Traitement de signal: Radar.

Transmission d'information et commutation: Téléphone, internet.

1-7 Exemples de systèmes embarqués:

Comme exemples de systèmes embarqués on peut citer : Système d'alarme, pilote automatique d'avion, gestion alimentation moteur d'une voiture, Smartphone, imprimante, ascenseur, système de surveillance, lave linge...

2- Les systèmes embarqués et temps réel :

2-1 Définition:

Un système temps réel se compose d'un ou plusieurs sous-systèmes devant répondre en un temps fini et spécifié à des stimuli générés par le monde extérieur [4].

Une réponse hors échéance est invalide, même si son contenu semble correct et peut causer de grands dégâts que ça soit humains ou matériels.

L'absence d'une réponse est aussi grave qu'une réponse erronée.

Fonctions fondamentales d'un système temps réel :

Les principales fonctions sont:

- La collecte d'information (capteurs, communications)

- Les actions du système sur son environnement (actuateurs)

- Les interactions humain-machine (assister les opérateurs)

Exigences temporelles :

- Il est important de favoriser les processus selon leurs nécessités afin de respecter les échéances.

- La Limitation les ressources conduit au blocage, ce qui ne permet pas de progresser.

But des contraintes temporelles :

L'application doit disposer d'une image précise et cohérente de la réalité au cours du temps

L'objectif du temps réel vise donc à borner la différence entre l'image du système dans la réalité et celle dans l'application ($|R(t)-A(t)|<\varepsilon$)

Pour actualiser l'image dans l'application, celle-ci lit notamment des capteurs périodiquement

Contraintes Non Fonctionnelles :

Les contraintes non fonctionnelles sont importantes voire critiques.

Ces systèmes doivent être prévisibles ainsi que leur Réactivité et leur cohérence temporelle et définir des fenêtres temporelles pour la validité des données.

Il faut bien aussi respecter l'échelle de temps du système, ni trop vite, ni trop lent

Exemple: (avion = msec, voiture = sec)

- Les traitements doivent êtres exécutés en un temps borné (connu)

- la Fiabilité et la Disponibilité doivent être satisfaites.

- Il faut Assurer l'exactitude numérique du résultat des traitements et la continuité de la fonction en cas de conditions adverses (tolérance aux pannes, aux malveillances …)

2-2 Classification:

On distingue le temps réel strict ou dur (de l'anglais hard real-time), le temps réel souple ou mou (en anglais soft real-time) et le temps réel ferme (firm real-time), suivant l'importance accordée aux contraintes temporelles :

A) Le temps réel strict :

Ce type ne tolère aucun dépassement de ces contraintes, car dans certains cas ce dépassement peut provoquer des graves dégâts, voire catastrophiques comme dans le pilotage automatique d'avion, système de surveillance de centrale nucléaire, etc.

B) Le temps réel souple:

On parle de temps réel souple si un dépassement occasionnel ne met pas en danger le

système, c'est a dire si les retards sont autorisés : échéance molle.

On tolère un dépassement exceptionnel, qui pourra être compensé à court terme.

Exemple: Systèmes d'acquisition d'image pour affichage .

C) Le temps réel ferme (firm real-time):

Le manquement occasionnel des échéances n'influence pas le système.

Ex.: projection vidéo (perte de quelques trames d'images).

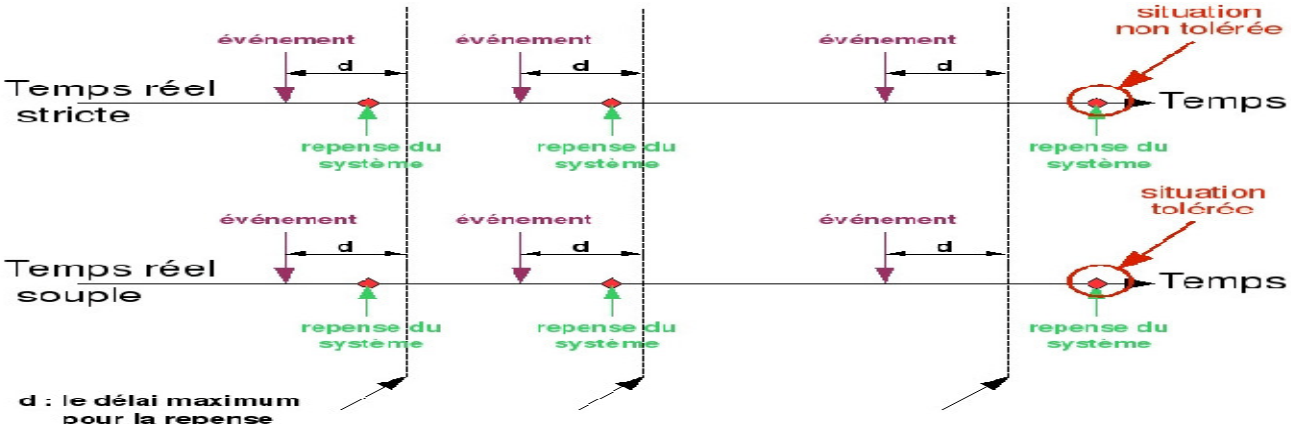

Figure 4 : Classification temps réels

Test d'acceptabilité :

Il faut prouver que les limites temporelles ne se seront pas dépassées, c'est la théorie de l'ordonnancement. Elle dépend de l'ordonnanceur utilisé et des caractéristiques des tâches du système.

Condition de charge [5]:

Pour tout système de tâches, la condition suivante est nécessaire mais pas suffisante à sa faisabilité :

$$\sum_{i=1}^{n} Ci/Ti \leq 1$$

Avec :

C_i le temps de calcul de la tâche n° i

T_i sa période.

Si la valeur est supérieur 1 veut dire que le processeur ne peut pas fournir plus de temps.

Temps de réponse dans le cas le plus défavorable:

Une tâche est faisable si son temps de réponse au maximum est inférieur ou égal à son échéance.

Un système est faisable si toutes ces tâches sont faisables.

2-3 Caractéristiques du temps réel:

- Il doit respecter les échéances temporelles et garantir la fiabilité.

- Le fonctionnement doit être conforme aux exigences.

- La vérification à priori de la conception.

- Il faut y avoir des plates formes d'essai: On doit vérifier a l'avance le bon fonctionnement du système.

2-4 Les différents types d'événements dans les temps réels:

Dans les systèmes temps réel l'ordonnancement des taches nous permet de planifier l'exécution des requêtes en respectant les contraintes de temps.

Il ya 03 types d'événements:

-Périodique : Les événements se réveillent d'une manière régulière

-Apériodique : Dans ce cas les événements ne suivent aucune règle, ils arrivent d'une manière aléatoire.

-Sporadique : Les événements ne suivent aucune période dans le temps, mais dés qu'un événement arrive un laps de temps doit s'écouler pour qu'un autre événement se manifeste.

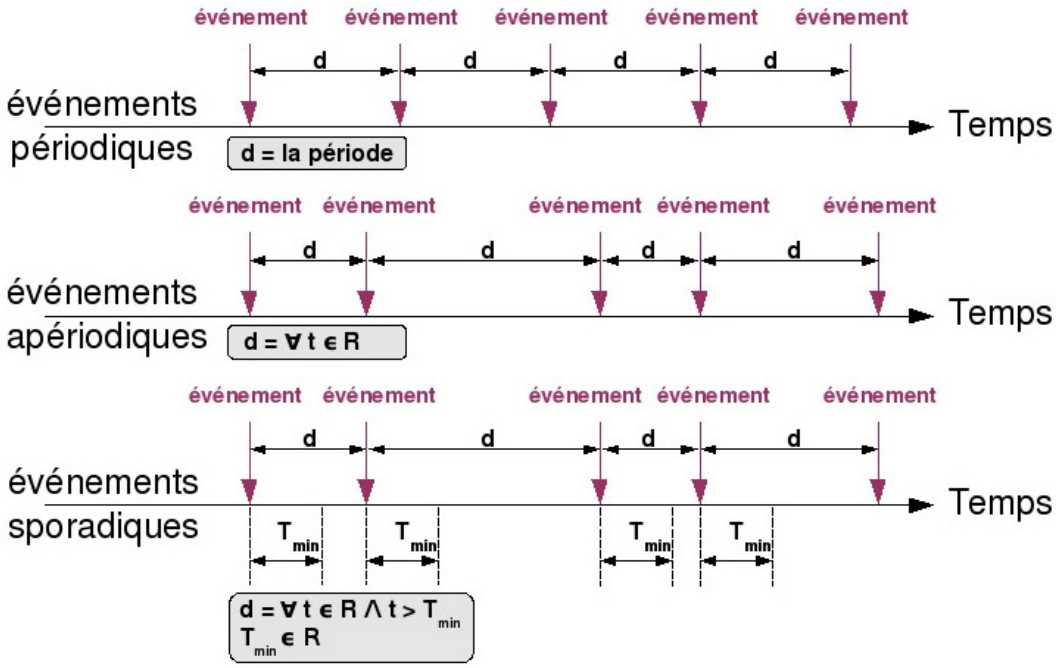

Figure 5 : Représentation de l'afflux des événements dans les trois classes

2.5 Conception d'un système temps réel:

Dans la conception d'un système temps réel l'enjeu le plus important c'est le temps, le système doit répondre toujours au bon moment, c'est pour cela qu'il faut le prendre au sérieux lors de la conception.

Dans cette partie nous allons présenter les critères à prendre pour la conception d'un système temps réel qui sont:

- Description et expression des contraintes temps réel
- Représentation simultanée de l'évolution logique et temporelle du système
- Prédiction et estimation des temps de réponse
- Sélection de l'architecture, du matériel et du logiciel
- Le matériel conditionne les coûts de production
- Le logiciel conditionne les coûts de développement
- Compromis et équilibre matériel-logiciel (Outils de spécification et validation).

Chapitre II

1 Les systèmes d'exploitation temps réel opens sources existants :

Dans cette partie nous allons présenter les différents systèmes d'exploitation open source existants qui sont:

FreeRTOS, Contiki, TinyOS, LiteOS(en), Xenomai, BeRTOS, ChibiOS/RT, DuinOS, eCosPro, Lepton (en), Nano-RK, PICOS18, RIOT OS, SimpleAVROS, Trampoline Operating System (OSEK), TUD:OS, TNKernel, Mynewt, NuttX, RTEMS, RT-Thread.

