RAPPORT PROJET MULTI-TÂCHE-TEMPS REEL : Le développement d'un système temps réel et multi-tâches Gestion automatique

MEMBRE DU GROUPE :

- ILOKI Clarsi Trésor
- ESSAEGH Mohamed Amine.

Année-Académique : 2023-2024

1. Analyse des fonctionnalités et contraintes et étapes

❖ Analyse des fonctionnalités

- **Interaction avec l'environnement :**

  Le capteur de mouvement doit déclencher une réponse immédiate si une personne est détectée, TV1 doit s'allumer instantanément.

  Le capteur de lumière détermine en temps réel si c'est le jour ou la nuit.

- **Contrainte Temporelle**

  Chaque téléviseur doit respecter une contrainte temporelle stricte de 2heures d'allumage continu, ce qui impose une gestion précise du temps.

- **Fiabilité**

  Le système doit gérer les priorités par exemple, donner la priorité à TV1 lors de la détection d' une personne. Elle doit être fiable sinon il peut avoir une catastrophe.

- **Prédictibilité**

  Les délais d'exécution des taches extinction des téléviseurs sont cohérents et prévisibles, elle doit respecter les 2h d'allumage continu ou la détection d'une personne.

- **Déterminisme**

  Le système produit des sorties prévisibles en réponse à des entrées spécifiques comme détection de mouvement et d'état jour/nuit.

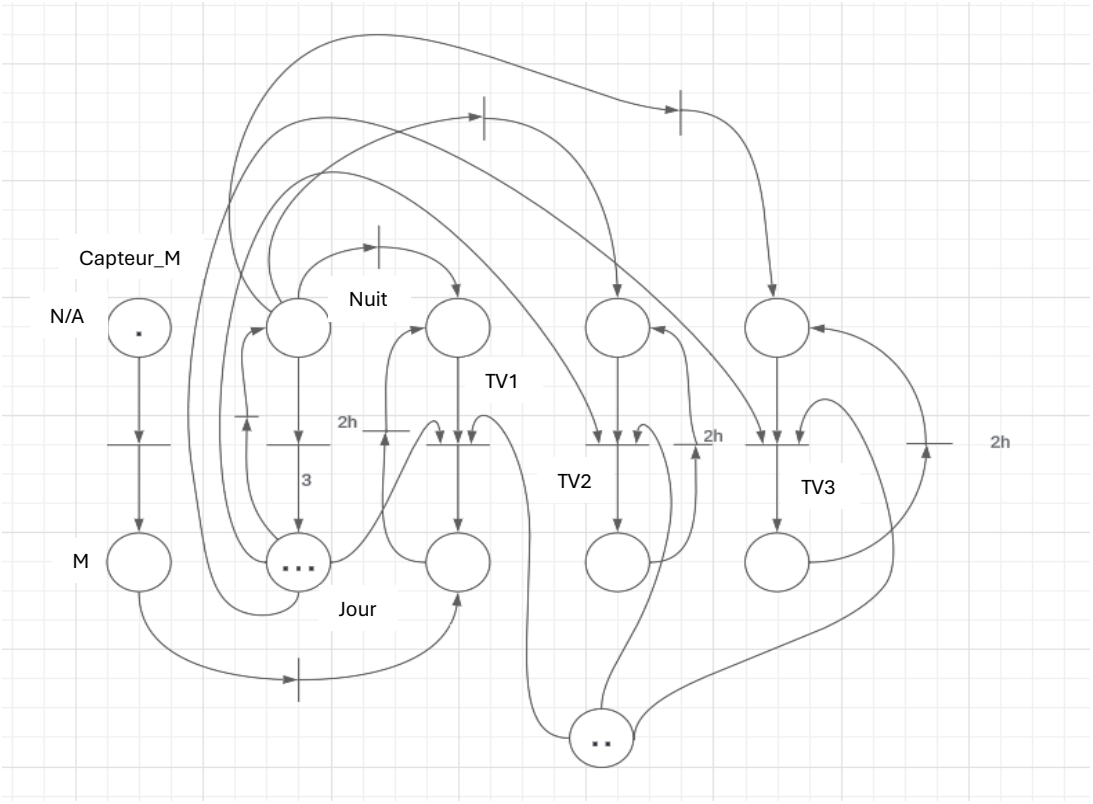
❖ Contraintes

- Durée d'allumage continus des téléviseurs limitée à 2 heures,
- Nombre de téléviseurs allumés simultanément limité à 2,
- Aucun téléviseur peut s'allumer le soir sauf le téléviseur 1 en cas d'urgence

