25 MAI 2024

PROJET SYSTEMES MULTITACHES ET TEMPS REELS

Gestion Automatisés des télévisions dans une maison

Réalisé par :

-OGOULA Ted-Marlon

-OSSILA BALA Jennifer

-WOULOU LIHOUSSOU Casey-Inola

1-Analyse fonctionnelle du système de Gestion automatique des télévisions d'une maison

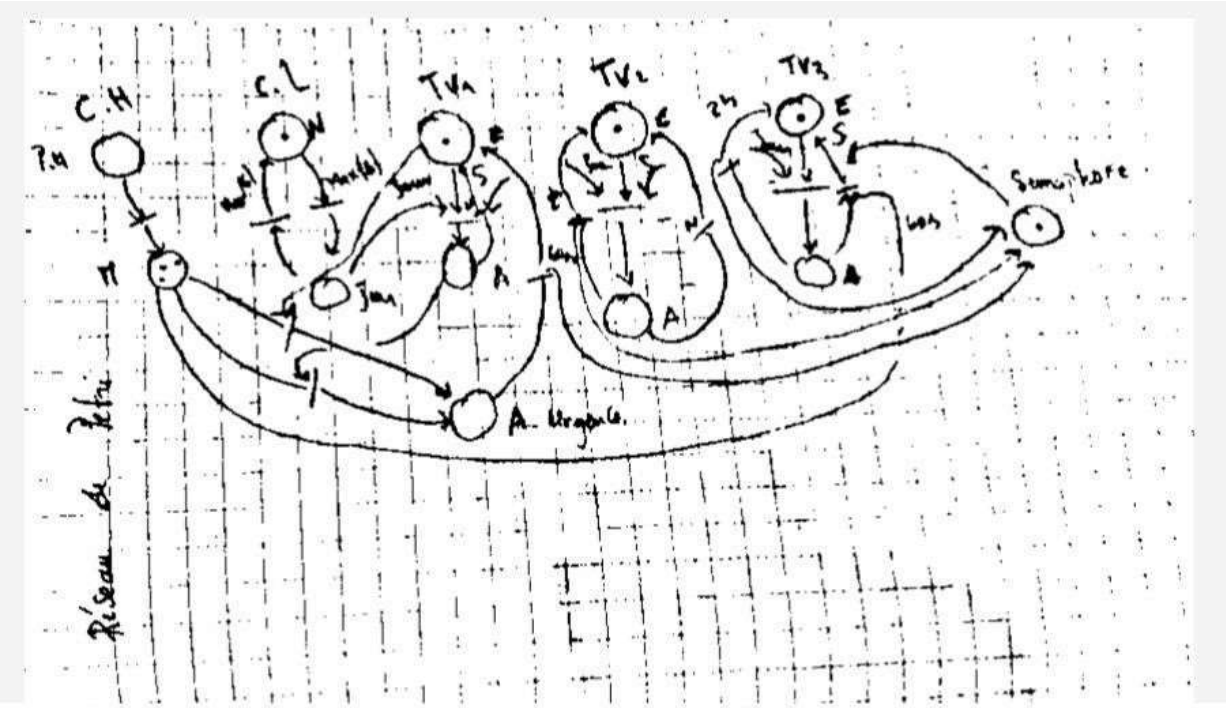
Dans l'optique de notre projet de fin de module intitulé Multitâches et temps réels, nous avons été conduits à modéliser un système de gestion de télévisions dans un ménage.
Il s'agit d'un système permettant de réduire de manière significative le gaspillage d'électricité dans les maisons et aussi de surveiller l'activité des plus jeunes vis à vis des écrans.
Nous pouvons appuyer cela par quelques chiffres alarmants, environ 98% des enfants en France de moins de 5 ans sont devant des écrans et cela peut conduire à des troubles psychologiques et des problèmes oculaires.
Ainsi, nous proposons un système permettant de contrôler ces écrans, de réduire considérablement les effets néfastes sur les usagers et aussi de protéger l'habitat en cas d'intrusion.

Tout d'abord, le système permettra que:

- Les écrans soient allumés uniquement en journée grâce à un capteur de luminosité pendant une durée maximale de 2 heures puis s'éteint automatiquement,
- Aussi, le système ne peut gérer que 2 écrans allumés simultanément,
- Enfin en cas de détection de mouvement pour le capteur de mouvement, un message d'urgence sera affiché par l'écran principal (TV1) pendant une minute.

2-Modélisation: Réseau de Pétri

Dans la suite de notre étude, nous nous sommes focalisés sur 3 télévisions afin de simplifier notre étude. Et nous avons obtenu le réseau de Pétri suivant :



3. Liste des tâches

Grâce au réseau de Pétri ci-dessus, il est possible de lister les différentes tâches de notre système.

Numérotation	Désignation	Priorités
T1	Lire le capteur de luminosité	2
T2	Allumer TV1	1
T3	Allumer TV2	1
T4	Allumer TV3	1
	Eteindre TV1	1

T5	Eteindre TV2	1
T6	Eteindre TV3	1
T7	Afficher message d'urgence	3
T8		
T9	Lire état du capteur de mouvement	3
T10	Prendre Sémaphore	2
T11	Vendre Sémaphore	2

NB : les tâches les plus prioritaires sont celles avec un grand indice de priorité.