2 Définition et caractéristiques principales de ces systèmes:

FreeRTOS:

FreeRTOS est un système d'exploitation en temps réel (RTOS) leader du marché pour les microcontrôleurs et les petits microprocesseurs. Distribué gratuitement sous la licence open source du MIT, FreeRTOS comprend un noyau et un ensemble croissant de bibliothèques pouvant être utilisées dans tous les secteurs de l'industrie. FreeRTOS est construit en mettant l'accent sur la fiabilité et la facilité d'utilisation.

Il existe de plus en tout 4 variantes de FreeRTOS :

- FreeRTOS

- FreeRTOS + Trace : idem que FreeRTOS avec des outils propriétaires permettant d'instrumenter le code. Par exemple des outils graphique de trace.

- OpenRTOS : version commerciale sous licence de FreeRTOS

- SafeRTOS : version certifié SIL3 TUV

Caractéristiques [7] :

- Les livres et manuels de référence disponibles; il a un faible encombrement mémoire, faible surcharge et exécution rapide; il est destiné aux amateurs et développeurs professionnels travaillant sur des produits commerciaux. Il prend en charge la Coroutine (les coroutines dans FreeRTOS sont des tâches simples et légères) et la trace via des macros de

trace génériques. Des outils tels que Tracealyzer utilise une stratégie de tarification freemium , offrant une version gratuite limitée en fonctionnalités

Contiki :

Contiki est un système d'exploitation léger et flexible pour capteurs miniatures en réseau.

Caractéristiques :

- L'espace mémoire utilisé par le système d'exploitation et par l'application doit être suffisamment faible pour être contenu dans la mémoire du capteur.

- L'énergie électrique, souvent apportée par une batterie, peut être problématique à renouveler. Des systèmes de captage, comme des éléments photovoltaïques, éoliennes, ou autres peuvent être utilisés dans certains cas.

Contiki implémente 2 mécanismes de communication :

la couche de protocole Rime. Elle fournit à la couche applicative un jeu d'instructions de communication, permettant les différentes connexions avec les capteurs voisins.

la couche uIP. Contiki implémente uIPv4 et uIPv6.

- La portabilité consiste à adapter le système d'exploitation aux capteurs, selon les éléments électroniques les constituant.

- L'interopérabilité d'un capteur est le fait de pouvoir communiquer avec des capteurs gérés par un système d'exploitation différent. Le support de 6LoWPAN permet à Contiki de communiquer avec les matériels via un réseau sans-fil suivant la norme 802.15.4. Contiki est réputé pour être un système d'exploitation robuste et mature, fournissant IPv4 et IPv6 pour les réseaux de capteurs sans-fil.

LiteOS(en) :

LiteOS est un système d' exploitation pour smartphone du fabricant chinois de smartphones Huawei, léger et open source .LiteOS est conçu pour avoir un faible encombrement, ce qui économise de l'espace et réduit la charge du système d'exploitation sur l'appareil. Le système d'exploitation est mis à disposition sous une licence BSD, qui est une classe de licences simples et gratuites pour les logiciels informatiques qui a été initialement développé à l'Université de Californie à Berkeley. LiteOS prend en charge les smartphones,

les appareils portables , les applications de fabrication intelligentes, les maisons intelligentes et l'Internet des véhicules (IoV). L'OS sert également de plateforme de développement d'appareils intelligents. La plateforme simplifie le développement et la connectivité des appareils IoT, tout en se concentrant également sur l'amélioration de l'expérience utilisateur.

Caractéristiques :

- Il a un petit noyau léger (<10 Ko), faible consommation, démarrage rapide en quelques millisecondes, il prend en charge NB-IoT, Wi-Fi, Ethernet, BLE, Zigbee et d'autres protocoles IoT différents et il prend en charge aussi l'accès à différentes plateformes cloud.

ChibiOS/RT :

ChibiOS/RT est un système d'exploitation temps réel rapide et compact supportant différentes architectures et est un logiciel libre sous licence GPLv3. Il est développé par Giovanni Di Sirio. Il est notamment utilisé par le synthétiseur Axoloti sur un processeur STM32 F4 de STMicroelectronics.

Caractéristiques[6] :

Il est Préemptif multithreading, il a 128 niveaux de priorité, Les threads sont planifiés à tour de rôle au même niveau de priorité, Il possède des minuteries logicielles, variables de condition, les messages synchrones et asynchrones, les Files d'attente, mutex avec prise en charge de l' algorithme d'héritage prioritaire, il prend en charge la bibliothèque du système de fichiers FatFs et les E / S synchrones et asynchrones avec capacité de temporisation.

Lepton (en) [8] :

Le système d’exploitation temps réel pour systèmes enfouis Lepton a été développé par les fondateurs d’o10ée (se prononce Odyssée).

Utilisé depuis plus de 5 ans dans l’industrie, cet RTOS open source (licence MPL) propose une architecture évoluée intégrant la notion de processus, une conception type UNIX avec notamment une orientation tout fichier et une interface système POSIX. Cette approche permet :

- d'avoir un système léger, une prise en main rapide, une grande flexibilité, portabilité et stabilité du système et il est économique en consommation d'énergie.

Nano-RK [9] :

Est un système d'exploitation en temps réel (RTOS) de l'Université Carnegie Mellon conçu pour fonctionner sur des microcontrôleurs pour une utilisation dans les réseaux de capteurs. Nano-RK prend en charge un planificateur entièrement préemptif à priorité fixe avec des primitives de synchronisation à granularité fine pour prendre en charge des ensembles de tâches en temps réel

Caractéristiques :

- Sa configuration est statique.

- Disponibilité de la minuterie de surveillance.

- Il possède un mode de veille profond.

PICOS18 [10] **:**

PICos18 est un noyau temps réel basé sur la norme automobile OSEK/VDXÔ et destiné aux microcontrôleurs PIC18 de la société Microchip Technology Inc. Il comporte :

- Le gestionnaire d'évènement qui permet la gestion des évènements d'une tâche.

- Le gestionnaire d'interruption qui permet l'activation et la désactivation des interruptions du système.

- C'est le cœur du noyau qui gère les tâches.

- Le gestionnaire d'alarmes et de compteurs permet la mise à jour périodique des alarmes et compteurs associées aux tâches.

- Les Hook routines est utilisé pour le déroulement normal du noyau de façon à prendre temporairement le contrôle du système.

- Le gestionnaire de tâches permet de changer l'état d'une tâche.

RIOT OS :

RIOT est un système d'exploitation léger pour systèmes en réseau avec des contraintes de mémoire, focalisé sur les appareils à faible consommation électrique pour l'Internet des objets. C'est un logiciel libre, publié sous Licence publique générale limitée GNU (LGPL)

Caractéristiques:

- RIOT offre un environnement facile et convivial pour les programmeurs, il n'est pas gourmand en matière de ressources, il a une architecture micro-noyau et planificateur tickless sur des appareils très légers, robustesse et flexibilité d'empreinte de code, permet une efficacité en énergie maximale et enfin multi-threading avec une surcharge ultra faible inférieur à 25 octets par thread.

- Avec RIOT on peut préparer les petites applications sur internet avec une prise en charge système commune, la compatibilité., ...).

TUD:OS [11] :

Le système d'exploitation TUD:OS est un système d'exploitation basé sur le micro-noyau L4, avec de bonnes propriétés en temps réel et de sécurité, développé au TU Dresden.

Caractéristiques:

- Son micro-noyau lui permet de réduire la base informatique de confiance en exécutant de petites applications approuvées côte à côte avec des applications non approuvées, toutes en tant qu'applications en mode utilisateur non privilégiées.

- Il ya une possibilité que des applications en temps réel côte à côte avec des applications non en temps réel soient exécutées en même temps.

- Il offre toujours un environnement familier par la possibilité d'exécuter une ou plusieurs instances de L4Linux. L4Linux est un port Linux du micro-noyau Fiasco qui est compatible binaire avec Linux natif mais fonctionne en mode utilisateur privé.

- Une panne d'un pilote de périphérique n'entraînera pas une panne complète du système, le pilote peut être redémarré et le système continuera de fonctionner sans aucun problème..

TNKernel [12] **:**

TNKernel est un noyau temps réel compact et très rapide pour les microprocesseurs 32/16/8 bits intégrés. TNKernel s'est inspiré de la spécification µITRON 4.0.

TNKernel est un appareil entièrement portable, il comprend des sémaphores, des mutex, des files d'attente de données, des drapeaux d'événements et des pools de mémoire de taille fixe. Le système effectue une planification basée sur la priorité préemptive.

Caractéristiques :

Dans TNKernel , une tâche est une branche du code qui s'exécute simultanément avec d'autres tâches du point de vue du programmeur. Au niveau physique, les tâches sont réellement exécutées à l'aide du partage du temps processeur. Chaque tâche peut être considérée comme un programme indépendant, qui s'exécute dans son propre contexte (registres du processeur, pointeur de pile, etc.).

Apache Mynewt :

Est un système d' exploitation modulaire en temps réel pour les appareils connectés à l' Internet des objets (IoT), qui doivent fonctionner pendant de longues périodes sous des contraintes d'alimentation, de mémoire et de stockage. Il est un logiciel libre et open-source en incubation sous la Apache Software Foundation , avec le code source distribué sous la licence Apache 2.0 , une licence permissive qui est favorable à l' adoption commerciale des logiciels open-source

Caractéristiques[13] :

Les minuteries sont programmables, il est multithreading préventif, les journaux et statistiques situés au niveau du système, prise en charge WiFi via l'interface socket, et IP de base, mise à jour du firmware à distance, sémaphores, mutex et files d' attente d' événements

NuttX :

NuttX est un système d'exploitation en temps réel (RTOS) à faible empreinte. Extensible des environnements de microcontrôleur 8 bits à 32 bits, les principales normes régissant NuttX sont les normes Posix et ANSI. Des API standard supplémentaires d'Unix et d'autres RTOS courants (tels que VxWorks) sont adoptées pour des fonctionnalités non disponibles sous ces normes, ou pour des fonctionnalités qui ne conviennent pas aux environnements profondément intégrés (tels que fork ())

Caractéristiques[14] :

- Il est modulaire, il prend en charge l'héritage prioritaire et la programmation en FIFO. Il est préemptif et le fonctionnement est sans tique, possède un systèmes de fichiers multiples et les modules de noyau sont chargeables. Il est évolutif, configurable .

