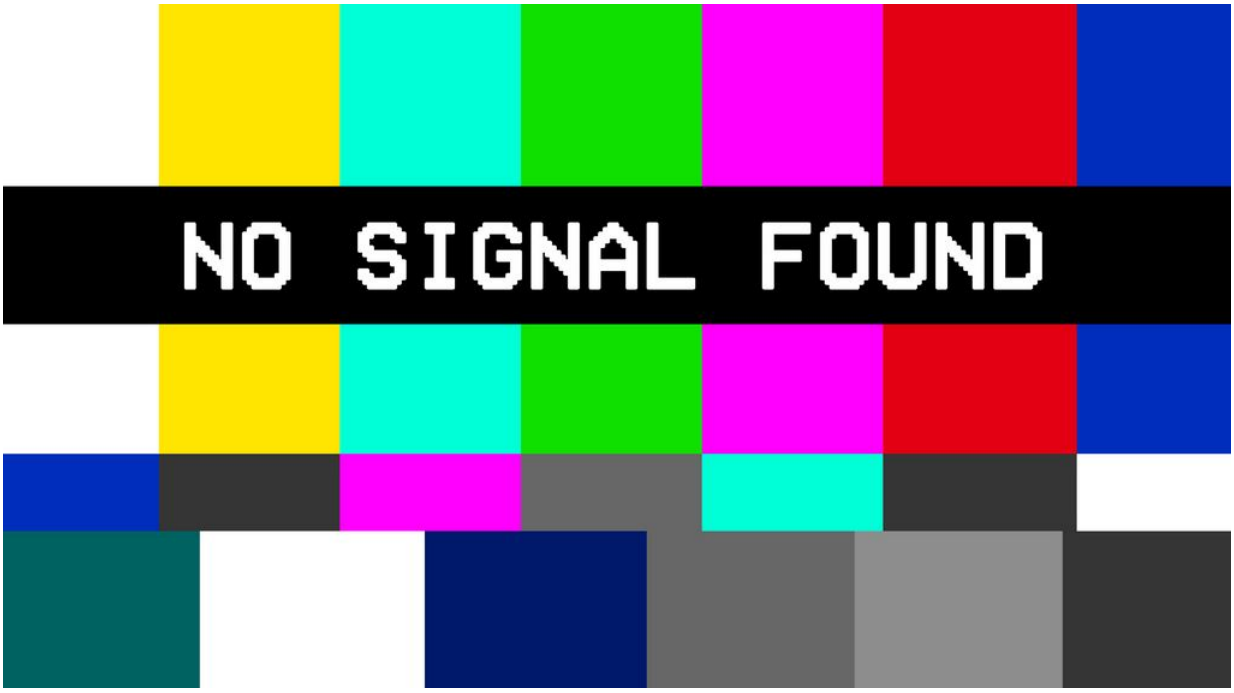
Haute école d'ingénierie et d'architecture Fribourg
Hochschule für Technik und Architektur Freiburg

# Systèm embarqués

*Auteurs :*
*Marc* Roten
*Sven* Rouvinez

*Professeur :*
*Daniel* Gachet




25 octobre 2018

# Table des matières

# 1 Heure de travail

6 heures

# 2 Synthèse

Ce TP a pour but de nous familiariser avec l'interaction des différentes entrées/ sorties de la Beaglebone.
Il faut pouvoir afficher un nombre sur les deux 7 segments et incrémenter/ décrémenter ce nombre en utilisant la "wheel" pour le mode 1 qui se choisit en cliquant sur le bouton 1 et pour le mode 2 avec le bouton 2, il faut aussi pouvoir "déplacer un segment" entre les différents afficheur et pour finir il faut pouvoir réinitialiser les 2 modes avec le bouton 3

**Sven**

- Non acquis : pour ma part j'ai relativement bien compris ce qu'il fallait faire et comment
- Acquis mais à exercer : la navigation entre les 2 digits
- Parfaitement acquis : `struct` en C ainsi que l'utilisation des headers et des commandes préprocesseur

**Marc**

- Non acquis : Rien n'es non acquis pour ma part,ce tp m'a aidé pour ma compréhension générale de la programmation en C. Toute la difficulté de ce TP reposait dans la communication avec le BeagleBone. Le reste c'est juste de la logique.
- Acquis mais à exercer : Pour ma Part la part, le C est nouveau, je me suis habitué à la syntaxe mais celà reste à exercer. Les parties qui restent à exercer en plus de celà est le nom des variables à utiliser pour communiquer avec le BeagleBone.
- Parfaitement acquis : Par ce TP, je comprend maintenant bien la programmation modulaire, le fait d'avoir un fichier décomposé en plusieurs fichiers.