4. code

Pour réaliser le code suivant, nous avons eu recours à site de simulation (Wokwi), notre code peut être compris comme suite :

```
#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 screenOffTimer4; // Timer pour éteindre l'écran 3
TimerHandle_t screenOffTimer3; // Timer pour éteindre l'écran 2
TimerHandle_t screenOffTimer1; // Timer pour éteindre l'écran 1
TimerHandle_t screenOffTimer2; // 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("Bonjour ! "); // Affiche "Bonjour " 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); // Libère la sémaphore
       lcd2.clear(); // Efface l'écran LCD 2
      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
}
// Fonction de rappel pour éteindre l'écran
void turnOffScreen3(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
}
// Fonction de rappel pour éteindre l'écran
void turnOffScreen4(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
      }
    }
  }
}
// Tâche de gestion de l'écran en cas d'allumage de la télévision
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 2
        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(screenOffTimer3, 0); // Démarre le timer pour éteindre l'écran LCD 2 
après un certain délai
        vTaskDelay(pdMS_TO_TICKS(2000)); // Attend 2 secondes
      }
    }
  }
}
// Tâche de gestion de l'écran en cas d'allumage de la télévision
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
```

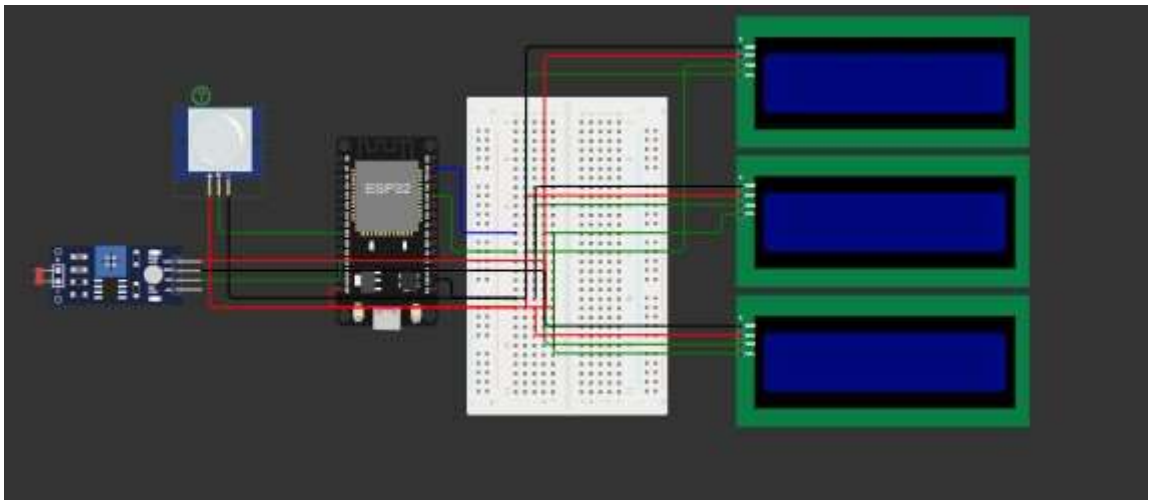
```
        lcd3.backlight(); // Allume le rétroéclairage de l'écran LCD 3
        lcd3.setCursor(0, 0); // Définit la position du curseur de l'écran LCD 3
        lcd3.print("TV3"); // Affiche "TV3" sur l'écran LCD 3
        xTimerStart(screenOffTimer4, 0); // Démarre le timer pour éteindre l'écran LCD 1
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


        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(screenOffTimer2, 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(screenOffTimer2, 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 éteint
void taskTelevision1_case3(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
          if(xSemaphoreTake(tvSemaphore, portMAX_DELAY) == NULL){
           Serial.println("Motion Detected! Ecran 1 "); // Affiche "Motion Detected! Ecran 1 était dans l'état éteint" sur le moniteur série
          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
          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(screenOffTimer2, 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); // Crée un timer pour éteindre l'écran LCD 1 après 5 secondes
  screenOffTimer2 = 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
```

```
  screenOffTimer3 = xTimerCreate("Timer Ecran 2", pdMS_TO_TICKS(5000), pdFALSE, 0, turnOffScreen3); // Crée un timer pour éteindre l'écran LCD 2 après 5 secondes
  screenOffTimer4 = xTimerCreate("Timer Ecran 3", pdMS_TO_TICKS(5000), pdFALSE, 0, turnOffScreen4); // Crée un timer pour éteindre l'écran LCD 3 après 5 secondes
  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); // 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
  xTaskCreatePinnedToCore(taskTelevision1_case3, "Urgence télé 1 cas 3", 1500, NULL, 4, NULL, 1); // Crée une tâche pour gérer les événements de mouvement lorsque 2 écrans sont  allumés
  xTaskCreatePinnedToCore(taskTelevision2, "Allumer télé 2", 1500, NULL, 1, NULL, 1); // Crée une tâche pour allumer la télévision 2
  xTaskCreatePinnedToCore(taskTelevision3, "Allumer télé 3", 1500, NULL, 1, NULL, 1); // Crée une tâche pour allumer la télévision 3
}

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

Montage :



5. Tests et Perspectives

Durant les tests ce programme, nous avons eu du mal avec l'évènement 3 où les écrans 2 et 3 sont allumés et il y'a détection d'un intrus. Les écrans ne s'arrêtent pas mais exceptionnellement les 3 écrans sont allumés pour afficher le message d'urgence.

Pour améliorer ce code, l'utilisateur pourra commander ces télévisions par une application via Bluetooth ou un réseau Wi-Fi. On peut ajouter un buzzer pour signaler lorsqu'un mouvement sera détecté ou même placer une caméra qui va capturer les images lorsqu'il y'a un mouvement puis l'envoie l'image sur le téléphone du propriétaire. Autant de possibilité pour rendre ce système et ce code plus optimal.