الجـمهوريـة الجـزائريـة الديمقـراطية الشعبيـة
People's Democratic republic of Algeria
وزارة التعليـم العالي و البحث العلـمي
Ministry of Higher Education and Scientific Research
جـامعـة عبد الحميـد بن باديـس – مستغانـم
Abdel Hamid Ibn Badis University – Mostaganem
كليـة العلـوم و التكنولـوجـيا
Faculty of Sciences and Technology
قسم هنـدسـة كهربائية
Department of electrical engineering

N° d'ordre : M2……./GE/2022

Mémoire de fin d'étude
Présente pour obtenir le diplôme de
Master en Génie Electrique

Filière : Electronique.

Option: Electronique des systèmes embarqués

Par:

BELLAKEHAL SID AHMED

BENDEHIBA SID AHMED

Thème

Mise en œuvre de freeRTOS pour la réalisation d'un capteur-Transducteur-actionneur multitâche

Soutenu le 18/09/ 2022 devant le jury composé de :

Président(e) :	Mr wissam Benstali	Pr.	Université de Mostaganem
Examinateur (rice) :	Mm Khadidja Berdja	Pr	Université de Mostaganem
Encadreur (e) :	Mr Nasreddine Abdellaoui	MAA	Université de Mostaganem

Année Universitaire 2021/2022

Dédicaces

Remerciements

Table des matières

DEDICACE

Je dédie ce précieux travail aux êtres les plus chers

monde, à qui je témoigne mon amour et mon affection pour

leur encouragement, leur compréhension et leur patience,

qui ont su me comprendre et m'ont poussé à apprendre,

c'est de vous dont je parle très

Chers parents.

A mes sœurs, Yemna . Sabrina.. Marwa .Fatima

A mon frère, Adem

Et toute la famille « BELLAKEHAL » sans exception.

A ma fiancée MEDJDA qui m'ont toujours soutenu,

Sans omettre BENDHIBA SID AHMED avec qui j'ai

élaboré mon projet de fin d'étude.

En fin à tous ceux qui m'apprécient à ma juste valeur.

RMERCIEMENTS

Je commence par le remerciement du bon "DIEU" pour la santé, la volonté et la Patience qui m'a donné pour accomplir ce modeste travail.

Ce mémoire a pu, être mené à terme grâce à ceux qui ont, de loin ou de près, voulu nous apporter leurs aides et conseils.

Je remercie en particulier: mon père, ma mère, mes frères et soeurs.

Aussi, il nous est particulièrement agréable d'adresser nos remerciements :

A nos encadreurs:

- *Monsieur Nasreddin Abdellaoui, qui n'a cessé de nous guider tout au long de l'élaboration de ce mémoire. Ce travail n'a pas reçu la dimension qu'imposent les jets sans son intervention.*

Aux membres de jury en personne de:

- *Monsieur le président Benstali Wissam*

- *Monsieur l'examinateur Rebhi Mustapha*

J'espère qu'ils trouveront ici la profondeur de ma reconnaissance qui a jugé.

Je remercie tous ces membres de ma famille, ainsi que mes amis et collègues qui m'ont beaucoup aidée t'en courage dans mes études.

RESUME

L'utilisation d'un capteur intelligent et simple dans l'enseignement de la physique se développe rapidement. Le but de ce mémoire est de décrire le développement de capteurs, actionneurs et transducteurs multitâches simples. Des capteurs, actionneurs et transducteurs multitâches d'instrumentation ont été développés sur la base d'Arduino mega 2560 avec FreeRTOS_AVR. Les commandes et les données sont transmises et reçues sans fil en série par Arduino et un ordinateur portable via le module Bluetooth. Ensuite, les résultats sont affichés sous forme d'icônes graphiques, et d'indicateurs visuels réalisés à l'aide de Processing 3. Il permet à l'environnement d'interagir avec le système. Les résultats montrent que l'évaluation visuelle a été confirmée avec le système fonctionnant comme prévu. En outre, le code logiciel développé s'est avéré adapté à l'installation sur l'Arduino MEGA2560. Mots clés multitâche, capteurs, actionneurs, transducteurs, ArduinoMEGA2560, FreeRTOS_AVR, Processing 3

Abstract

The use of a smart and simple sensor in physics education is increasing rapidly. The purpose of this memory is to describe the development of simple multitasking sensors, actuators, and transducers. Instrumentation multitasking sensors, actuators, and transducers have been developed based on Arduino mega 2560 with FreeRTOS_AVR. Commands and data are transmitted and received wireless serially by Arduino and laptop through the Bluetooth module. Then, the results are displayed in the form of graphical icons, and visual indicators made using Processing 3. It allows the environment to interact with the system. The results show that visual assessment has been confirmed with the system performing as designed to do. Besides, the developed software code is proven suitable for installation on the Arduino MEGA2560. Keywords multitasking, sensors, actuators, transducers, ArduinoMEGA2560, FreeRTOS_AVR, Processing 3

الملخص

يتزايد استخدام جهاز استشعار ذكي وبسيط في تعليم الفيزياء بشكل سريع. الغرض من هذه مذكرة هو وصف تطوير أجهزة استشعار بسيطة متعددة المهام، ومشغلات، ومحولات طاقة. تم تطوير أجهزة الاستشعار متعددة المهام والمحركات والمحولات على أساس mega Arduino مع AVR_FreeRTOS 2560 .يتم إرسال الأوامر والبيانات واستلامها السلكي عن Arduino والكمبيوتر المحمول من خالال وحدة Bluetooth .بعد ذلك ، يتم عرض النتائج طريق في شكل أيقونات رسمية ، ومؤشرات بصرية مصنوعة باستخدام معالجة 3 .وهي تسمح للبيئة بالتفاعل مع النظام. تظهر النتائج أنه تم تأكيد التقييم البصري مع أداء النظام على النحو المصمم للقيام به. إلى جانب ذلك ، ثبت أن الكلمات المفتاحية ArduinoMEGA2560المطور مناسب للتثبيت على MEGA2560 Arduino.كود البرنامج المعالجة، FreeRTOS_AVR 3تعدد المهام ، أجهزة الاستشعار ، المحركات ، محولات الطاقة ، ،

Introduction générale

Des cartes à microcontrôleur(s) équipées de capteurs pour mesurer la température, l'humidité relative, l'intensité lumineuse, la distance et d'autres propriétés physiques, sont des systèmes embarqués souvent utilisés dans de nombreuses expériences de physique éducative, dans des applications d'automatisation industrielle ainsi que dans d'autres domaines.

Certains de ces cartes à microcontrôleur(s) sont utilisés avec des actionneurs. D'autres sont utilisés comme transducteurs.

Quand ces systèmes contiennent un capteur qui peut effectuer une tâche simple, les séquences de contrôle ne posent aucun problème. En revanche des difficultés appréciables commencent à s'accumuler à mesure que la nécessité d'exécuter plusieurs tâches devient évidente : En effet, le délai nécessaire requis par une tâche empêchera le microcontrôleur de recevoir les signaux des autres capteurs pendant ce délai. Par exemple, si plusieurs capteurs doivent être connectés avec des servomoteurs et d'autres sur une seule carte à microcontrôleur dans une application donnée pour mesurer la température, l'humidité et les distances, un système de contrôle approprié doit être mis en place. Si ces tâches doivent être exécutées en utilisant le modèle de programmation à boucle unique, il sera difficile de contrôler une tâche avec l'intervalle de temps précis.

En revanche, une carte à microcontrôleur équipé d'un système d'exploitation en temps réel (RTOS) permet de surmonter cette limitation : un RTOS permet une réaction ultérieure dans la contrainte de temps imposée. Il fait basculer l'exécution d'une tâche à une autre en s'assurant que chaque tâche dispose d'un temps de traitement adéquat tout en tenant compte des priorités des tâches.

C'est autour de ce contexte que se situe le sujet de notre projet. Il s'agit de réaliser un capteur-transducteur-actionneur mettant en œuvre un RTOS pour exécuter une application embarquée multitâches.