❖ Etapes

- Détection de mouvement
- Vérification du jour/soir
- Gestion des téléviseurs

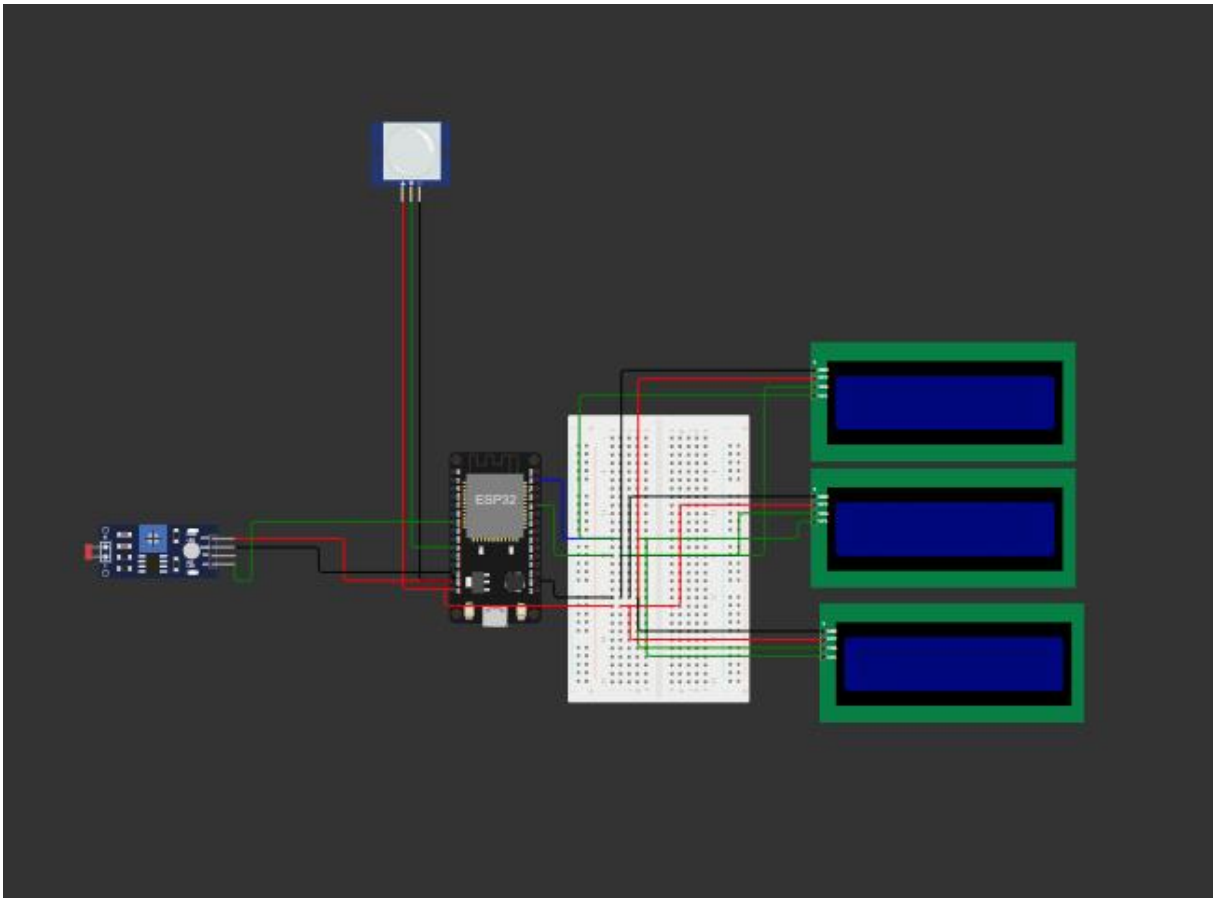
2. Réseau de pétri



3. Liste des taches et les priorités

Tâches	Priorités
Détection de mouvement	
Détection de jour/nuit	
Allumage de la TV1	
Allumage de la TV2	
Allumage de la TV3	
Eteindre la TV1	
Eteindre la TV2	
Eteindre la TV3	