RTEMS :

Le Real-Time Exécutive pour systèmes multiprocesseurs ou RTEMS est un système d'exploitation open source en temps réel (RTOS) qui prend en charge les interfaces de programmation d'application standard ouvertes (API). Il est utilisé dans le vol spatial, le médical, les réseaux et de nombreux autres appareils embarqués utilisant des architectures de processeur.

La particularité de RTEMS, par rapport à d'autres systèmes d'exploitation est le soutien de nombreuses API standard. RTEMS comprend la pile TCP / IP dérivé de FreeBSD et il supporte de nombreux systèmes de fichiers, y compris NFS et la FAT, le système de fichiers utilisé par l'ancien DOS (FAT16) et Microsoft Windows 98 (FAT32).

RT-Thread :

RT-Thread est un système d'exploitation open source temps réel pour les appareils embarqués. Il est distribué sous la licence Apache 2.0 +. RT-Thread est développé par l'équipe de développement de RT-Thread basée en Chine , après dix ans de développement entièrement concentré.

Caractéristiques[15] :

- Il ya une grande efficacité dans la communication entre les threads.

- Son noyau temps réel est orienté objet.

- 8, 32 ou 256 ordonnancement prioritaire ordonnancement multi-thread.

- Pour éviter l'inversion de priorité, les thread sont synchronisés.

- La gestion de la mémoire statique prend en charge la suspension ou la reprise des threads lorsqu'elle alloue ou libère un bloc de mémoire et une gestion dynamique des tas sans thread;

- Un Framework de pilote de périphérique pour garantir une interface standard à une application de professionnelle.

BeRTOS :

BeRTOS est un système d'exploitation en temps réel conçu pour les systèmes embarqués .

Il a une conception très modulaire, qui permet de l'exécuter sur différentes architectures, allant de minuscules microcontrôleurs 8 bits comme l' Avr d'Atmel à

l' architecture ARM 32 bits , et sur des environnements hébergés tels que Linux et Microsoft Windows. BeRTOS est open source , écrit en ANSI C , et pris en charge par les bibliothèques TLS / SSL intégrées populaires telles que wolfSSL .

Bertos est préemptive et multitâches dont le noyau implémente de nombreux IPC primitives comme: Les signaux, Sémaphores et messages.

En plus du noyau, BeRTOS fournit une couche d'abstraction matérielle qui comprend un grand nombre de pilotes périphériques (minuterie, série, ADC , moteurs, écran LCD, capteurs NTC, clavier, buzzer, mémoires), des algorithmes (table de hachage , CRC , MD2 , entropie , RLE), des protocoles de communication et un sous-système de fenêtrage graphique pour les petits écrans.

TinyOS :

TinyOS est un système d'exploitation open source, sous licence BSD, conçu pour les appareils sans fil basse consommation, tels que ceux utilisés dans les réseaux de capteurs, les réseaux personnels, les bâtiments intelligents et les compteurs intelligents.

DuinOS:

Est un système d'exploitation temps réel avec un système en multitache préemptif pour les cartes Arduino utilisant des microcontrôleurs Atmel AVR. Il est développé par RobotGroup. Il est basé sur un noyau solide qui est FreeRTOS et qui est complètement open source. Ce système s'intègre dans l'environnement de développement de l'Arduino.

EcosPro[16]**:**

eCos, qui signifie «système d'exploitation configurable intégré», est un RTOS open source pour les applications profondément intégrées. Déployée sur une variété de marchés et d'appareils, la popularité des eCos est le résultat d'une variété d'avantages commerciaux et techniques par rapport aux offres RTOS concurrentes.

Xenomai:

Xenomai est une extension libre du noyau Linux auquel il apportet des fonctionnalités temps réel dures.

Xenomai utilise la couche de virtualisation Adeos. Il apporte une approche symétrique entre programmation noyau et programmation système au niveau utilisateur sous Linux.

Il introduit le concept de machine virtuelle en programmation temps réel et permet ainsi de disposer de plusieurs interfaces de programmation au choix du programmeur.

Caractéristiques:

Xenomai se distingue par ses performances ainsi que la possibilité d'utiliser son API sans avoir obligatoirement à utiliser son co-noyau Xenomai Cobalt

Trampoline Operating System (OSEK):

Le trampoline est un RTOS statique pour les petits systèmes embarqués. Son API est alignée sur les normes OSEK / VDX OS et AUTOSAR OS 4.2.

Simple AVROS:

Est un RTOS spécialement conçu pour les microcontrôleurs AVR. Compilateur pris en charge - AVR Studio. Langue - Assemblage. Il s'agit d'un RTOS composé de code source. Microcontrôleurs pris en charge - ATmega48.

3 comparatif entre ces systèmes:

- Contiki et LiteOS sont plus flexibles dans le cas d'un changement d'environnement dynamique ou dans le cas d'une reprogrammation au travers du réseau, car ils ont un système dynamique et modulaire contrairement à TinyOS basé sur un système statique et monolithique(homogène)

- RIOT est basé sur une architecture à micro-noyaux. Contrairement à d'autres systèmes d'exploitation à faible utilisation de mémoire comme TinyOS ou Contiki, RIOT permet la programmation d'applications en langages C et C++ ainsi que le multithreading et le temps réel.

Tableau comparatifs des systèmes temps réels open sources existants [17]:

Nom	**Licence**	**Plateformes cible**
BeRTOS	GNU GPL modifiée	DSP56K, I196, IA32, ARM, AVR
ChibiOS/RT	GNU GPL modifiée	x86, ARM7, ARM Cortex-M3, AVR, MSP430
Contiki	BSD	MSP430, A8VR
DuinOS	GNU GPL modifiée	puces Atmel AVR utilisées sur les cartes Arduino
eCosPro	GNU GPL modifiée et eCosPro license [archive]	ARM/XScale, CalmRISC, 68000/Coldfire, fr30, FR-V, H8, IA32, MIPS, MN10300, NIOS II, OpenRISC, PowerPC, SPARC, SuperH, V8xx
FreeRTOS	Licence MIT	ARM, AVR, AVR32, HCS12, IA32, MicroBlaze, MSP430, PIC, Renesas H8/S, 8052, STM32, NIOS II (Altera)
Lepton (en)	MPL	ARM9 (ATMEL AT91SAM9261, AT91SAM9260), ARM7(ATMEL AT91SAM7x, AT91SAM7SE, AT91M55800), CortexM3 (ST STM32F103, Texas Instrument Stellaris) et CortexM4 (Freescale KINETIS).
Nano-RK	Mixed	AVR, MSP430
NuttX RTOS	BSD	Linux user mode, ARM7, ARM9, 8052, SH-1, Renesas MC16C/26, Zilog Z16F, Zilog eZ80 Acclaim!, Zilog Z8Encore!, Z80, partial ports for MIPS
OSEK	n/a	Engine control units
PICOS18	GNU GPL	PIC18
RIOT OS	LGPLv2.1	ARM, MSP430
RTEMS	GNU GPL modifiée	ARM, Blackfin, ColdFire, TI C3x/C4x, H8/300, x86, 68k, Milkymist (en) SoC, MIPS, Nios II, PowerPC, SuperH, SPARC, ERC32, LEON, Mongoose-V
SimpleAVROS	GPLv3	AVR seulement
ThreadX	Propriétaire	ARC, ARM/Thumb, AVR32, BlackFin, ColdFire/68K, H8/300H, Luminary Micro Stellaris, M-CORE, MicroBlaze, PIC24/dsPIC, PIC32, MIPS, V8xx, Nios II, PowerPC, SH, SHARC, StarCore, STM32, StrongARM, TMS320C54x, TMS320C6x, x86/x386, XScale, Xtensa/Diamond, ZSP
Trampoline Operating System (OSEK)	GNU LGPL	AVR, H8/300H, POSIX, NEC V850e, ARM7, Infineon C166, HC S12 ou PowerPC
TNKernel	BSD	ARM, PIC24/dsPIC, HCS08
Xenomai	GPLv2	x86, x86_64, PowerPC, ARM, Analog Devices Blackfin BF52x, BF53x, BF54x & BF56x, NIOS II

Chapitre III

1 Mise en œuvre de FreeRTOS sur NUCLEO STM32F411RE:

Dans cette partie nous allons décrire le étapes à suivre pour implémenter FreeRTOS sur la NUCLEO STM32F411RE .

Tout d'abord nous allons donner une description de cette carte

2 Le Microcontrôleur STM32F411RET6 :

Microcontrôleurs à très faible puissance basés sur le processeur haute performance RISC 32 bits Arm® Cortex®-M4 fonctionnant à une fréquence allant jusqu'à 80 MHz.

Il comporte une unité à virgule flottante de précision unique qui prend en charge toutes les instructions de traitement des données et tous les types de données Arm® de précision unique.

Il utilise un ensemble complet d'instructions DSP et une unité de protection de la mémoire (MPU) qui augmente la sécurité des applications.

Figure 6 : Le microcontrôleur STM32F411RET6

3 La carte de développement NUCLEO-F411RE Nucléo-64

La carte de développement Nucleo est compatible avec Arduino™ et prend en charge la plupart de ses shields. Les embases ST Morpho permettent également d'augmenter la fonctionnalité de la plate-forme de développement ouverte (ODP) Nucleo.

La carte de développement STMicroelectronics NUCLEO-F411RE permet de créer et d'évaluer tout prototype avec la famille STM32 F4 de microcontrôleurs haute performance dans une application intégrée. La carte ARM mbed Enabled intègre un débogueur et programmateur de circuit ST-Link/V2 (SWIM/SWD) et des interfaces avec le microcontrôleur STM32F411RET6 intégré ou un microcontrôleur STM32 adapté.

Qu'est-ce qui est intégré ?