Le projet étant relativement complexe et nécessitant d'abord l'acquisition d'un minimum de compétences pour pouvoir le réaliser, nous avons adopté la démarche suivante: Dans un premier temps, une recherche bibliographique approfondie a été effectuée sur des projets similaires. Nous avons par la suite, sélectionné l'approche de

conception de notre application, ensuite, nous avons fait les choix quand aux « outils de conception » , « langages et environnement de développement ».

Ceci étant dit, le mémoire de notre projet est organisé en cinq chapitres. Le premier est consacré aux vocabulaires et à la présentation de toutes les notions relatives aux capteurs, transducteurs et actionneurs, le deuxième à l'architecture matérielle et logicielle d'un capteur-transducteur-actionneur multitâches, le troisième aux différentes approches de conceptions des applications embarquées multitâches de manière générale, le quatrième à la réalisation matérielle et le cinquième à la réalisation logicielle.

Chapitre 1 : Généralités sur les capteurs, transducteurs et actionneurs

1.1. Définition d'un capteur

Un **capteur** est un dispositif transformant l'état d'une grandeur physique observée en une grandeur utilisable, telle qu'une tension électrique, une hauteur de mercure, une intensité ou la déviation d'une aiguille. Il ne peut être utilisé que comme entrée d'un système.

Exemples : Capteur de température lm35, capteur d'humidité dht11 (Figure 1.1)

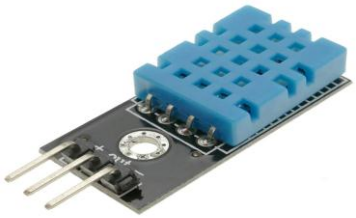

Figure 1.1 : quelques exemples de capteurs

1.2. Définition d'un capteur intelligent

Un capteur intelligent est un capteur analogique ou/et numérique ou/et TOR muni d'un microprocesseur ou microcontrôleur permettant de le doter de fonctionnalités comme la communication, le traitement du signal, …etc.

1.3. Définition d'un transducteur

C'est un dispositif qui convertit une forme d'énergie en une autre forme et contrairement à un capteur, on peut rencontrer des transducteurs utilisés comme entrée et d'autres comme sortie d'un système.

Exemples : ampoule à incandescence, microphone, moteur électrique (figure 1.2).

Figure 1.2 : Quelques exemples de transducteurs

Remarque : Tous les capteurs sont des transducteurs mais tous les transducteurs ne sont pas des capteurs.

Par exemple :

Le microphone : convertit le son en signaux électriques.

Le haut-parleur : convertit l'électricité en son.

Les deux sont des transducteurs mais seul le microphone est un capteur.

1.4. Définition d'un actionneur

C'est un dispositif qui convertit une forme d'énergie en une action(mouvement).

C'est un dispositif qui actionne ou déplace quelque chose. Un actionneur utilise de l'énergie pour fournir du mouvement. Par conséquent, un actionneur est un cas particulier de transducteur.

Il ne peut être utilisé que comme sortie d'un système.

Par exemple : Le moteur électrique

Remarque : tout comme il existe des capteurs intelligents, il existe des actionneurs intelligents et des transducteurs intelligents ou des combinaisons de ces 3 dispositifs.

Chapitre 2 : Architecture matérielle et logicielle d'un capteur-transducteur-actionneur multitâche

2.1. Introduction :

Une implémentation multitâche à proprement parler, ne peut reposer que sur un système d'exploitation supportant le multitâche, reposant lui-même, au plus bas niveau, sur une architecture matérielle permettant l'entrelacement temporel de plusieurs programmes la Figure 2.1 ci-dessous montre les différents composants que doit comprendre l'architecture matérielle et logicielle d'un C-A-T multitâches.

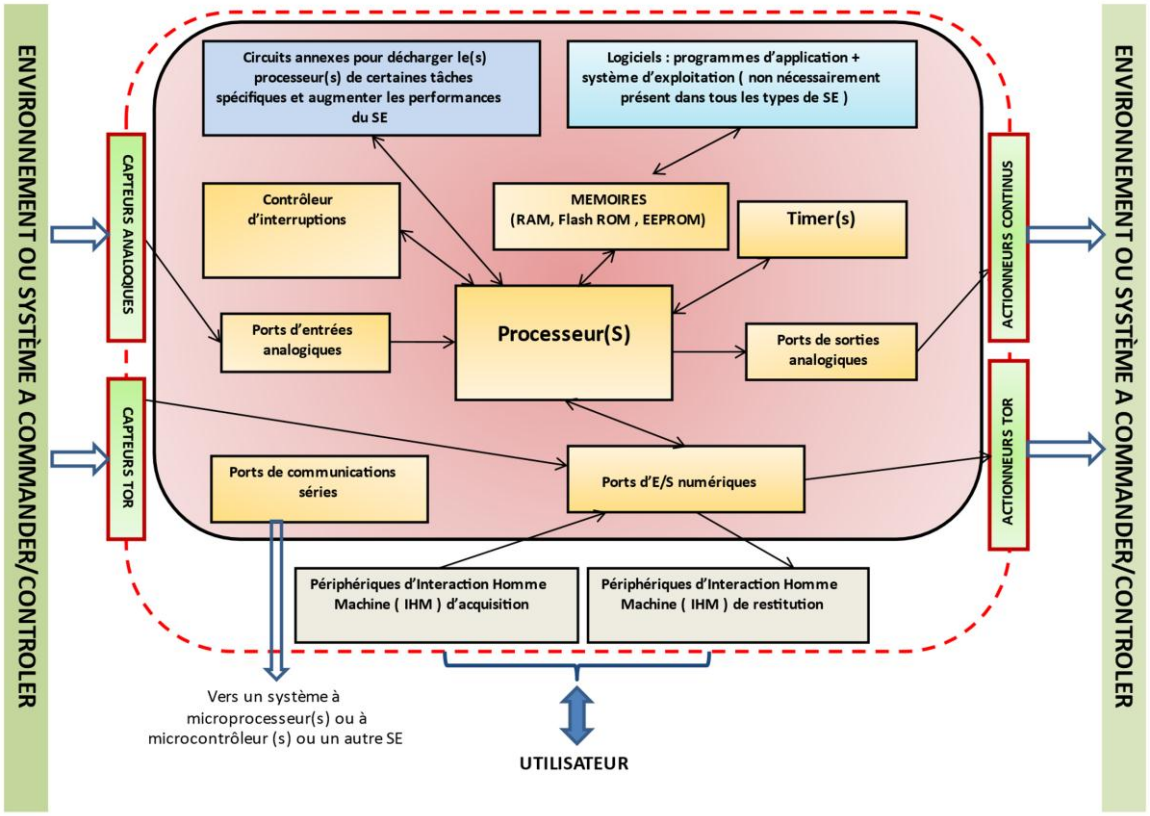

Figure2.1 : Les différents composants matériels et logiciels d'un C-A-T multitâches

Dans les paragraphes qui suivent, vont être discutés de manière succincte les points suivants : les différents types de système d'exploitation, avec ceux dédiés à l'embarqué ensuite l'architecture que doit avoir une application embarquée dans un C-T-A.

2.2. Les différents types de systèmes d'exploitation

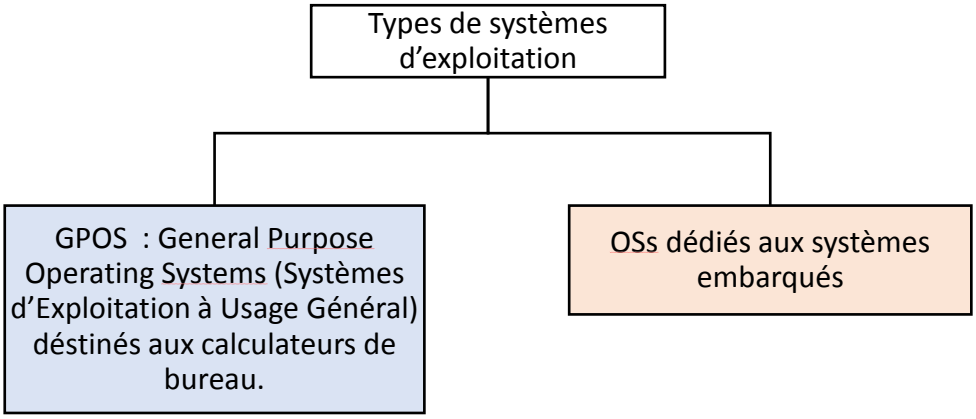

Figure 2.2 : Les différentes classes de systèmes d'exploitation

Exemples de systèmes d'exploitation temps réel dédiés à l'embarqué :

eCos, FreeRTOS, Linux embarqué, LiteOS, LynxOS, MenuetOS, NuttX, OS-9, PikeOS, QNX, RTEMS, RTLinux, RT-Thread, RTX, µC/OS-II, VxWorks, Zephyr.