4. Circuit du système



5. Programmation du système sur ESP32

```
#include <LiquidCrystal_I2C.h> // Inclut la bibliothèque LiquidCrystal_I2C

// Seuil de luminosité pour autoriser l'allumage des télévisions
#define SEUIL_LUMINOSITE 500 // Définit la valeur seuil de luminosité pour
allumer les télévisions

// Déclaration des pins pour les capteurs et actionneurs
#define LUMINOSITY_SENSOR_PIN 33 // Définit le pin pour le capteur de
luminosité
#define MOTION_SENSOR_PIN 27 // Définit le pin pour le capteur de mouvement

// I2C LCD display addresses
const int LCD1_ADDRESS = 0x27; // Adresse I2C de l'écran LCD 1
const int LCD2_ADDRESS = 0x28; // Adresse I2C de l'écran LCD 2
const int LCD3_ADDRESS = 0x29; // Adresse I2C de l'écran LCD 3

// Déclaration des timers
TimerHandle_t screenOffTimer1; // Timer pour éteindre l'écran 1
TimerHandle_t screenOffTimer2;
```

```
TimerHandle_t screenOffTimer3;
TimerHandle_t screenOffTimer4; // Timer pour éteindre l'écran 1 en cas
d'urgence

// Déclaration des objets LiquidCrystal_I2C pour chaque écran LCD
LiquidCrystal_I2C lcd1(LCD1_ADDRESS, 16, 2); // LCD 1 avec adresse, colonnes
et lignes spécifiées
LiquidCrystal_I2C lcd2(LCD2_ADDRESS, 16, 2); // LCD 2 avec adresse, colonnes
et lignes spécifiées
LiquidCrystal_I2C lcd3(LCD3_ADDRESS, 16, 2); // LCD 3 avec adresse, colonnes
et lignes spécifiées

// Déclaration de la file de messages pour les événements de luminosité
QueueHandle_t luminosityEventQueue; // File de messages pour les événements de
luminosité
QueueHandle_t motionEventQueue; // File de messages pour les événements de
mouvement

// Déclaration de la sémaphore multiple pour contrôler l'accès aux écrans
SemaphoreHandle_t tvSemaphore; // Sémaphore pour contrôler l'accès aux écrans


// Tâche de gestion de la luminosité
void taskLuminosity(void *pvParameters) {
  int luminosity; // Déclare une variable pour la luminosité
  while (1) { // Boucle infinie
    luminosity = analogRead(LUMINOSITY_SENSOR_PIN); // Lit la valeur de
luminosité à partir du capteur
    const float GAMMA = 0.7; // Définit la valeur de gamma pour la conversion
de la luminosité
    const float RL10 = 50; // Définit la valeur de RL10 pour la conversion de
la luminosité
    float voltage = luminosity / 1024.0 * 5.0; // Calcule la tension à partir
de la valeur de luminosité
    float resistance = 2000 * voltage / (1 - voltage / 5); // Calcule la
résistance à partir de la tension
    float luminosityValue = pow(RL10 * 1e3 * pow(10, GAMMA) / resistance, (1 /
GAMMA)); // Calcule la valeur de luminosité corrigée

    Serial.print("Luminosity: "); // Affiche le message "Luminosity: " sur le
moniteur série
    Serial.println(luminosityValue); // Affiche la valeur de luminosité
corrigée sur le moniteur série

    if (luminosityValue >= SEUIL_LUMINOSITE) { // Vérifie si la luminosité
dépasse le seuil

      Serial.println("Il fait jour ! "); // Affiche "Il fait jour ! " sur le
moniteur série
```

```
      xQueueSend(luminosityEventQueue, &luminosity, portMAX_DELAY); // Envoie
un message à la file d'événements de luminosité
    } else { // Si la luminosité est inférieure au seuil
      Serial.println("Il fait Nuit ! "); // Affiche "Il fait Nuit ! " sur le
moniteur série
      lcd1.clear(); // Efface l'écran LCD 1
      lcd1.noBacklight(); // Éteint le rétroéclairage de l'écran LCD 1
      xSemaphoreGive(tvSemaphore);
      lcd2.clear();
       lcd2.noBacklight(); // Éteint le rétroéclairage de l'écran LCD 2