• Carte de développement STM32 Nucléo avec microcontrôleur STM32F411RET6
• Trois LED : communication USB ; utilisateur ; alimentation
• Mémoire Flash de 512 ko
• Connectivité Arduino Uno Revision 3
• Connexion STMicroelectronics Morpho
• Programmateur/débogueur ST-LINK/V2-1 avec connecteur SWD
• Commutateur de mode de sélection pour utiliser le kit comme ST-LINK/V2-1 autonome
• Alimentations : USB VBUS ou source externe (3,3 V, 5 V, 7 à 12 V)
• Des boutons-poussoirs : utilisateur et reset
• Interfaces prises en charge sur USB : port COM virtuel ; stockage de masse ; port de débogage
• Bibliothèque logicielle complète HAL STM32
• La prise en charge des environnements de développement intégré (IDE) inclut les IDE IAR, Keil et GCC

Figure 7: La Nucleo STM32F411RE

Les caractéristiques :

Produit	De 2007 à courant
Conçu par	STMicroelectronics
Max. CPU fréquence d'horloge	24 à 400 MHz
Min. taille caractéristique	130 à 40 nm
microarchitecture	ARM Cortex-M7F ARM Cortex-M4F ARM Cortex-M3 ARM Cortex-M0 + ARM Cortex-M0

4 FreeROS:

FreeRTOS est un environnement exécutif à très faible empreinte mémoire (<10 KB) complètement gratuit et open source. En utilisant une bibliothèque libre supportant les cœurs ARM Cortex, en particulier la gamme STM32, FreeRTOS est utilisable sur tout microcontrôleur muni d'un tel processeur.

On peut dire aussi que FreeRTOS peut être décrit comme un Scheduler ou bien un ordonnanceur !

5 Le STM32CubeIDE[18]:

Pour l'implémentation de FreeRTOS nous avons utilisé le logiciel STM32CubeIDE qui est complètement gratuit.

Pour commencer, il faudrait accéder à la page STM32CubeIDE et télécharger le programme d'installation pour le système d'exploitation utilisé. On doit créer un compte sur le site de ST (gratuit, mais nécessite une adresse e-mail).

Figure 8 : Ecran de bienvenue pour l'installation STM32CubeIDE

Configuration de notre carte

On va démarrer STM32CubeIDE, et une fenêtre nous demandera de choisir notre espace de travail et on clique sur Lunch.

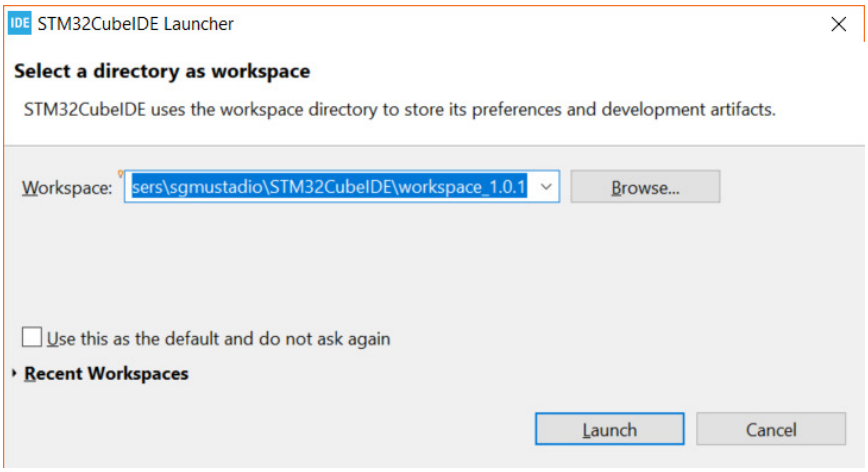

Figure 9: Sélection du répertoire de sauvegarde des projets

On sélectionne Fichier puis Nouveau Projet STM32. On sélectionne l' onglet Sélecteur de carte et on recherche «STM32F411RE» dans la barre de recherche. Le tableau Nucleo apparaît dans la colonne centrale. On Clique sur le nom du STM3F411RE dans le volet inférieur central pour le sélectionner, puis on clique sur NEXT .

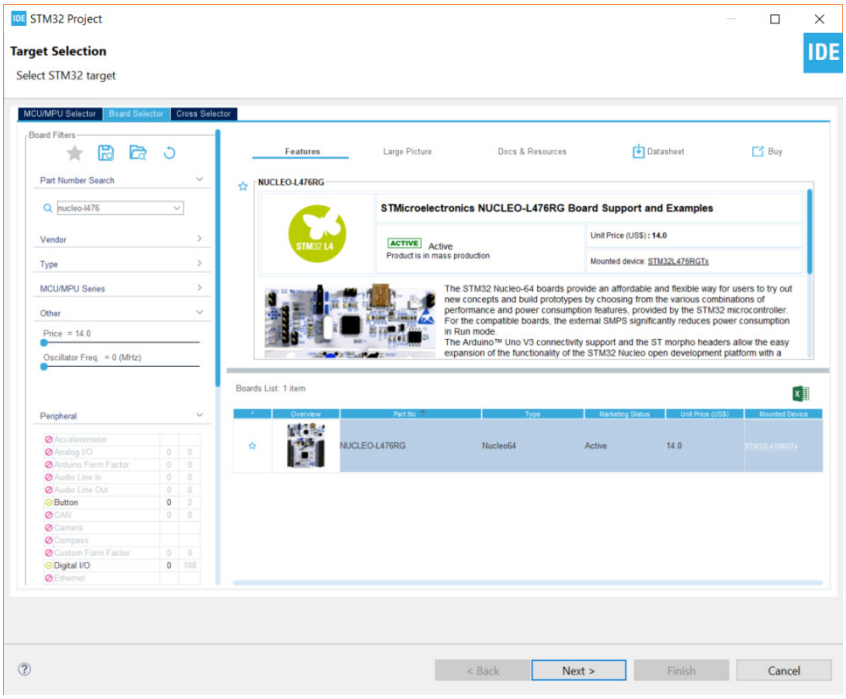

Figure 10 : Sélection de la carte STM32F411RE

Maintenant on va donner un nom à notre projet et on laisse les autres options à leurs valeurs par défaut et on clique sur finish.

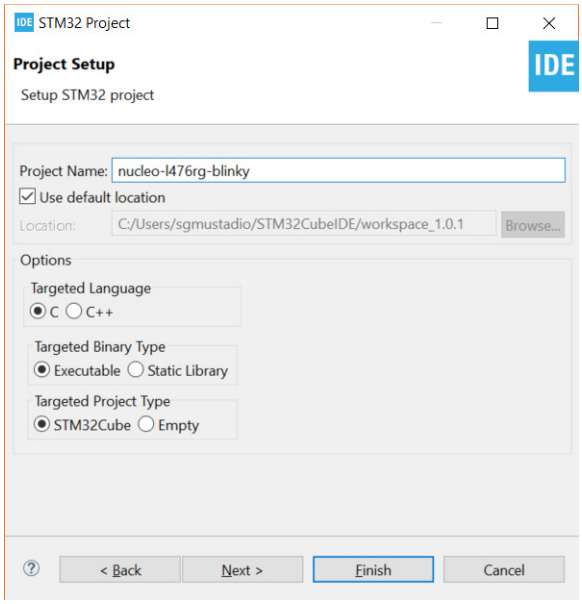

Figure 11: Configuration du projet

6 Activer FreeRTOS sur STM32CubeIDE [19] :

Maintenant que le logiciel STM32CubeIDE est installé et notre projet est crée, nous allons décrire les étapes d'implémentations de FreeRTOS sur notre NCLEO en se basant sur cet exemple:

Dans CubeMX, on va aller dans Catégories puis Middleware puis FREERTOS . Sous Mode , on va changé Interface en CMSIS_V2 .

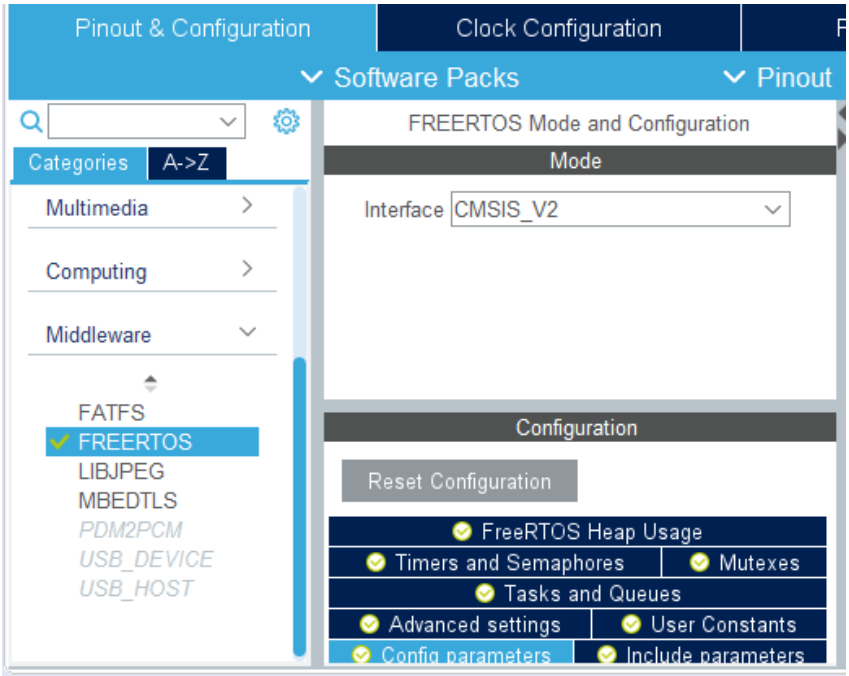

Figure 12: Changement d'interface de FreRTOS.

Dans le volet Configuration , sous Tâches et files d'attente , on va double-cliquer avec la souris sur la tâche par défaut pour apporter des modifications. On changera le nom de la tâche en blink01 et la fonction d'entrée en StartBlink01.