2.3. Architecture de l'application embarquée dans un C-T-A multitâches

Le comportement concurrent des évènements et grandeurs physiques externes nous contraint à décrire l'environnement comme un système fortement parallèle. Ce qui conduit naturellement à ne voir comme architecture la mieux adaptée pour répondre à ce comportement parallèle de l'environnement externe qu'une architecture multitâche (Figure 2.3).

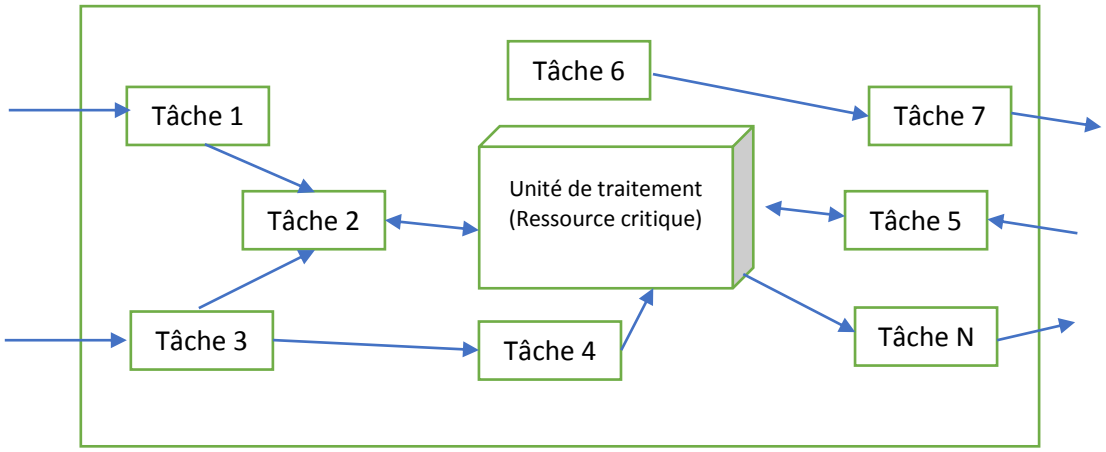

Figure 2.3. : Représentation schématique de l'architecture multitâche d'une application embarquée dans un C-A-T

Chapitre 3 : Approches de conception d'une application embarquée multitâches.

3.1. Introduction

En pratique, il existe plusieurs approches :

- La boucle principale plus routines de service d'interruption.
- Les ordonnanceurs coopératifs.
- Les RTOS avec leur ordonnanceur hybride (coopératif + priorisés et préemptifs).
- Les machines virtuelles, les machines d'état et la conception pilotée par des délais réactifs.

Dans les paragraphes qui suivent, nous allons nous limiter aux trois premières les autres étant relativement complexes à présenter.

3.2. Approche de la boucle principale plus routines de service d'interruption

Cette approche est standard pour les petites implémentations logicielles et constitue une bonne solution lorsqu'il n'y a pas besoin d'exécution parallèle bien séparée. Le parallélisme est obtenu grâce aux routines de service d'interruption (ISR), qui sont déclenchées par des interruptions logicielles ou matérielles. La boucle principale est une boucle infinie qui répète indéfiniment les mêmes activités. Son flux est suspendu lorsqu'une interruption se produit pendant que l'interruption est traitée par l'ISR. Le pseudo-code ci-dessous illustre cette solution.

```
variableGlobale1
variableGlobale2
…
variableGlobaleN
int main()
{
InitSystem() ;
For( ;;)
{
Tache1();
Tache2();
…
TacheN() ;
}
}
Isr1()
{
//……..
}
Isr2()
{
//……..
}
…
IsrN()
{
//……..
}
}
```

Algorithme: application embarquée développée à l'aide de l'approche « boucle infinie + services d'interruption »

Avantage : mise en œuvre facile.

Inconv1énients :

Il y a plusieurs problèmes avec cette approche :

Premièrement, l'itinérance conduit inévitablement à l'utilisation de variables à portée globale.

Deuxièmement, il est difficile de séparer l'application en modules indépendants qui ont leurs propres contraintes de temps. Les dépendances d'une telle implémentation sont trop nombreuses et les changements sont donc difficiles à maintenir.

Troisièmement, les ISR doivent s'exécuter rapidement et revenir une fois terminés. Cela crée des retards pour certaines activités. Le traitement réel doit se produire lorsque le contrôle revient à ce point dans la boucle principale. Selon l'instant où l'interruption s'est produite, un cycle de boucle complet peut s'écouler avant que l'action demandée par l'ISR ne soit exécutée par le logiciel de boucle principale. En d'autres termes, cette solution est la plus facile à créer, mais c'est le choix le moins flexible. Il convient aux petites applications bien définies où des modifications et une maintenance, minimales sont prévues tout au long de la durée de vie de la conception.

3.3. Approche basée sur les ordonnanceurs coopératifs

Les ordonnanceurs coopératifs sont implémentés le plus souvent et pour des raisons de facilité d'utilisation sous forme de librairies.

Avec cette méthode, les tâches d'une application coopèrent pour s'exécuter.

Cette coopération consiste en des appels à l'ordonnanceur pour lui rendre le contrôle afin de passer la main à une autre tâche.

Autrement dit, avec cette méthode, une tâche en cours d'exécution continue de s'exécuter tant qu'elle ne rend pas la main de façon explicite à l'ordonnanceur.

Le passage de la main à la tâche suivante est réalisé par un appel d'une fonction spéciale de l'ordonnanceur.

Avec cette méthode, le rôle de l'ordonnanceur se limite à démarrer les tâches et à leur permettre de lui rendre volontairement le contrôle pour passer la main à une autre tâche.

On donne à la figure 3.1 suivante l'organigramme d'une application embarquée réalisée à l'aide d'un ordonnanceur coopératif

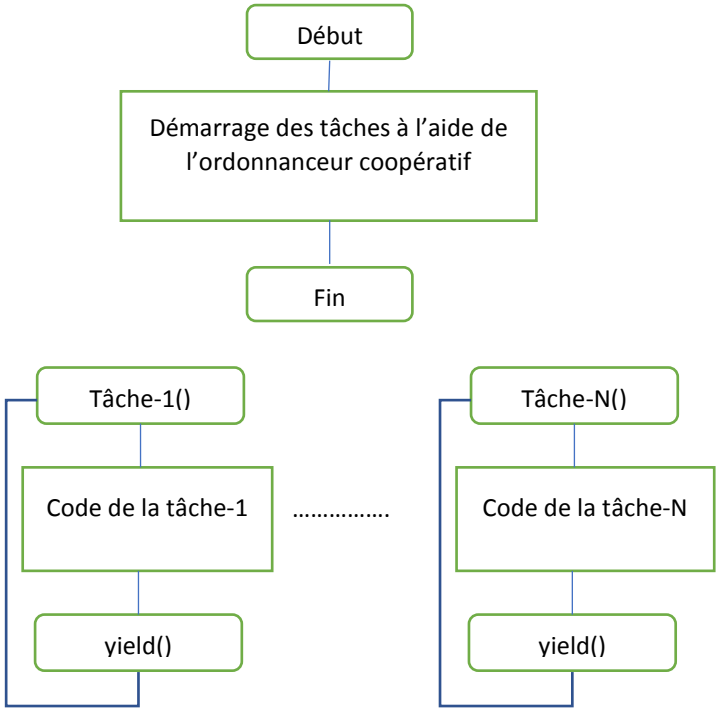

Figure3.1: Application embarquée développée à l'aide d'un ordonnanceur coopératif

Exemples d'ordonnanceurs coopératifs :

L'ordonnanceur coopératif « scheduler » de arduino.