      xSemaphoreGive(tvSemaphore); // Libère la sémaphore
      lcd3.clear(); // Efface l'écran LCD 3
      lcd3.noBacklight(); // Éteint le rétroéclairage de l'écran LCD 3
      xSemaphoreGive(tvSemaphore); // Libère la sémaphore
    }

    vTaskDelay(pdMS_TO_TICKS(1000)); // Attends 1 seconde
  }
}

// Fonction de rappel pour éteindre l'écran
void turnOffScreen1(TimerHandle_t xTimer) {
  lcd1.clear(); // Efface l'écran LCD 1
  lcd1.noBacklight(); // Éteint le rétroéclairage de l'écran LCD 1
  xSemaphoreGive(tvSemaphore); // Libère la sémaphore pour permettre l'accès à
d'autres tâches
}

void turnOffScreen2(TimerHandle_t xTimer) {
  lcd2.clear(); // Efface l'écran LCD 2
  lcd2.noBacklight(); // Éteint le rétroéclairage de l'écran LCD 2
  xSemaphoreGive(tvSemaphore); // Libère la sémaphore pour permettre l'accès à
d'autres tâches
}

void turnOffScreen3(TimerHandle_t xTimer) {
  lcd3.clear(); // Efface l'écran LCD 3
  lcd3.noBacklight(); // Éteint le rétroéclairage de l'écran LCD 3
  xSemaphoreGive(tvSemaphore); // Libère la sémaphore pour permettre l'accès à
d'autres tâches
}

// Tâche de gestion de l'écran en cas d'allumage de la télévision
void taskTelevision1(void *pvParameters) {
  int luminosity; // Déclare une variable pour la luminosité
  while (1) { // Boucle infinie
    if (xQueueReceive(luminosityEventQueue, &luminosity, portMAX_DELAY) ==
pdPASS) { // Vérifie s'il y a un événement de luminosité dans la file
```

```
      if (xSemaphoreTake(tvSemaphore, portMAX_DELAY) == pdTRUE) { // Essaye de
prendre la sémaphore pour contrôler l'accès à l'écran
        lcd1.init(); // Initialise l'écran LCD 1
        lcd1.backlight(); // Allume le rétroéclairage de l'écran LCD 1
        lcd1.setCursor(0, 0); // Définit la position du curseur de l'écran LCD
1
        lcd1.print("TV1"); // Affiche "TV1" sur l'écran LCD 1
        xTimerStart(screenOffTimer1, 0); // Démarre le timer pour éteindre
l'écran LCD 1 après un certain délai
        vTaskDelay(pdMS_TO_TICKS(2000)); // Attend 2 secondes
      }
    }
  }
}

void taskTelevision2(void *pvParameters) {
  int luminosity; // Déclare une variable pour la luminosité
  while (1) { // Boucle infinie
    if (xQueueReceive(luminosityEventQueue, &luminosity, portMAX_DELAY) ==
pdPASS) { // Vérifie s'il y a un événement de luminosité dans la file
      if (xSemaphoreTake(tvSemaphore, portMAX_DELAY) == pdTRUE) { // Essaye de
prendre la sémaphore pour contrôler l'accès à l'écran
        lcd2.init(); // Initialise l'écran LCD 1
        lcd2.backlight(); // Allume le rétroéclairage de l'écran LCD 2
        lcd2.setCursor(0, 0); // Définit la position du curseur de l'écran LCD
2
        lcd2.print("TV2"); // Affiche "TV2" sur l'écran LCD 2
        xTimerStart(screenOffTimer2, 0); // Démarre le timer pour éteindre
l'écran LCD 2 après un certain délai
        vTaskDelay(pdMS_TO_TICKS(2000)); // Attend 2 secondes
      }
    }
  }
}

void taskTelevision3(void *pvParameters) {
  int luminosity; // Déclare une variable pour la luminosité
  while (1) { // Boucle infinie
    if (xQueueReceive(luminosityEventQueue, &luminosity, portMAX_DELAY) ==
pdPASS) { // Vérifie s'il y a un événement de luminosité dans la file
      if (xSemaphoreTake(tvSemaphore, portMAX_DELAY) == pdTRUE) { // Essaye de
prendre la sémaphore pour contrôler l'accès à l'écran
        lcd3.init(); // Initialise l'écran LCD 3
        lcd3.backlight(); // Allume le rétroéclairage de l'écran LCD 3