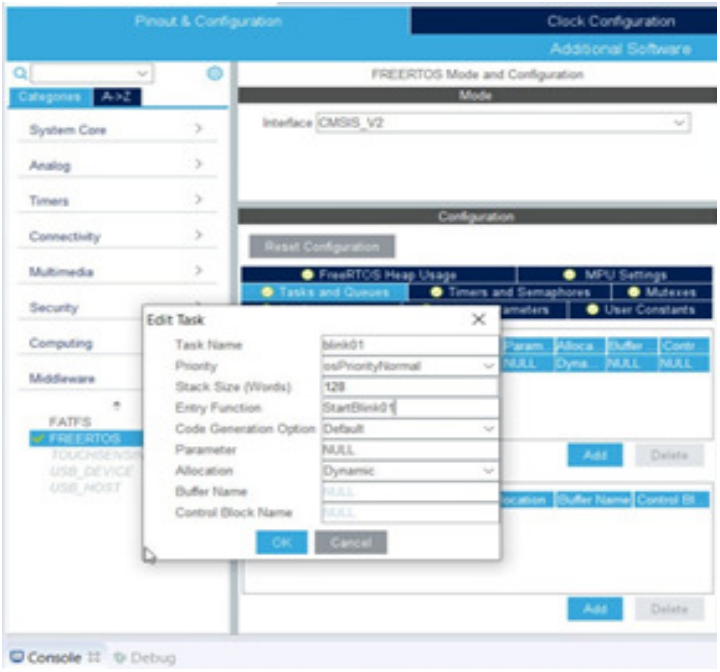

Figure 13: Modification de la tâche1

On clique sur OK et sur Ajouter pour créer une nouvelle tâche, ensuite on va modifier le nom de la tâche en blink02, définir la priorité sur osPriorityBelowNormal et on va modifier le nom de la fonction d'entrée en StartBlink02. La priorité de la tâche 2 est inférieure à celle de la tâche 1, ce qui signifie que si blink01 et blink02 s'exécutent simultanément, blink01 aura la priorité.

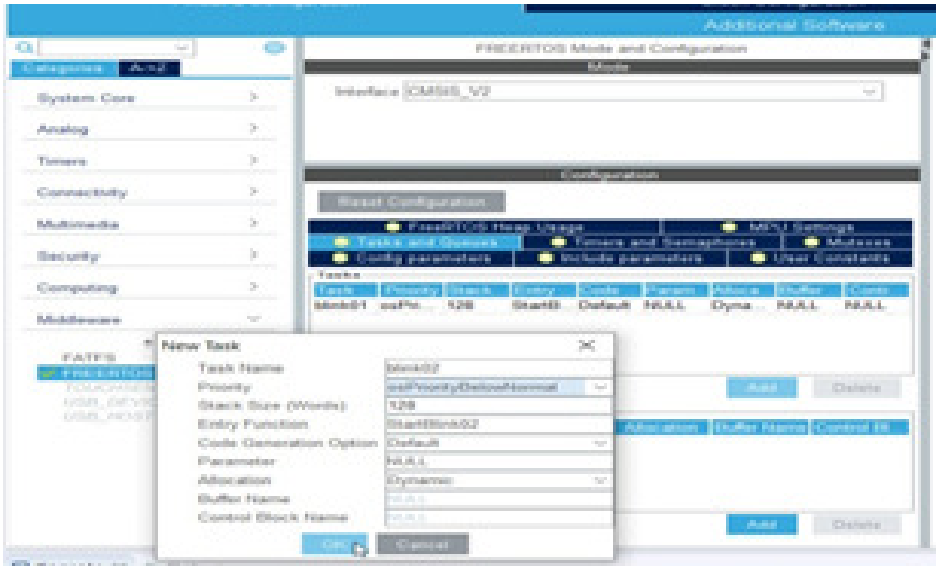

Figure 14: Ajout d'une deuxième tâche.

On va Enregistrer et générer notre code.

Dans le fichier main.c, seront édités les taches qui seront lancées sous forme de threads [20] :

Exemple de programme de création d'un thread blink01 :

```
 / * Créer le (s) thread (s) * /
 / * définition et création de blink01 * /
 const osThreadAttr_t blink01_attributes = {
   .name = "blink01",
   .priority = (osPriority_t) osPriorityNormal,
   .stack_size = 128
 };
 blink01Handle = osThreadNew (StartBlink01, NULL, & blink01_attributes);

/ * CODE UTILISATEUR BEGIN Header_StartBlink01 * /
/ **
  * @brief Fonction implémentant le thread blink01.
  * Argument @param   : Non utilisé
  * @retval Aucun
  * /
/ * USER CODE END Header_StartBlink01 * /
void StartBlink01 (void * argument)
{
 / * USER CODE BEGIN  5 * /
 / * Boucle infinie * /
 for (;;)
 {
   HAL_GPIO_TogglePin (GPIOA, GPIO_PIN_5);
   osDelay (500);
 }
```

StartBlink01 () est un exemple de thread qui devra fonctionner simultanément avec d'autres threads, avec leur propre code de configuration et des boucles while infinies.

Pour l'exécution:

Une fois le code est copié sur main.c on sauvegarde le projet et on génère le code, puis on clique sur le bouton droit de la souris sur notre projet dans "explorer projet" dans STM32CubeIDE, un menu s'affiche puis on clique sur build configuration ensuite on clique sur build all.

Un fois terminée et aucune erreur affichée, on clique toujours avec le bouton droit de la souris sur notre projet comme indiqué ci-dessous, et on clique sur Run as et puis on clique sur STM32 Cortex-M C/C++ application et puis on clique sur OK.

Remarque: il faut que la carte Nucléo STMF411RE soit branchée et installée avec ses pilotes qui sont accessibles depuis le site www.st.com

Maintenant on doit voir le voyant LD2 clignoter, mais à un cycle de service variable qui augmente ou diminue par étapes distinctes. C'est le résultat des threads avec des temps d'attente différents.

Chapitre IV : Disjoncteur électronique pour la protection des équipements et des personnes

1 Le Réseau Electrique Basse Tension (domestique) et ses disfonctionnements:

L'énergie électrique en Algérie est une énergie vitale dont la plupart des citoyens ne peuvent se passer, étant donné que la seule ressource énergique utilisée pour l'alimentation de la plupart des équipements domestiques et pour l'éclairage.

Pour alimenter un consommateur en énergie électrique, quelques règles simples permettent de connaître la tension de livraison du réseau : Lorsque la puissance n'excède pas 40kVA, le client est raccordé au réseau de distribution basse tension BT 230/400V. Pour les puissances supérieures à 40kVA, le client est alimenté par une alimentation triphasée HTA sans neutre dite de 2éme catégorie, comprise entre 5.5kV, 10kV et 30kV (généralement 30 kV). Les plus gros consommateurs d'énergie électrique (grandes usines, sidérurgie,...) sont alimentés directement à des réseaux haut tension de transport HTB (60kv, 220kv ou 400kv).

La consommation en basse tension qui représente la plus grande tranche de consommation de l'énergie électrique et qui est en forte progression ces dernières années, est par moments () et en certains lieux instable, ce qui provoque des disfonctionnements néfastes pour les équipements électriques. Parmi ces disfonctionnements, on pourra citer :

Les micro coupures :

Le fonctionnement des appareils électriques et électroniques, en particulier des ordinateurs, peut souffrir de perturbations en cas de microcoupures. Même s'ils sont en principe conçus pour résister à ces phénomènes, les appareils peuvent subir des dommages en cas de microcoupures nombreuses et répétées.

Sur tension :

Quand il y a une tension supérieure à la normale, presque la totalité du matériel qui fonctionne avec de l'électricité risque d'être endommagé, pire encore des vies humaines sont mis en danger car les surtensions peuvent occasionner des dégâts sur les circuits électriques et causer des incendies.

Sous tension :

Moins dangereuse que la surtension, une tension au dessous de la normale peut provoquer le disfonctionnement ou l'arrêt des appareils qui fonctionne avec de l'électricité.

2 Conception du disjoncteur électronique

A cet effet et pour essayer de remédier à ces problèmes épineux et en remplacement du disjoncteur différentiel classique utilisé dans les installations électriques domestiques, nous avons pensé à concevoir un système qui a pour objectif "la surveillance du réseau électrique domestique pour la protection des équipements matériels et des personnes" et pour cela nous proposons le schéma synoptique suivant :

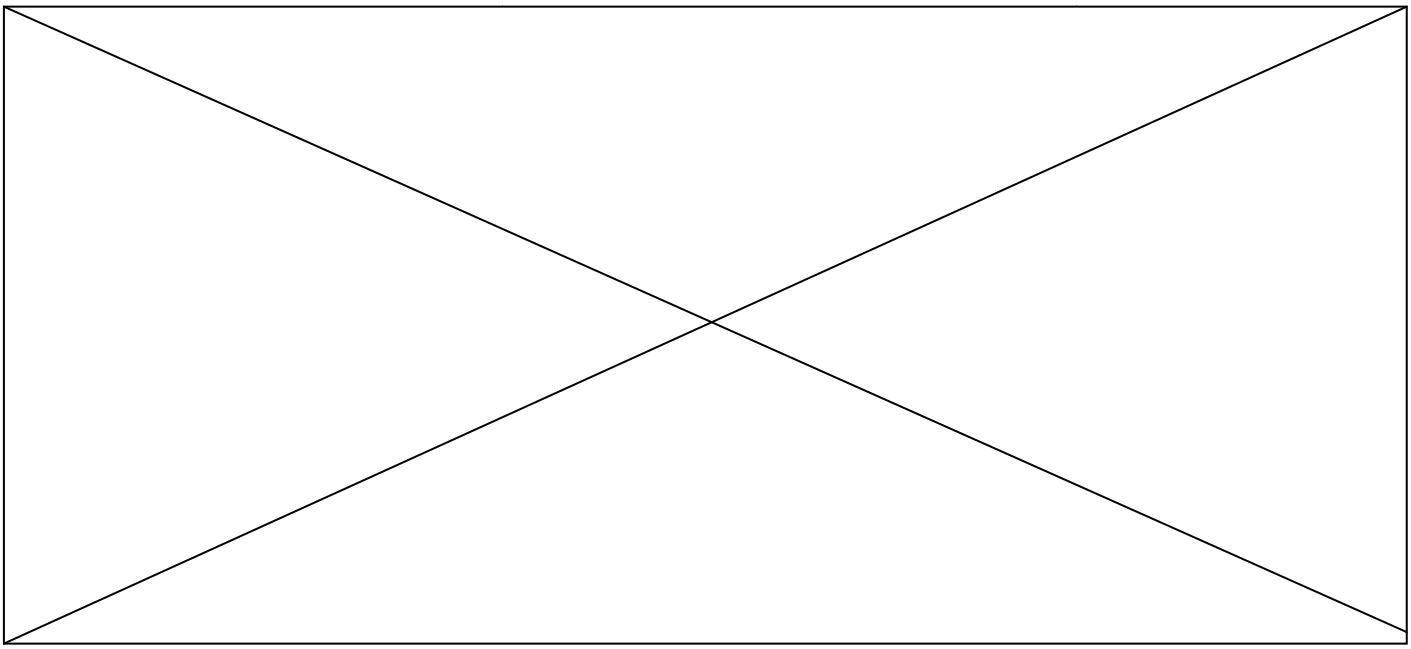

Figure 15 : Structure du disjoncteur électronique

2-1 Description et fonctionnement du système:

Dans le système que l'on se propose de concevoir (Fig. 15), le microcontrôleur (MC) acquière en temps réel les valeurs de la tension du réseau et du courant consommé par l'abonné.