On donne, un exemple d'implémentation en langage arduino :

```
// Inclusion de l'ordonnanceur multitache.
#include <Scheduler.h>
//Déclarations des variables globales et constantes

void setup() {
//Configuration du matériel

// Ajout de deux tâches "loop1" et "loop2" à l'ordonnanceur.
// "loop" est toujours lancée par défaut.

Scheduler.start(setup1,loop1);
Scheduler.start(setup2,loop2);
}
```

```
// tâche no.1
//Déclarations des variables globales et constantes
Void setup1(){
//Configuration du matériel
}

void loop() {
//traitement  de la tâche 1
Yield(); // pour passer la main à la tâche suivante
}


// tâche no.2
//Déclarations des variables globales et constantes
Void setup2(){
//Configuration du matériel
}

void loop2() {
//traitement  de la tâche 2
yield();// pour passer la main à la tâche suivante
 }
```

Exemple d'implémentation d'une application embarquée développée à l'aide d'un ordonnanceur coopératif, en langage Arduino

Avantages :

Le multitâche coopératif permet une mise en œuvre beaucoup plus simple des applications embarquées car leur exécution n'est jamais interrompue de manière inattendue par l'ordonnanceur.

Inconvénients :

Comme un système multitâche coopératif repose sur le fait que chaque tâche cède régulièrement du temps à d'autres tâches du système, une application embarquée mal conçue peut consommer tout le temps CPU pour elle-même, soit en effectuant des calculs étendus, soit en attendant occupée ; les deux entraîneraient le blocage de l'ensemble du système.

3.4. L'approche basée sur l'utilisation d'un RTOS

Cette approche facilite énormément le développement d'une application embarquée multitâches. En effet le développement se limite uniquement à la création des tâches et à leurs exécutions respectivement à l'aide de l'API et de l'ordonnanceur du RTOS qui le plus souvent se présente sous forme hybride (i.e : coopératif + priorisé et préemptif).

On donne à la figure 3.2 suivante l'organigramme d'une application embarquée réalisée à l'aide d'un ordonnanceur hybride.

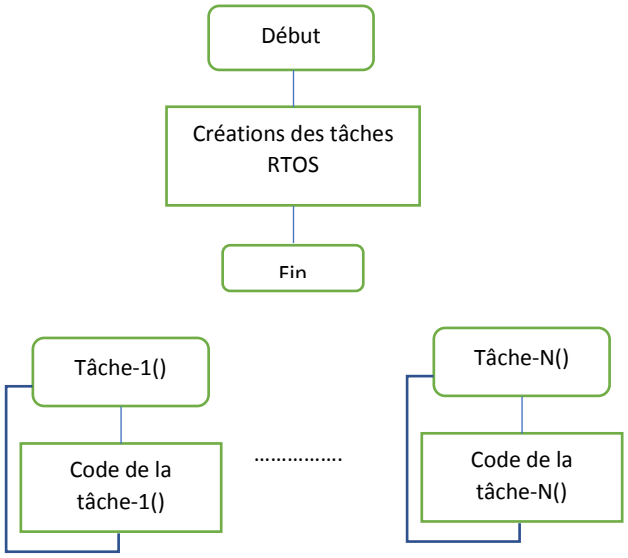

Figure 3.2 : Application embarquée développée à l'aide d'un RTOS

Avantages :

La réalisation d'une application embarquée à l'aide d'un RTOS, présente plusieurs avantages dont on peut citer entre autres :

- Concentration sur l'application, et non sur l'exécution des tâches.
- Gestion automatique du temps de CPU entre les tâches, suivant un critère de sélection prédéfini (e.g : priorité)
- Synchronisation de l'accès aux ressources
- Communication entre les tâches
- Ajout/retrait de tâches sans modifier l'application existante

Inconvénients :

Nécessite d'abord l'acquisition de compétences qui peut prendre quelques semaines voire quelques mois avant de pouvoir le déployer dans la réalisation d'une application embarquée.

Chapitre 4 : Réalisation matérielle

4.1. Matériel pouvant être utilisé pour réaliser un capteur-transducteur-actionneur.

On donne à la figure 4.1 suivante, une liste non exhaustive des différentes technologies pouvant être utilisées pour réaliser un capteur-transducteur-actionneur.

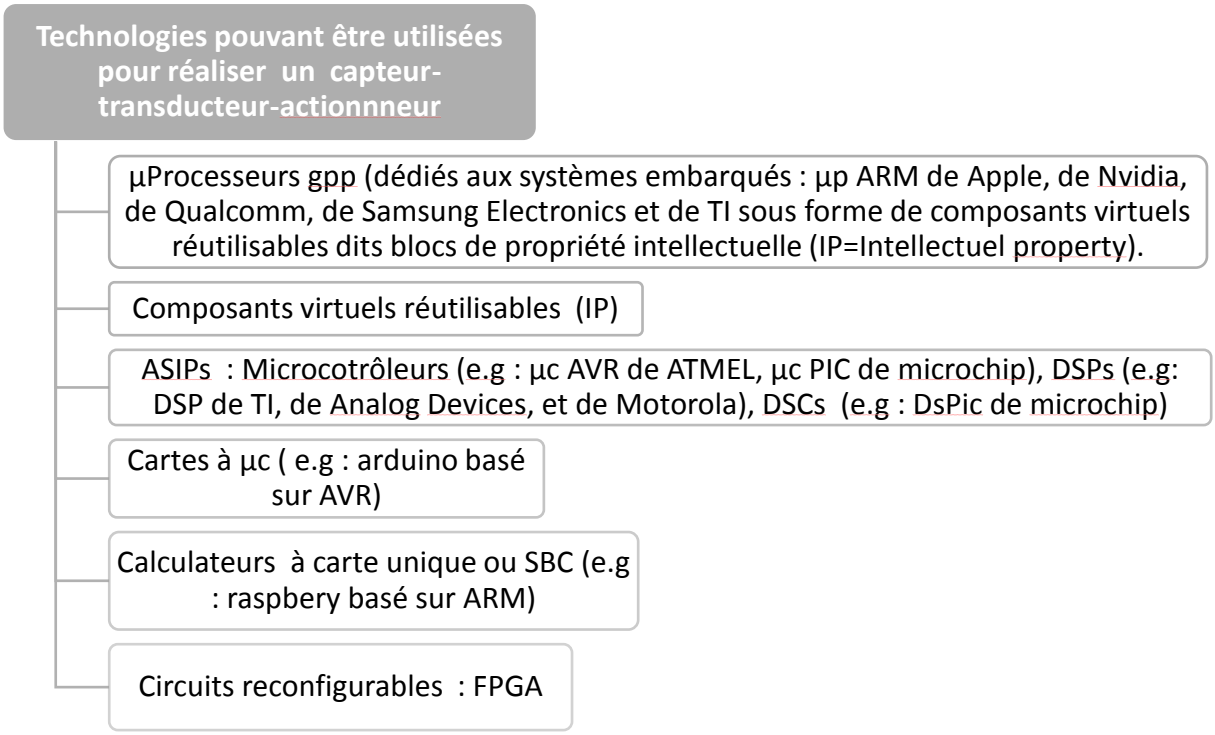

Figure 4.1 : Composants pouvant être utilisés pour réaliser un C-A-T

Dans notre projet, nous avons opté pour une carte à microcontrôleur : arduino mega

On donne à la figure 4.2… et à la figure 4.3, suivantes, respectivement le schéma bloc et le schéma électrique du C-A-T réalisé.