        lcd3.setCursor(0, 0); // Définit la position du curseur de l'écran LCD
3
        lcd3.print("TV3"); // Affiche "TV3" sur l'écran LCD 3
        xTimerStart(screenOffTimer3, 0); // Démarre le timer pour éteindre
l'écran LCD 3 après un certain délai
```

```
        vTaskDelay(pdMS_TO_TICKS(2000)); // Attend 2 secondes
      }
    }
  }
}
// Routine de service d'interruption pour le capteur de mouvement
// Cette fonction doit être très courte car elle va interrompre le système.
// L'idée est que cette ISR (Routine de Service d'Interruption) envoie
simplement un
//signal à d'autres fonctions prioritaires (tâches normales mais avec priorité
haute)
// pour éviter de prendre trop de temps.
// Si un code long est placé dans cette fonction ISR, le système risque d'être
// interrompu pendant une longue période.
// Nous envoyons un signal rapidement à travers la file de messages
// depuis la fonction ISR, puis nous sortons de celle-ci.
// Les tâches qui ont besoin de ce signal doivent être programmées
//avec une priorité maximale.


void IRAM_ATTR motionSensorISR() {
  int danger = 1; // Déclare une variable pour indiquer le danger
  BaseType_t xHigherPriorityTaskWoken = pdFALSE; // Déclare une variable pour
indiquer si une tâche à priorité supérieure a été réveillée
  xQueueSendFromISR(motionEventQueue, &danger, &xHigherPriorityTaskWoken); //
Envoie un message à la file d'événements de mouvement depuis l'interruption
  portYIELD_FROM_ISR(xHigherPriorityTaskWoken); // Indique que la tâche à
priorité supérieure doit être réveillée
}

// Nouvelle tâche pour gérer les événements de mouvement lorsque l'écran 1 est
éteint
void taskTelevision1_case1(void *pvParameters) {
  int danger; // Déclare une variable pour indiquer le danger
  while (1) { // Boucle infinie
    if (xQueueReceive(motionEventQueue, &danger, portMAX_DELAY) == pdPASS) {
// Vérifie s'il y a un événement de mouvement dans la file
        Serial.println("Motion Detected! Ecran 1 était dans l'état éteint");
// Affiche "Motion Detected! Ecran 1 était dans l'état éteint" sur le moniteur
série
        lcd2.clear(); // Efface l'écran LCD 2
        lcd2.noBacklight();

        lcd1.init(); // Initialise l'écran LCD 1
        lcd1.backlight(); // Allume le rétroéclairage de l'écran LCD 1
        lcd1.setCursor(0, 0); // Définit la position du curseur de l'écran LCD
1
        lcd1.print("Message d'urgence!"); // Affiche "Message d'urgence!" sur
l'écran LCD 1
```

```
        xTimerStart(screenOffTimer4, 0); // Démarre le timer pour éteindre
l'écran LCD 1 après un certain délai
      }
    }
}


// Nouvelle tâche pour gérer les événements de mouvement lorsque l'écran 1 est
allumé
void taskTelevision1_case2(void *pvParameters) {
  int danger; // Déclare une variable pour indiquer le danger
  while (1) { // Boucle infinie
    if (xQueueReceive(motionEventQueue, &danger, portMAX_DELAY) == pdPASS) {
// Vérifie s'il y a un événement de mouvement dans la file
        Serial.println("Motion Detected! Ecran 1 était dans l'état allumé");
// Affiche "Motion Detected! Ecran 1 était dans l'état allumé" sur le moniteur
série
// Problème :
// Pourquoi ne pas créer qu'une seule fonction de gestion d'urgence ?
// Pourquoi séparer les deux cas ?
// Pourquoi ne pas simplement éteindre puis rallumer l'écran ?

// Explication :
// Dans notre cas actuel, cette approche fonctionne efficacement (pas besoin
de deux fonctions),
// car éteindre et rallumer un écran ne représente pas un coût significatif