Le microcontrôleur fermera d'une manière automatique l'interrupteur (Triac ou deux Thyristors montés tête bèche) pour l'alimentation des équipements en l'absence de disfonctionnement (Tension du réseau correcte).

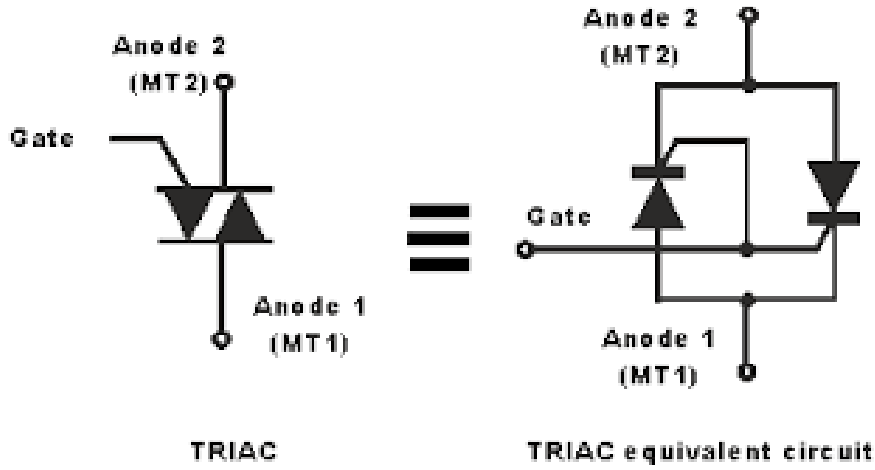

Figure 16: Symbole du Triac

En cas de surtension, sous-tension, coupures intempestives, dépassement d'une limite maximale du courant (réglable et mise à 30A par défaut), d'un différentiel entre le courant de sortie et d'entrée dépassant certain seuil (programmable), le système va arrêter l'alimentation jusqu'à ce que le réseau se stabilise et/ou que le disfonctionnement disparaisse. La mesure des courants se fera par l'utilisation de deux capteurs à effet Hall.

Le capteur de courant à effet Hall est un type de capteur de courant exploitant l'effet Hall pour produire une tension qui est l'image exacte (avec un facteur de proportionnalité connu) du courant à mesurer ou à visualiser.

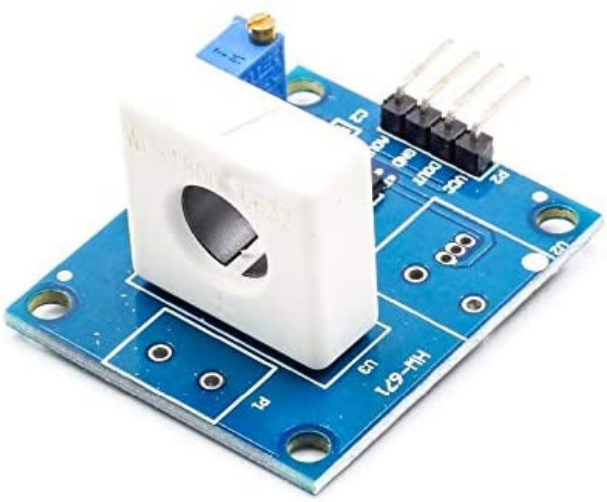

Figure 17: Exemple de capteur à effet Hall pour la mesure de courant alternatif.

De plus, le système affichera sur écran LCD, divers paramètres du réseau telle la tension, consommation instantanée, ...

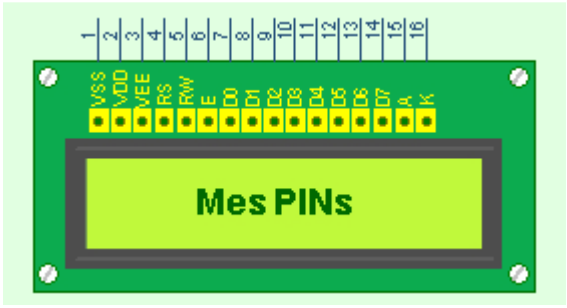

Figure 18: Afficheur LCD

Nous proposons alors, pour la réalisation d'un tel système, le schéma électrique qui suit qui s'architecture autour une carte Nucléo STM32F411RE qui embarque un microcontrôleur 32 bit de référence STM32F411RET6

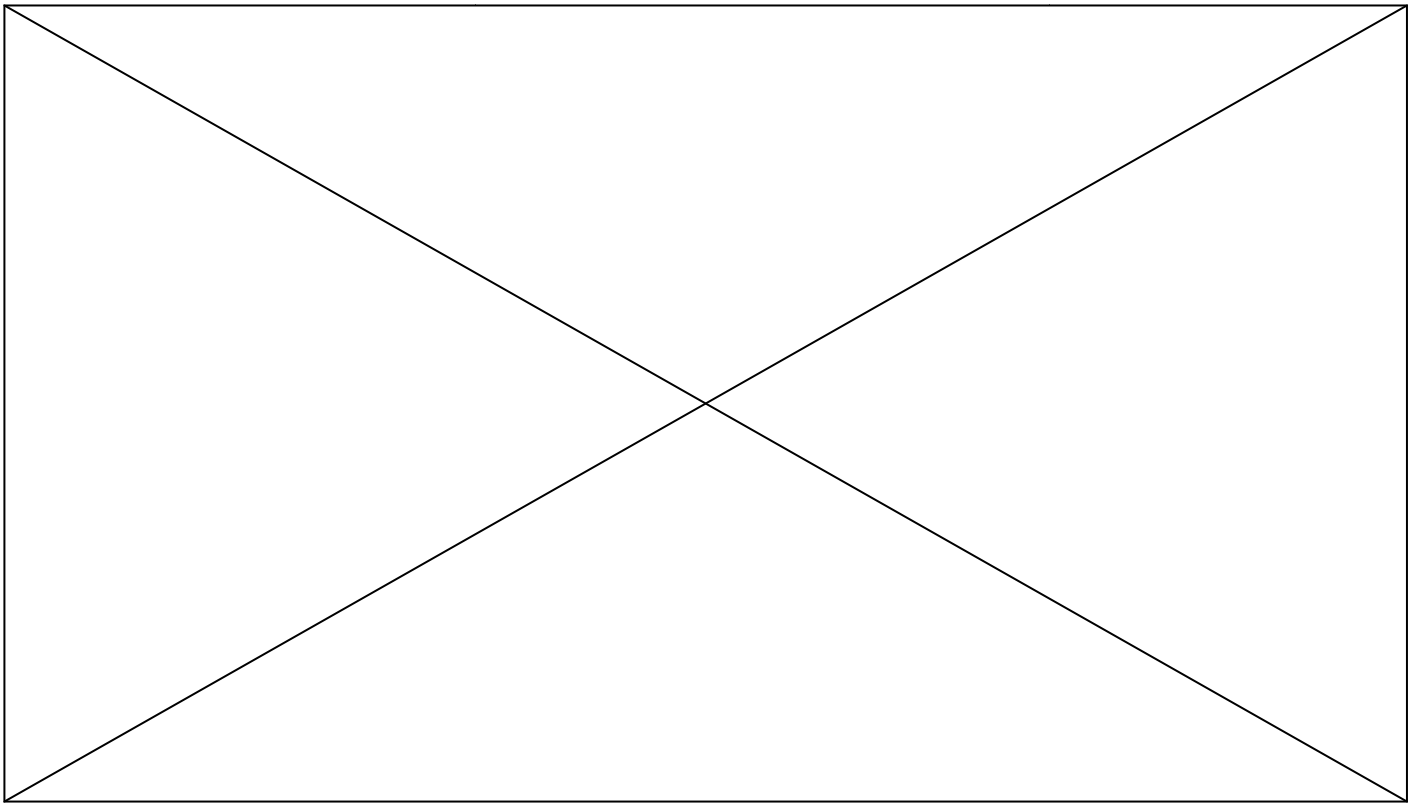

Figure 19: Schéma électrique

Pour l'acquisition de la tension du réseau, nous avons utilisé un transformateur 220/6V en cascade avec un pont de diode et une capacité de filtrage. Ce circuit va nous délivrer une tension continue proportionnelle à la tension du réseau. Via un potentiomètre, le microcontrôleur va acquérir une partie de cette tension en utilisant l'un de ses convertisseurs analogique numérique CAN. Pour l'acquisition du courant, des capteurs de courant en sortie de l'interrupteur seront utilisés pour mesurer, en utilisant éventuellement les CAN du microcontrôleur, les valeurs du courant de sortie et celui du courant d'entrée.

Un régulateur de tension de type 7805 sera aussi utilisé pour l'alimentation de notre système.

Pour l'affichage de divers informations sur l'état du réseau et de notre système, on se propose d'utiliser des LED et/ou un afficheur LCD qui vas nous afficher par exemple la consommation en temps réel ainsi que diverses informations sur le réseau : tension du réseau, différentiel entre courant de sortie et courant d'entrée, valeur du courant, puissance instantanée, ...