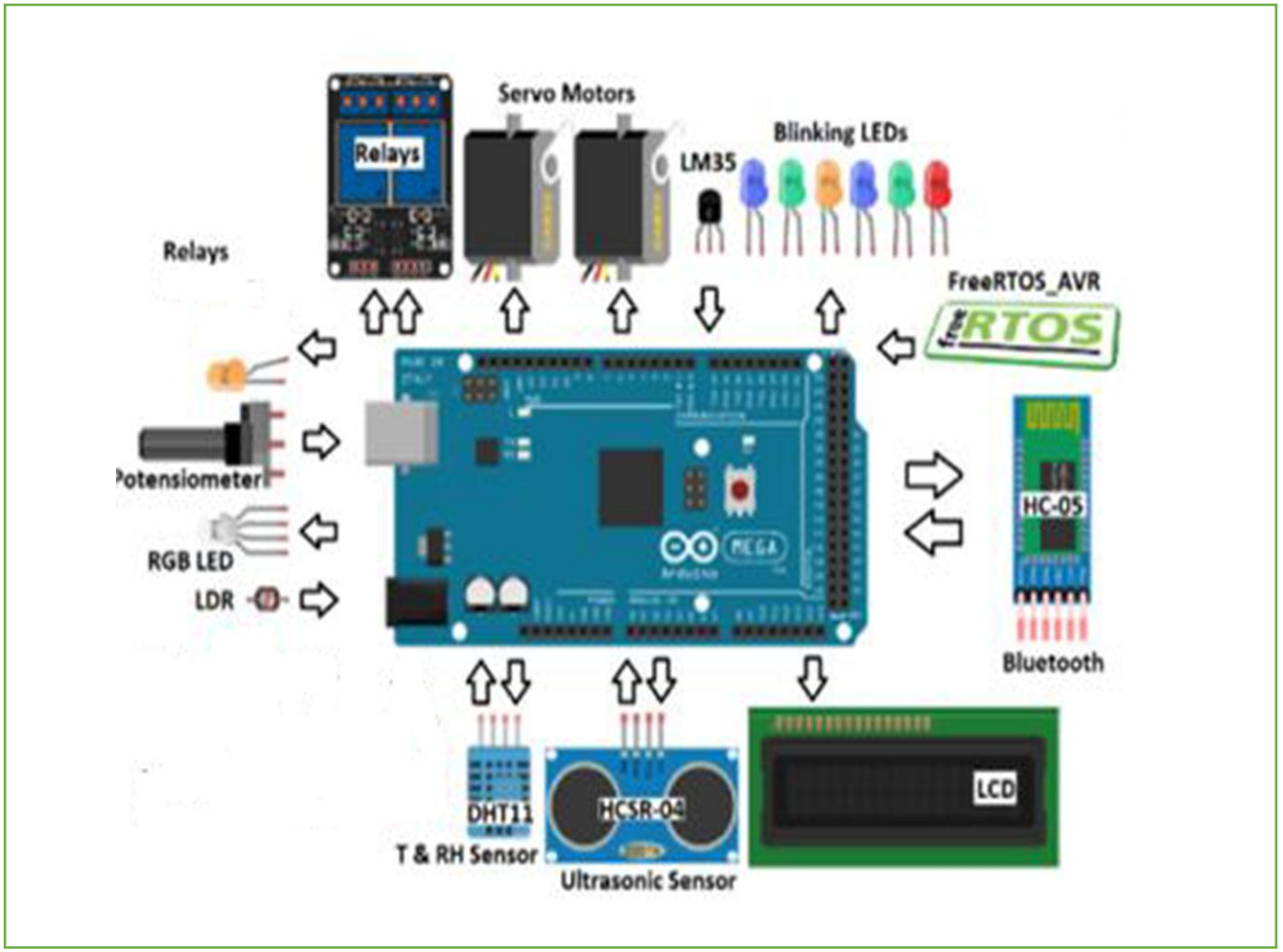

Figure 4.2 : Schéma bloc du C-A-T réalisé

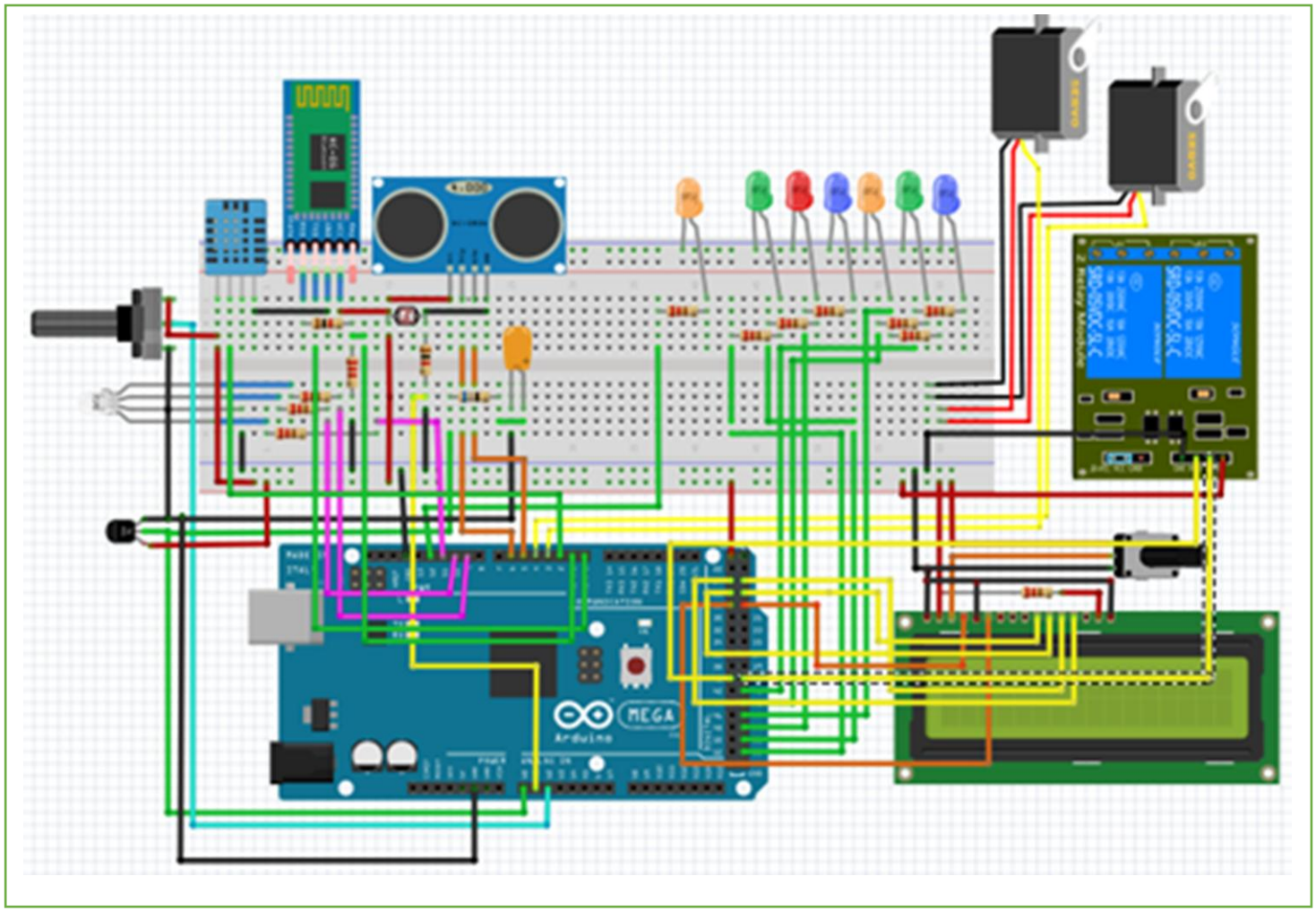

Figure4.3 : Schéma électrique du C-A-T réalisé

Le tableau suivante, récapitule les différents composants que nous avons utilisés avec leur rôle (capteur, actionneur ou transducteur) ainsi que les tâches de gestion qui leurs sont associées pour réaliser notre projet.

