# PROJET 2 – SNIOT : Maison connectée avec ZephyrOs et ESP32

## Introduction

L’objectif est de découvrir le système d’exploitation embarqué temps réel Zephyr. Vous allez découvrir l’environnement de développement en réalisant un projet de maison connectée. Vous allez apprendre à :

* Réaliser une description matérielle du système dans le Device Tree
* Utiliser les fonctionnalités d’un système temps réel (Programmation Concurrente, Thread, Sémaphore, Mutex …)
* Utiliser le système de driver
* Communiquer des données sans fil
* Créer une API en python

Les rendus attendus sont :

* Le repo GitHub avec l’historique du travail
* Le rapport de conception rempli
* Une démonstration du programme

Vous êtes autorisé à utiliser toutes les ressources à votre disposition. Le plagiat est bien sûr interdit vous devez trouver votre propre conception et faire vos propres choix. Vous êtes néanmoins autorisé à vous aider mutuellement ! N’hésitez pas à solliciter de l’aide si besoin.

La partie rapport de projet est surtout là pour vous guider et vous aidez à vous poser les bonnes questions.

Le travail est à réaliser en monôme dans la limite du matériel disponible.

Un répertoire github contient les fichiers importants pour la réalisation du projet : <https://github.com/Vivoulia/sniot-zephyr.git>

Durée du projet : 3 séances

## Cahier des charges

Nous souhaitons réaliser un prototype de maison connecté en utilisant le kit SmartHome de Keyestudio : <https://wiki.keyestudio.com/KS5009_Keyestudio_Smart_Home>.

Vous trouverez la documentation du kit sur le dropbox suivant : <https://fs.keyestudio.com/KS5009>

**Attention : Vous devez travailler sur un repo GitHub tout au long de votre projet !**

|  |  |  |
| --- | --- | --- |
| **Fonction** | **Nom** | **Description** |
| F1 | Acquisition de la température et de l’humidité | Acquisition de la température et de l’humidité toutes les 10s avec le capteur XHT11 |
| F2 | Acquisition de la quantité de « vapeur » | Lecture analogique de la quantité de vapeur avec le Steam Sensor toutes les 10s |
| F3 | Allumage LED démarrage | Allumage de la LED Orange au démarrage du système |
| F4 | Système d’alarme d’intrusion | L’appui sur le bouton gauche place la maison en mode « Alarme ». Si un intru est détecté avec le capteur PIR un signal sonore à une fréquence de 1000Hz sonne tant que l’intru n’est pas parti. |
| F5 | Affichage LCD | Un message d’accueil s’affiche au lancement. Un message s’affiche en mode Alarme. Un message s’affiche quand un intru est détecté. |
| F6 | Envoi des données par Wifi | La température et l’humidité sont envoyées par Wifi à un serveur. |
| F7 (bonus) | Transmission NFC | Utiliser la puce NFC pour rediriger l’utilisateur vers un site internet par exemple |

## Rapport de projet

### Préliminaires

1. Décrire avec vos mots à quoi sert le mot-clé volatile et quand l’utiliser en programmation embarqué ? Le mot clé volatile est utilisé pour signaler au compilateur que la valeur d’une variable peut changer de manière inattendue, par exemple dans une interruption.
2. Quel outil pouvez-vous utiliser pour protéger une variable partagée entre deux threads ?  
   Nous pouvons utiliser le mutex.

### Allumage de la LED de démarrage

La LED orange est sur le GPIO 12. Ajouter la description de la LED dans la DeviceTree grâce à un fichier **d’overlay**. Récupérer le GPIO avec la macro prévue à cet effet dans le code. Initialiser le GPIO à l’état haut au démarrage.

### Affichage LCD

Votre collègue Bruno a développé un code pour le driver de l’afficheur I²C LCD 1602. Cependant il y avait un afterwork hier soir et vous le soupçonnez d'avoir bâclé le travail. Le driver contient probablement des erreurs faites attention !

Le driver est disponible sur le répertoire github du projet. Intégrer le driver à votre code en corrigeant les potentielles erreurs.

### Acquisition de la température et de l’humidité

Le capteur de température et d’humidité utilisé est le DTH11. Il fonctionne avec un bus de donnée sur une seule ligne GPIO. Un protocole bien spécifique est défini dans la datahseet du capteur. Nous n’allons heureusement pas implémenter ce protocole car Il existe déjà un driver Zephyr.