Pour faire fonctionner notre matériel, nous allons embarqué dans notre microcontrôleur un RTOS qui aura à gérer les différentes taches qu'il aura à accomplir. Pour

cela nous avons opté pour FreeRTOS. Ces taches, qui s'exécuteront en parallèle, rempliront chacune des fonctionnalités de surveillance et de protection de notre système et sont :

- Fonction surveillance de la tension du réseau
- Fonction différentielle qui sera de priorité la plus élevée
- Fonction affichage de priorité la plus basse

Au démarrage du système, une fonction d'initialisation du système sera exécutée.

2-2 Les différentes tâches:

Initialisation du système:

Au lancement de notre système (Figure 20 : Initialisation.), le microcontrôleur commencera par l'initialisation des différents périphériques que l'on utilisera, tel les GPIO, CAN et TIMERs.

La variable DDef , mise à zéro par défaut au démarrage du système, sera mise à 1 par la fonction différentielle quand cette dernière coupe le courant. Ceci empêchera la fonction de surveillance du réseau de rétablir le courant tant que cette variable ne sera pas remise à zéro par l'appui de l'abonné sur un bouton de remise en service du disjoncteur.

Par défaut, l'interrupteur est ouvert à la mise en service du système.

Le lancement des taches ne se fera qu'après une temporisation de quelques secondes pour éviter les microcoupures.

Ces taches seront programmées pour être exécutées par le l'ordonnanceur de FreeRTOS toutes les 20 ms. Le choix de ce temps sera reconsidéré après la réalisation d'un prototype du système, son expérimentation et des recherches plus poussées.

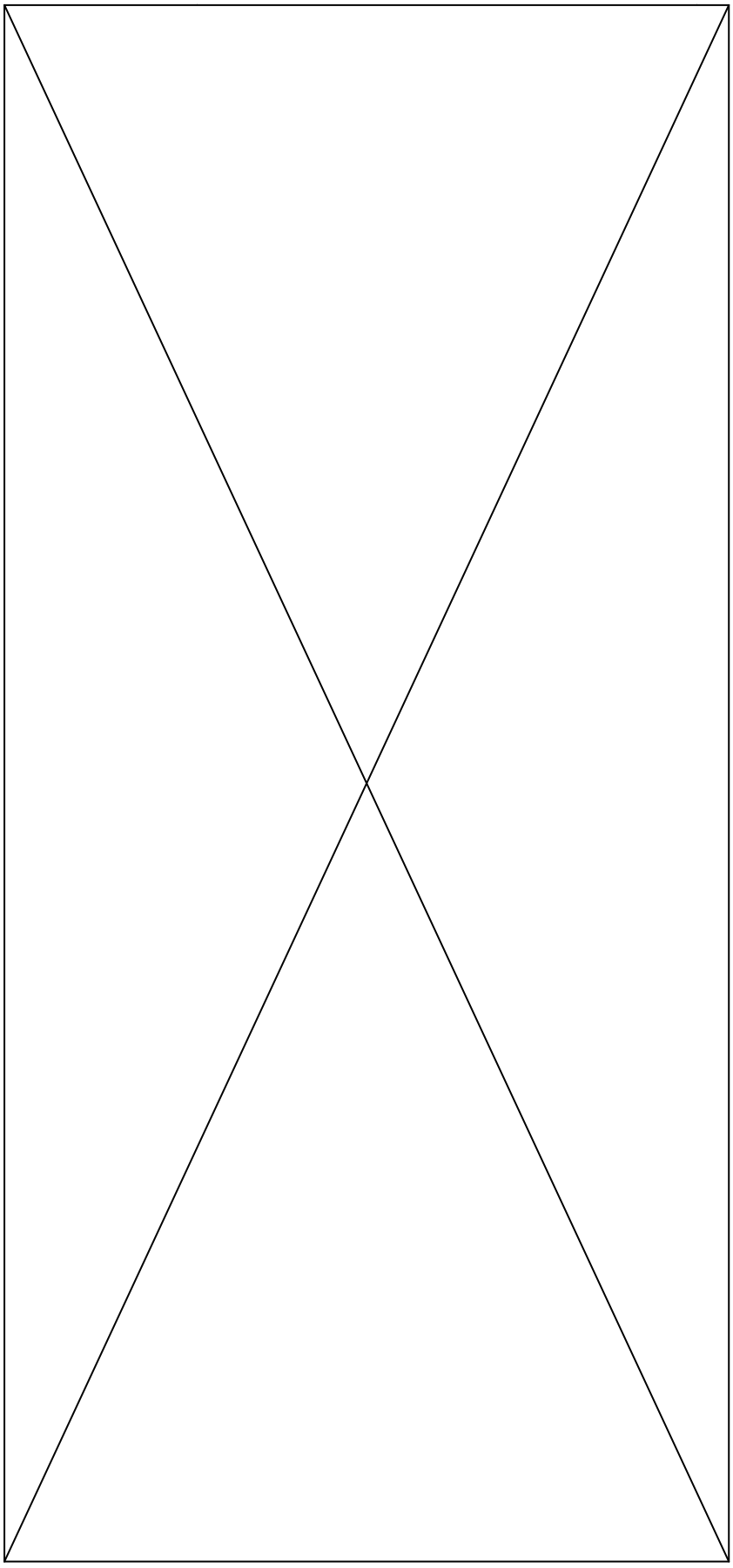

Figure 20 : Initialisation.

Surveillance de la tension du réseau électrique :

Cette tâche consiste à surveiller en temps réel la tension du réseau. En cas où cette dernier sort d'un certain intervalle [180V - 230 V] , le système va arrêter l'alimentation de l'abonné. Pour éviter les microcoupures, le courant ne sera rétabli qu'après stabilisation du réseau et une temporisation de quelques secondes.

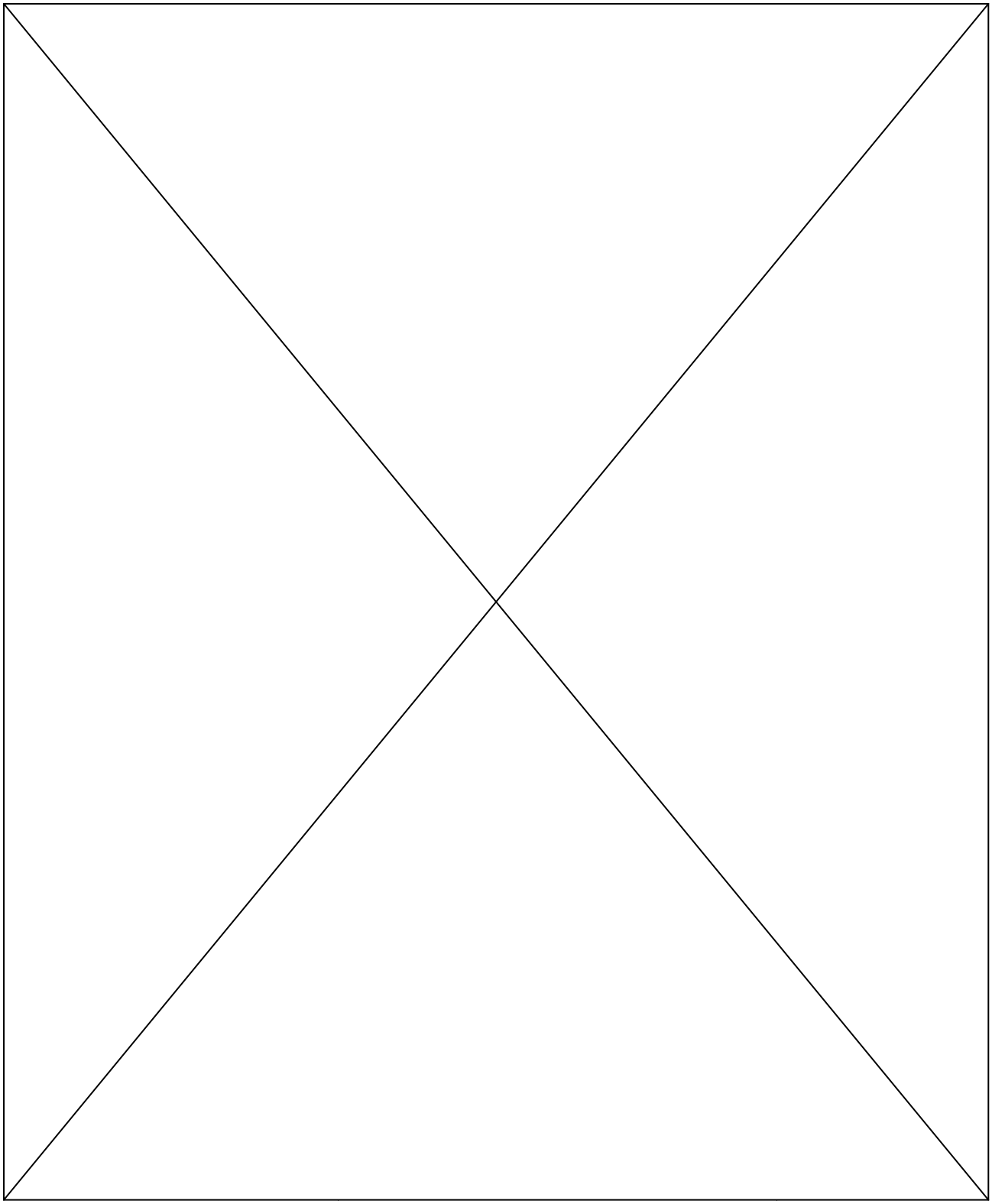

Figure 21 : Surveillance du réseau.

Fonction différentielle :

Cette tâche permet de surveiller la différence entre le courant sortant et le courant entrant. Si cette différence dépasse un certain seuil (programmable), le système disjoncte et ne se remettra en fonctionnement que sur intervention de l'abonné (Appui sur un bouton). Cette fonctionnalité permet une protection des personnes de l'électrocution si le seuil choisi est de l'ordre de 30 mA.