N°	Composant	Fonction	Type	Signal généré	Quantité	Tâche associée	Priorité
1	LM35	Mesurer la Température	Capteur	Analogique	1	TaskTempReadPin0	1
2	LDR	Mesurer la lumière	Capteur	Analogique	1	TaskTempReadPin1	1
3	Potentiomètre	Varier la tension	Capteur	Analogique	1	TaskAnalogRead012	1
4	HCSR04	Mesurer la distance	Capteur	Digital	1	TaskMDistance	1
5	DHT11	Mesurer la température et l'humidité	Capteur	Digital	1	TaskMTH	1
6	Servomoteur	Positionnement Angulaire	Actionneur	Digital	2	TaskCtrServo	1
7	Relais	Interrupteur	Actionneur	Digital	2	TaskCtrRelais	1
8	LED 1-6	Défilement de lumière	Transducteur	Digital	10	TaskChenillard	1
	LED 7	Indiquer l'état du relais 1		Digital		TaskCtrRelais	1
	LED 8	Indiquer l'état du relais 1		Digital			1
	LED 9	Réponse à la commande à partir de l'IHM		Digital		TaskCtrLed	1
	RGB LED	Varier le spectre de la lumière		Digital		TaskRGB	1
9	LCD 16x2	Afficher la température et l'intensité de la lumière.	Transducteur	Digital	1	TaskLcd	2

composants matériels et tâches associées pour réaliser le C-A-T

La figure 4.4 ci-dessous, une vue du C-A-T réalisé.

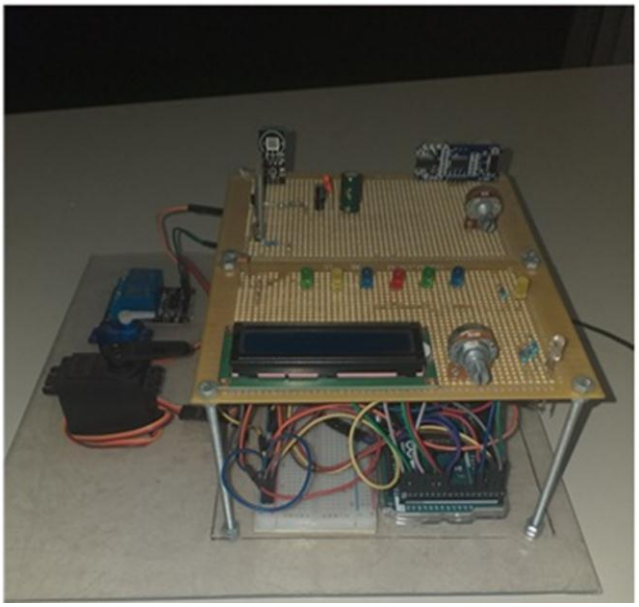

Figure 4.4 : Une vue de la maquette du C-A-T réalisé

Chapitre 5 : Réalisation logicielle

5.1. Introduction

Dans ce chapitre, freeRtos, la tâche au sens de freeRtos, le développement de l'application embarquée et celui de l'interface graphique vont être discutés.

5.2. Les systèmes d'exploitation temps réel (RTOS)

Dans les applications embarquées temps réel, un microcontrôleur ne peut exécuter qu'une seule tâche à un instant donné. Une partie du système d'exploitation, appelée planificateur ou ordonnanceur, est chargée de décider quand exécuter une tâche et laquelle. Il donne l'illusion d'une exécution simultanée en basculant rapidement entre les tâches (Figure 5.1).

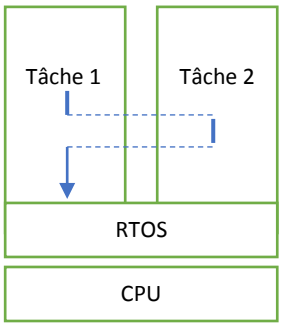

Figure5.1 : Exécution pseudo-parallèle

Le RTOS est un système d'exploitation destiné à servir des applications en temps réel via un microcontrôleur qui traite les données au fur et à mesure qu'elles arrivent, généralement sans délai. Les exigences de temps de traitement sont signalées en dixièmes de seconde ou par incréments plus courts.

De nos jours, de nombreux RTOS conçus avec open source sont disponibles au téléchargement et à l'utilisation gratuitement, y compris FreeRTOS. Le FreeRTOS, est une classe de RTOS conçus pour être suffisamment petits (faible empreinte) pour fonctionner sur un microcontrôleur pour des applications dans les domaines éducatifs et commerciaux. C'est un RTOS qui fournit une solution unique et indépendante pour de nombreux outils

d'architecture et de développement différents (https://freertos.org). Dans ce travail, nous avons sélectionné FreeRTOS pour une utilisation dans la carte microcontrôleur Arduino.

5.3. FreeRTOS pour Arduino

Dans cette section, nous décrivons brièvement comment effectuer plusieurs tâches à l'aide du microcontrôleur Arduino avec FreeRTOS. Cependant, pour l'utiliser pour Arduino, il existe deux bibliothèques Arduino : port of FreeRTOS par Bill Greiman(2015) pour la famille des microcontrôleurs Arduino AVR et la bibliothèque https://github.com/greiman/FreeRTOS-Arduino/tree/master/libraries/ pour la famille des microcontrôleurs arduino ARM. Ces bibliothèques sont FreeRTOS_AVR et FreeRTOS_ARM.

Dans une application embarquée, pour faire du multitâche, il suffit d'inclure FreeRTOS_AVR.h ou FreeRTOS_ARM.h, selon l'architecture du microcontrôlleur, ensuite exploiter les fonctions de l'API de FreeRTOS.

5.4. Les tâches dans freeRtos

Ce sont de simples fonctions qui généralement s'exécutent en boucle infinie (Figure 5.2) et qui suivent la structure générique suivante :

```
void vATaskFunction( void *pvParameters )
{
  while(1)
  {
    //code de la tâche
  }
}
```

Figure 5.2 : structure générique d'une tâche dans freeRtos

5.5. L'application embarquée réalisée

Pour écrire correctement des applications qui utilisent un RTOS, un développeur est confronté à trois difficultés : Déterminer correctement le nombre de tâches, attribuer à ces tâches des priorités, gérer leurs communications, leurs synchronisations et leurs concurrences.

Dans les paragraphes qui suivent nous allons développer chacun de ces points.

5.5.1. Le nombre de tâches de l'application embarquée du C.A.T réalisé

Pour déterminer le nombre correct de tâches que va comprendre l'application embarquée de notre C-A-T, nous avons procédé à sa décomposition.

Pour ce faire, nous avons adopté l'approche « Outside-in ».

Cette approche comprend principalement deux étapes : identifier les entrées et les sorties du C-A-T et les exprimer dans ce qu'on appelle dans la terminologie de cette approche, un diagramme de contexte, ensuite attribuer à chacune, éventuellement à plusieurs selon la complexité de leur gestion, une tâche pour obtenir la décomposition cherchée.

Un diagramme de contexte est un diagramme qui représente un système embarqué avec ses périphériques d'entrées et ses périphériques de sorties.

On donne à la figure 5.3 suivante, le diagramme de contexte du C-A-T que nous avons réalisé.