# 3 Pourrait-on se passer des fichiers d'entête (header files) en C ?

Oui il est possible de se passer des fichiers d'entête, il suffirait de mettre les signatures dans les fichiers source tout en haut

# 4 #pragma once

Il permet d'éviter un import multiple de header files en incluant une fois uniquement les fichiers dans la compilation et peut être accompagné des commandes preprocessor `#ifndef symbol #define`

# 5 Que faut-il placer dans un fichier d'entête ?

Il faut placer la signature des méthodes avec les paramètres et le type de retour pour les méthodes qui sont utilisées dans le fichier source (.c)

# 6 Quelle est l'utilité des mots-clef extern et static ?

Le mot-clé `extern` permet de déclarer une variable qui soit disponible à l'extérieur d'un fichier source et donc elle sera accessible pour l'ensemble du code `static` C'est une variable accessible uniquement dans le fichier source qui la déclare mais une fois initialisée elle gardera la même case mémoire durant toute l'exécution du code et donc elle ne sera pas remise à sa valeur initiale lors de l'appel de la fonction
Concernant une fonction, elle sera accessible uniquement dans le fichier source

# 7 Comment faut-il procéder pour définir une constante en C ?

- `const type nom_variable=valeur`
- `#define NOM valeur`

# 8 Quelle(s) différence(s) existe-t-il entre les instructions

1. `#define MAX 10`
2. `const int MAX=10`

La première est une commande preprocesseur et donc partout où l'on va utiliser MAX il sera remplacé par 10, mais ne possède pas de type, donc peu importe que l'on utilise MAX pour un int, un double un long, il sera remplacé dans tous les cas par 10.

Et l'autre est une variable qui est constante donc elle a un type.

# 9 Comment peut-on définir une énumération en C ? Quelle est son utilité ?

`enum colors {RED, YELLOW, BLUE}` et permet d'initialiser une séquence de constantes qui pourra être utilisé plus tard. Dans notre travail pratique on réalise un enum sur les différents états avec notre encodeur, et on passe via nos méthodes au travers de notre énum pour incrémenter ou décrémenter notre compteur.

On peut aussi rajouter que :

- enum est un bon moyen d'avoir un code parlant sans avoir à recourir à des commentaires longs et fastidieux.
- Economie non négligeable d'espace mémoire, très important en système embarqué.
- Garantie d'avoir des sorties stables si on alterne parmis des états connus d'une FSM (Finite State Machine)

# 10 Quelle(s) différence(s) existe-t-il entre une structure en C struct S{} et une classe en Java class C{} ?

`struct S{}` permet de définir une structure par exemple un tableau avec plusieurs champ et chaque champ a un nom, ou on peut représenter un
`class C{}` permet de déclarer une class, il n'existe pas de class en C, l'équivalent est `struct` et il n'y a pas de différence entre une `struct` et une class excepté la notion d'objet en Java qu'il n'y a pas en C

# 11 Comment faut-il procéder pour définir un tableau en C ? Peut-on lui donner des valeurs initiales lors de sa définition ?

`int array_declaration[10];` et avec l'affectation `int array_declaration[]={10,9,8,7};`

# 12 Comment faut-il procéder pour obtenir le nombre d'éléments contenus dans un tableau ?

`sizeof(array)/sizeof(array[0])`
`sizeof` retourne la taille du type que l'on divise par le taille du type contenu

# 13 Conclusion

Ce travail était particulièrement difficile au début mais suite à une découpe en sous-problèmes et avec l'aide du professeur, nous avons pu mener à bien le projet et rendre un projet fonctionnel. La programmation modulaire est donc inévitable pour rendre une tâche complexe en plusieurs petits problèmes plus facile à aborder.