// en termes de dépenses directes. Cependant, dans certains systèmes,
// une telle action peut être très coûteuse et entraîner des pertes
considérables.
// L'objectif ici est de sensibiliser à cette éventualité.

// Ainsi, nous avons créé deux fonctions pour gérer différemment les cas
d'urgence.
// Le cas d'urgence où l'écran 1 était éteint est moins grave, tandis que
// le cas d'urgence où l'écran 1 était allumé est plus grave
// car il nécessite à la fois l'extinction et l'allumage de l'écran (2 actions
nécessaires).

        lcd1.clear(); // Efface l'écran LCD 1
        lcd1.noBacklight(); // Éteint le rétroéclairage de l'écran LCD 1

        lcd1.init(); // Initialise l'écran LCD 1
        lcd1.backlight(); // Allume le rétroéclairage de l'écran LCD 1
        lcd1.setCursor(0, 0); // Définit la position du curseur de l'écran LCD
1
        lcd1.print("Message d'urgence!"); // Affiche "Message d'urgence!" sur
l'écran LCD 1
        xTimerStart(screenOffTimer4, 0); // Démarre le timer pour éteindre
l'écran LCD 1 après un certain délai
```

```
      }
    }
}


void setup() {
  Serial.begin(9600); // Initialise la communication série avec une vitesse de 9600 bauds
  pinMode(LUMINOSITY_SENSOR_PIN, INPUT); // Configure le pin du capteur de luminosité en entrée
  pinMode(MOTION_SENSOR_PIN, INPUT); // Configure le pin du capteur de mouvement en entrée

  luminosityEventQueue = xQueueCreate(5, sizeof(int)); // Crée une file d'événements de luminosité
  motionEventQueue = xQueueCreate(5, sizeof(int)); // Crée une file d'événements de mouvement

  tvSemaphore = xSemaphoreCreateCounting(2, 2); // Crée une sémaphore pour contrôler l'accès aux écrans

  screenOffTimer1 = xTimerCreate("Timer Ecran 1", pdMS_TO_TICKS(5000), pdFALSE, 0, turnOffScreen1);
  screenOffTimer2 = xTimerCreate("Timer Ecran 2", pdMS_TO_TICKS(5000), pdFALSE, 0, turnOffScreen2);
  screenOffTimer3 = xTimerCreate("Timer Ecran 3", pdMS_TO_TICKS(5000), pdFALSE, 0, turnOffScreen3); // Crée un timer pour éteindre l'écran LCD 1 après 5 secondes
  screenOffTimer4 = xTimerCreate("Timer Ecran 1 urgence", pdMS_TO_TICKS(1000), pdFALSE, 0, turnOffScreen1); // Crée un timer pour éteindre l'écran LCD 1 en cas d'urgence après 1 seconde

  attachInterrupt(digitalPinToInterrupt(MOTION_SENSOR_PIN), motionSensorISR, RISING); // Attache l'interruption au capteur de mouvement
  //Créeons les tâches sur un seul coeur ( le coeur 1) du processeur DUO_COEUR de ESP32. Nous devons programmés les tâches sur un seul coeur pour faire de la programmation concurentielle !
  xTaskCreatePinnedToCore(taskLuminosity, "Lire Luminosité", 1500, NULL, 2, NULL, 1); // Crée une tâche pour lire la luminosité
  xTaskCreatePinnedToCore(taskTelevision1, "Allumer télé 1", 1500, NULL, 1, NULL, 1);
  xTaskCreatePinnedToCore(taskTelevision1, "Allumer télé 2", 1500, NULL, 1, NULL, 1);
  xTaskCreatePinnedToCore(taskTelevision1, "Allumer télé 3", 1500, NULL, 1, NULL, 1);  // Crée une tâche pour allumer la télévision 1
  xTaskCreatePinnedToCore(taskTelevision1_case1, "Urgence télé 1 cas 1", 1500, NULL, 4, NULL, 1); // Crée une tâche pour gérer les événements de mouvement lorsque l'écran est éteint
```

```
  xTaskCreatePinnedToCore(taskTelevision1_case2, "Urgence télé 1 cas 2", 1500,
NULL, 4, NULL, 1); // Crée une tâche pour gérer les événements de mouvement
lorsque l'écran est allumé
}

void loop() {
  // Vide. Les tâches et les timers gèrent la logique.
}
```

Lien du projet : https://wokwi.com/projects/399805353510702081