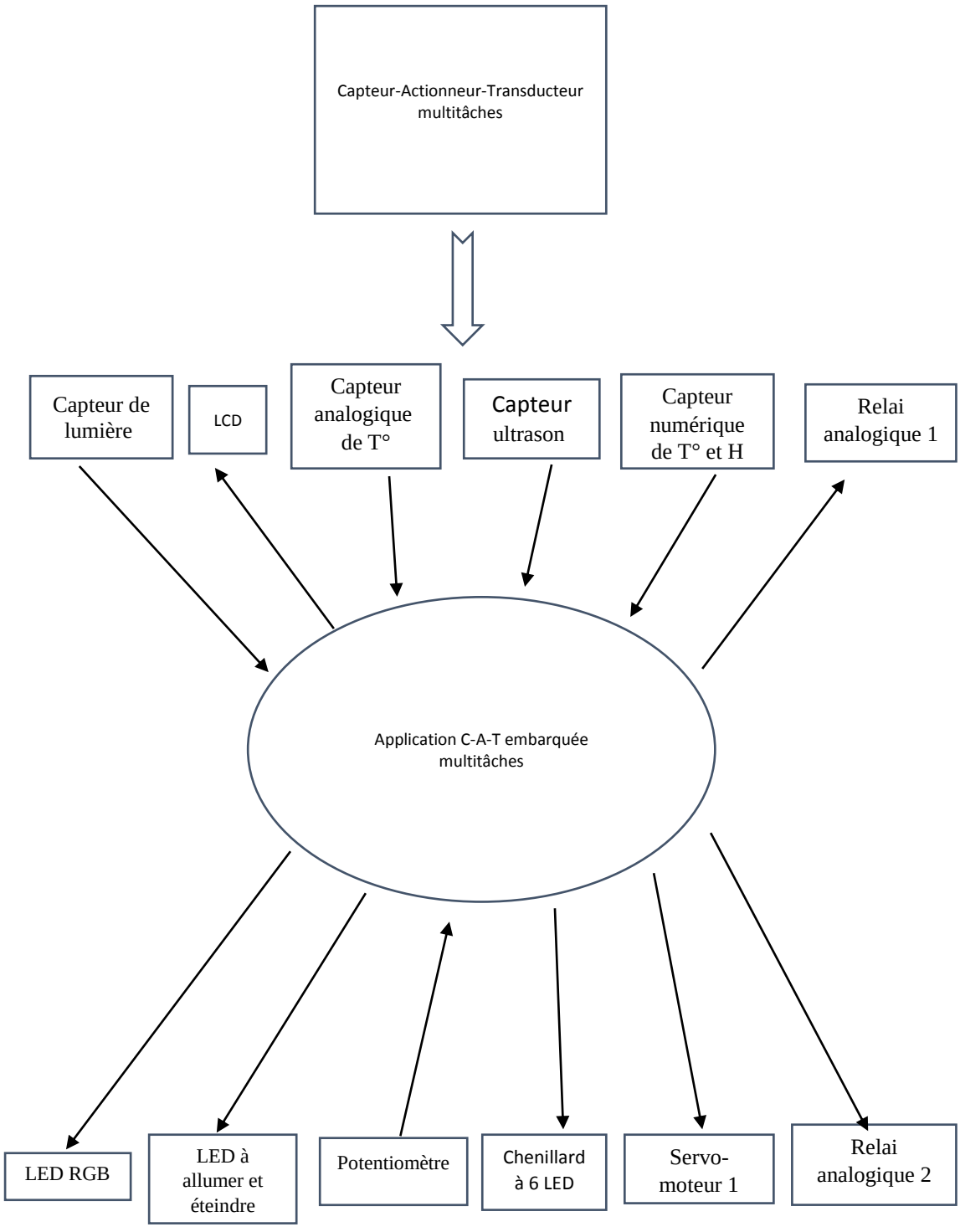

Figure 5.3 : Diagramme de contexte du C.A.T réalisé

Le cercle au centre du diagramme représente l'application embarquée à réaliser. Les cases rectangulaires représentent les périphériques d'entrée et de sortie. Les flèches représentent les flux des communications d'entrée et de sortie.

De même on donne à la figure 4 suivante, la décomposition en tâches, à laquelle nous avons abouti, de l'application que nous avons développée, déduite directement du diagramme de contexte :

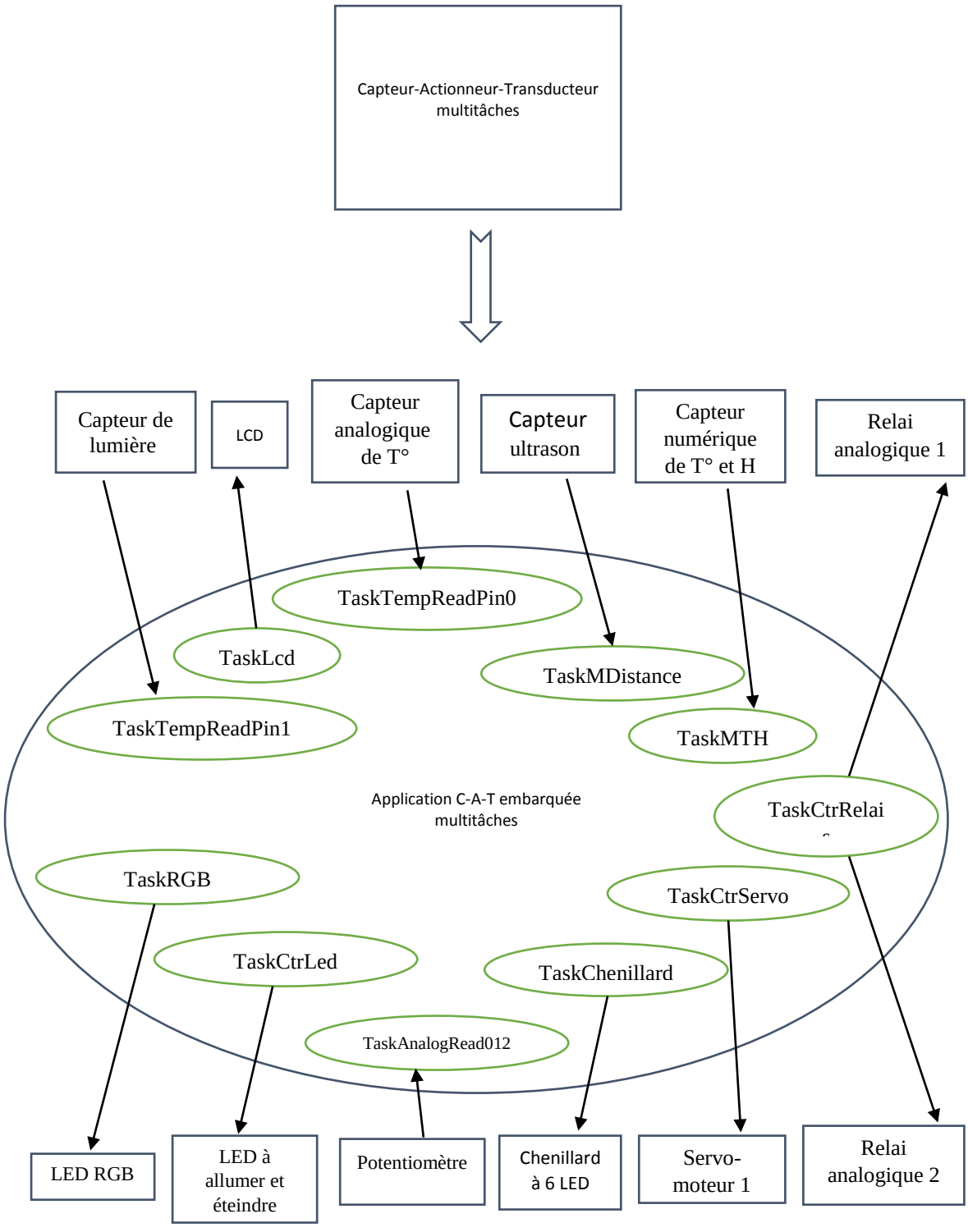

Figure 5.4 : Les tâches de l'application embarquée du C.A.T

Dans cette approche, les tâches sont représentées par des ellipses.

5.4.2. Les priorités des tâches

L'ordonnancement mis en œuvre dans freeRtos est basé sur le modèle du tourniquet (Round-Robin) avec gestion des priorités. Aussi, c'est ce modèle sur lequel nous nous sommes basés pour attribuer les priorités aux différentes tâches de l'application embarquée que nous avons réalisée.

Dans FreeRTOS Il n'y a aucun mécanisme automatique de gestion des priorités. La priorité d'une tâche ne pourra être changée qu'à la demande explicite du développeur.

5.4.3. La communication entre les tâches

FreeRTOS utilisant les files comme moyen de communication entre les tâches, c'est cet outil que nous avons utilisé pour faire communiquer les deux tâches qui envoient les données de mesure de la température et de la lumière à afficher, avec la tâche qui les reçoit et les affiche sur l'afficheur LCD (Figure 5.5).