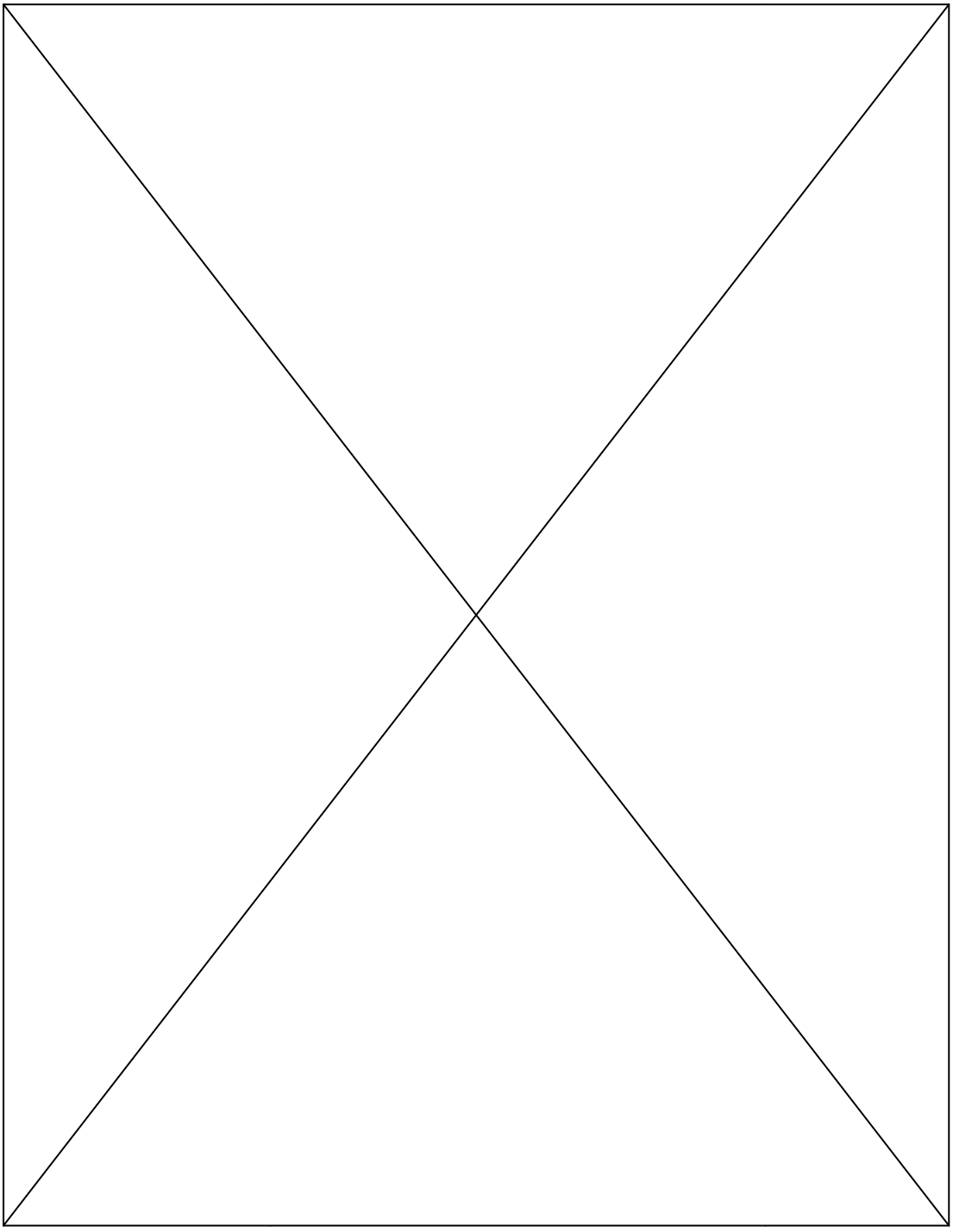

Figure 22 : Fonction différentielle

Pour remettre en service le disjoncteur, l'abonné devra appuyer sur le bouton RST.

Le bouton de remise en service (RST) va déclencher une interruption qui remettra DDef à 0.

La fonction de surveillance du réseau rétablira alors le courant si la tension du réseau est correct.

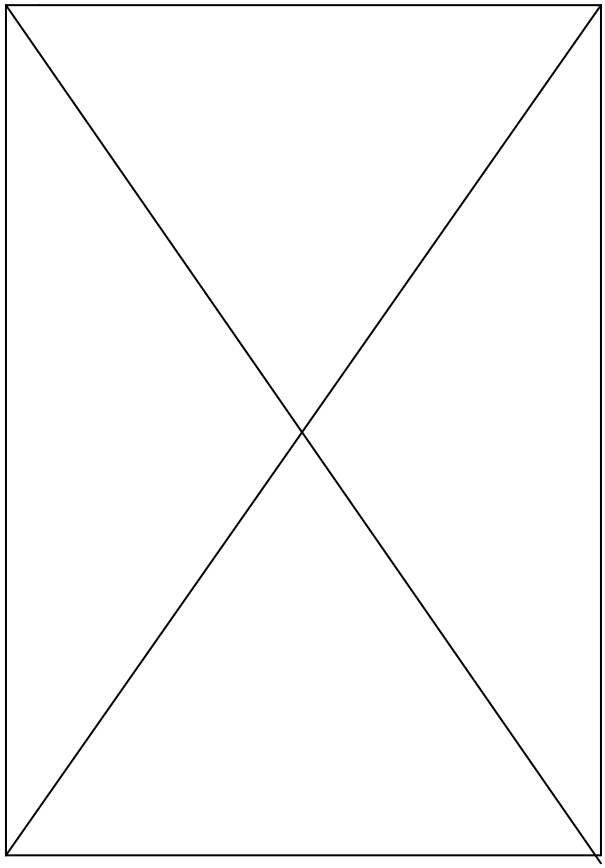

Figure 23 : Remise en service du courant électrique.

Fonction Affichage:

Cette tâche permettra, en utilisant les valeurs de la tension et du courant acquises d'afficher sur un écran LCD tout un ensemble d'informations sur l'état du réseau et la consommation de l'abonné. L'utilisation d'un ensemble de boutons permettra d'interagir avec le système pour choisir ce qui sera affiché et d'introduire certains paramètres de réglage du système comme la valeur de seuil du différentiel.

Conclusion

Nous avons conçu un système temps réel de surveillance du réseau électrique domestique pour la protection des équipements et des personnes. Ce disjoncteur électronique qui embarque une RTOS, en plus de remplir le rôle d'un disjoncteur différentiel classique, va protéger les équipements domestiques contre d'éventuelles surtensions, les microcoupures, dépassement du courant maximal (court circuit), ...

De plus par son système d'affichage sur LCD, l'abonné aura accès à plusieurs informations sur l'état du réseau et sa consommation.

Des améliorations peuvent être apportées à notre système pour en faire un IOT qui permettra notamment de prévenir l'abonné à distance d'éventuel disfonctionnement ou coupure de courant.

Bibliographie :

source: http://fr.wikipedia.org/wiki/Système_embarqué [1]

Les systèmes embarqués introduction Etienne Messerli Le 20 février 2019 [2]

Les systèmes embarqués introduction heigh-vd Eole d'inginerie et de gestion du canton de vaud [3]

https://perso.telecom-paristech.fr/pautet/seti/supports/rt-intro.pdf [4]

https://fr.wikipedia.org/wiki/Syst%C3%A8me_temps_r%C3%A9el [5]

https://en.wikipedia.org/wiki/ChibiOS/RT [6]

https://en.wikipedia.org/wiki/FreeRTOS [7]

https://o10ee.com/lepton/#toggle-id-4 [8]

A. Eswaran, A. Rowe et R. Rajkumar, «Nano-RK: Un système d'exploitation axé sur les ressources et sensible à l'énergie pour les réseaux de capteurs», IEEE Real-Time Systems Symposium, décembre 2005. [9]

www.pragmatec.net [10]

https://beagleboard.org/project/TUD%3AOS+on+the+Beagleboard/ [11]

http://www.tnkernel.com/tn_description.html [12]

https://mynewt.apache.org/about/ [13]

https://en.wikipedia.org/wiki/NuttX [14]

https://www.osrtos.com/rtos/rt-thread/ [15]

https://www.linuxjournal.com/content/ecoscentric-limiteds-ecospro-0 [16]

https://fr.wikipedia.org/wiki/Liste_des_syst%C3%A8mes_d%27exploitation_temps_r%C3%A9el [17]

Téléchargez STM32CubeIDE: https://www.st.com/en/development-tools/stm32cubeide.html[18]

https://www.st.com/resource/en/user_manual/dm00173145.pdf [19]

Référence API FreeRTOS: https://www.freertos.org/a00106.html

Référence API CMSIS-RTOS: http://www.keil.com/pack/doc/CMSIS/RTOS/html/group__CMSIS__RTOS.html [20]

Vue d'ensemble des offres STM32 de DigiKey: https://www.digikey.com/en/product-highlight/s/stmicroelectronics/stm32-overview

Offres Nucleo de DigiKey: https://www.digikey.com/en/product-highlight/s/stmicroelectronics/nucleo-development-boards

Glossaire:

API : Application Programming Interface
ADC : application delivry controller
ADC : Analog to Digital Converter
ASIC : application-specific integrated circuit
A: Amper
CPU: central processing unit
CTN : coefficient de température négatif
CRC : Cyclic Redundancy Check
DAC: Digital Analog Converter
DSP: Digital Signal Processor
GPIO: General Purpose Input/output
IOT : Internet of Things
IDE : Integrated Development Environment
IHM : interactions personne-machine
KVA : kilo voltampère
KV : Kilovolt
KW : kilowatt
LCD : liquid crystal display
LED : light-emitting diode
LoWPAN: Low power Wireless Personal Area Networks
MA : Milliamp
MC : Microcontroller
MMU: memory management unit
MS : Millisecond
MD2 : Message Digest 2
NB-IoT: Narrowband Internet of Things
NTC: New Technology Center
OS : Operating System
OSEK :Open Systems and their Interfaces for the Electronics in Motor Vehicles
PLL: Phase-locked loop
RISC : Reduced Instruction Set Computer
RTOS : real-time operating system
RLE : Run Length Encoding
SDIO: Secure Digital Input Output
SoC : system on a chip
SWD : Serial Wire Debug
SWIM : single wire interface module
TCP : Transmission Control Protocol
USART : Secure Digital Input Output
V : Volt