Ajouter dans l’overlay de votre projet un GPIO pour le capteur avec la syntaxe Figure 1 puis récupérer le *device* dans votre code.

En utilisant l’interface sensor de Zephyr et les fonctions sensor\_sample\_fetch, sensor\_channel\_get et sensor\_value\_to\_double ainsi que la documentation <https://docs.zephyrproject.org/latest/hardware/peripherals/sensor.html>, récupérer et afficher une fois toutes les 10 secondes les valeurs de température et d’humidité.



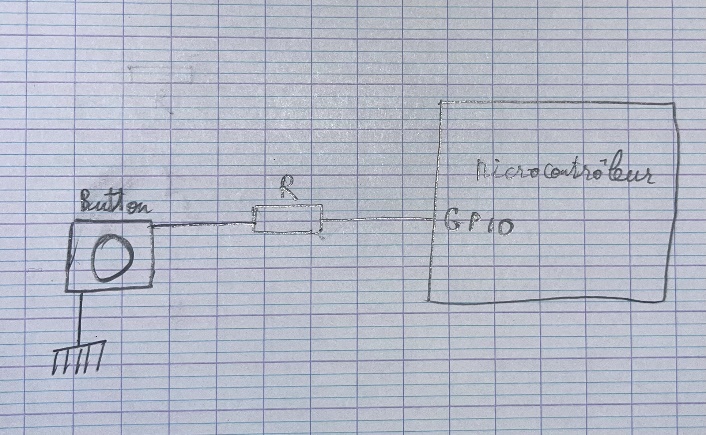
Figure : Syntaxe pour le pin GPIO dans l'overlay de votre projet

### Acquisition de l’humidité analogique

En utilisant l’exemple [exemple adc Zephyr](https://docs.zephyrproject.org/latest/samples/drivers/adc/README.html) implémenter la lecture analogique de l’humidité avec le « steam sensor ».

### Détection de l’appui bouton

1. Dessiner le schéma électronique classique permettant de relier un bouton à un microcontrôleur.



1. A quel phénomène faut-il être vigilant lorsqu’on cherche à détecter l’appui bouton dans le programme ? Lorsque le bouton est appuyé, il peut rebondir, provoquant des lectures instables et multiples détections d’appui. Ce phénomène s’appelle le rebond. Il peut être traité en ajoutant un délai dans le code.
2. Quand vous utilisez une variable dans une interruption quel mot-clé doit être utilisé pour déclarer cette variable ? Quand une variable est utilisée dans une interruption, elle doit être déclarée avec le mot-clé volatile. Cela empêche le compilateur d’optimiser la variable, assurant qu’elle est toujours lue dans sa valeur actuelle.

Deux boutons sont présents sur le système sur les pins 16 et 27. Ajouter les dans l’overlay de votre projet. N’oublier pas d’ajouter un alias. Récupérer ensuite le pin gpio dans le programme. Initialiser le GPIO et configurer une interruption. Ecrire un code permettant de détecter les appuis bouton. Vérifier le bon fonctionnement de ce code dans le moniteur.

Nous allons maintenant rajouter une fonctionnalité d’affichage LCD reliée à l’appui bouton. Un code propre et bien conçu ne doit pas contenir de longues procédures dans les interruptions. La fonctionnalité d’affichage sera donc placée dans un processus autre que la fonction d’interruption.

### Buzzer

Le buzzer est relié au pin 25. Ajouter le GPIO dans l’overlay de votre projet et initialisé le dans votre code. Le buzzer ne contient pas d’oscillateur intégré. Vous devez donc programmer une PWM permettant de générer le son à la fréquence demandée. Créer un thread permettant de faire osciller le buzzer. Afin d’éviter d’avoir mal à la tête à la fin de la journée ajouter une configuration permettant de désactiver le buzzer. Relier l’activation du buzzer avec le bouton. Tester en affichant un message sur l’écran LCD.

### Capteur de présence et système d’alarme

Maintenant que le buzzer fonctionne il reste à ajouter le capteur de présence qui est sur le pin 14. Ajouter le GPIO correspondant au projet. Pour utiliser le capteur, il suffit de lire la valeur renvoyée par celui-ci :

* 0 : Pas de présence
* 1 : Présence détectée

Vérifier le bon fonctionnement du capteur puis réaliser le système d’alarme complet. Merci de ne pas utiliser le buzzer pendant la phase de test mais seulement des print ou l’écran LCD.

Réaliser l’ensemble du cahier des charges à l’exception de la partie Wifi.