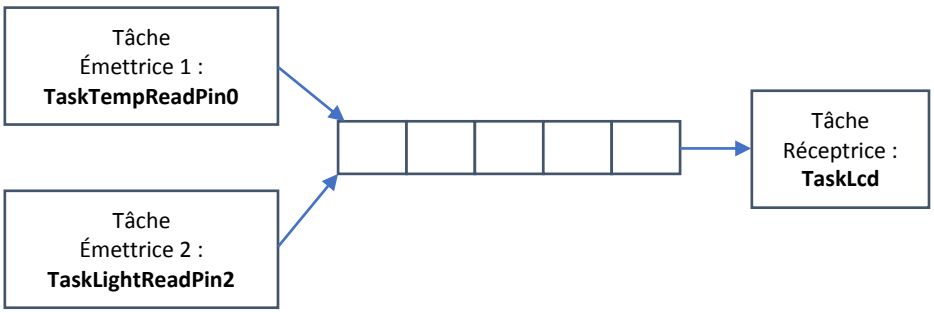

Figure 5.5 : communication par file entre les tâches de mesure de la température et de la lumière et la tâche de gestion de l'afficheur LCD

5.4.4. La concurrences des tâches

FreeRTOS synchronise les tâches en utilisant principalement deux mécanismes : les Sémaphores et les Mutex.

Les tâches de l'application que nous avons développée, partagent le lien série pour envoyer des données à afficher à la console virtuelle de l'EDI d' Arduino. Mutex étant plus facile à utiliser, c'est ce mécanisme que nous avons mis en oeuvre pour réaliser l'exclusion mutuelle entre nos tâches concurrentes.

On donne le bout de code en langage C montrant comment fonctionne l'exclusion mutuelle de la ressource COMxx entre les tâches qui la partagent.

```
1: while (COMxx_occupe); // si COMxx occupé,attendre jusqu'à libre
2: COMxx_occupe = 1; // utiliser le COMxx
3: envoyerOctets(COMxx, donnee);
4: COMxx_occupe = 0;// libérer le COMxx
```

Algorithme : principe de fonctionnement de l'exclusion mutuelle appliquée au lien série partagé par les tâches de l'application embarquée réalisée.

5.5. L'interface graphique réalisée

On donne à la figure 5.6 suivante, l'interface graphique que nous avons réalisée. Nous avons développé toutes les fonctions de cette interface avec Processing 3.0.

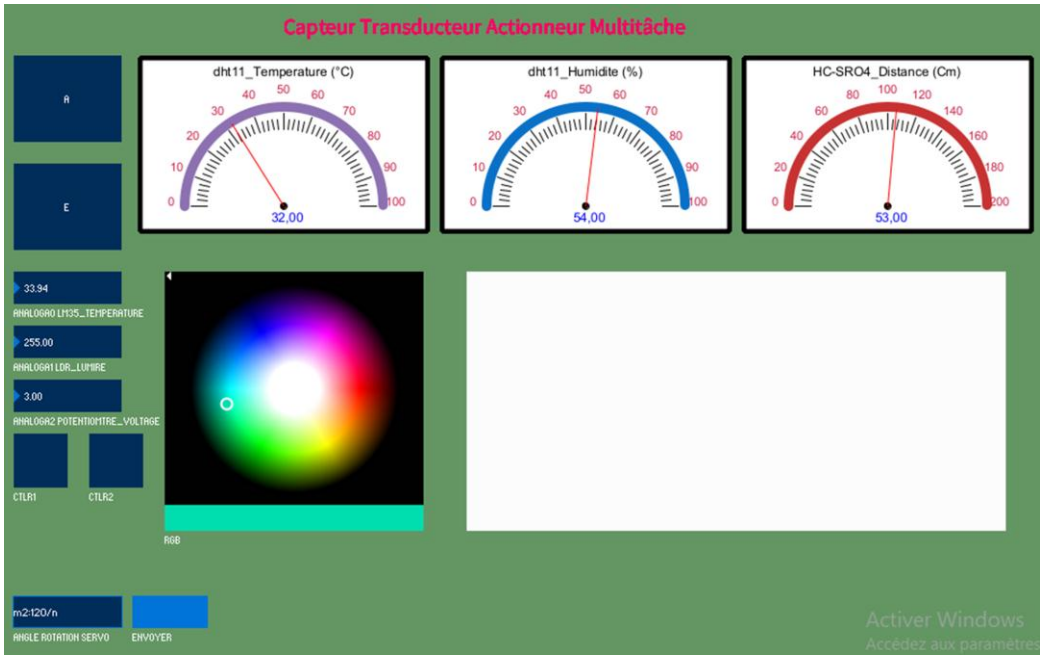

Figure 5.6 : L'interface graphique réalisée

La figure 5.7 ci-dessous, montre une vue de l'ensemble C-A-T et interface graphique en fonctionnement.

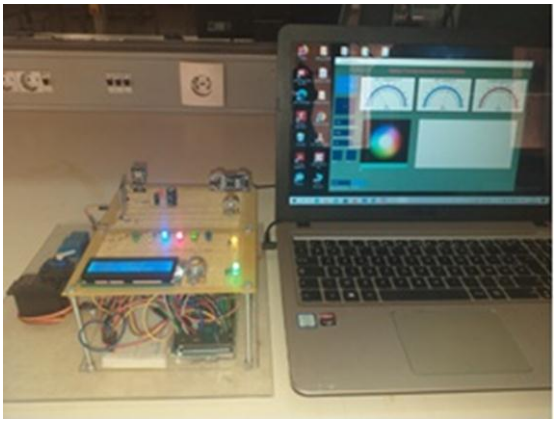

Figure 5.7 : vue de la maquette et de son interface, en fonctionnement.

Conclusion générale

Nous avons réalisé dans notre projet de fin d'études, un capteur-transducteur-actionneur multitâches à l'aide du système d'exploitation temps réel freeRtos qui permet de mesurer de par sa conception n'importe quelle grandeur physique et de commander n'importe quel système.

Sa composante logicielle est constituée de deux parties : l'application multitâches embarquée de gestion des activités internes et externes du C-T-A et l'IHM de type GUI qui permet d'afficher les valeurs de mesure et d'envoyer des commandes à ce dernier à l'aide de la liaison série ou via bluetooth.

Nous avons développé l'application embarquée avec le langage arduino et l'IHM avec le langage processing.

La mise en œuvre de freeRtos pour la réalisation de l'application embarquée, nous a énormément facilité le travail, sans lequel, le développement de cette dernière aurait été très difficile.

Ce travail nous a été très bénéfique car il nous a permis de consolider nos connaissances, de beaucoup apprendre et de se familiariser davantage avec des techniques de développement qui nous ont permis d'améliorer nos compétences et nos acquis dans la programmation multitâche et des interfaces graphiques.

Enfin ce projet peut constituer une très bonne assise et un très bon prélude pour des travaux se faisant dans ce domaine car nous y avons réunis toutes les bases nécessaires.

Bibliographie

1. Feijs, L. “Multi-tasking and Arduino: Why and How? Design and semantics of form and movement”. Retrieved May 1, 2018, from https://ccrma.stanford.edu/~gurevich/Feijs_ArduinoMultitasking.pdf, 2013.

2. Buonocunto, P., Biondi, A., & Lorefice, P. “Real-Time Multitasking in Arduino”. Proceedings of the 9th IEEE International Symposium on Industrial Embedded Systems (SIES), https://doi.org/10.1109/SIES.2014.7087331. Pisa, Italy, 2014.

3. Barry, R. “Multitasking on AVR”. Retrieved May 3, 2018, from d1.amobbs.com/bbs_upload782111/files_19/ ourdev_488657.pdf , 2004.

4. Pantano, N., & Timmerman, M.P. “Real-Time Operating System on Arduino. OSSEC mini project”. Retrieved May 5, 2018, from https://id.scribd.com/document/222365331/Nicolas-Pantano-Report, 2011.

5. Principes fondamentaux du noyau FreeRTOS https://docs.aws.amazon.com/fr_fr/freertos/latest/userguide/dev-guide-freertos-kernel.html