Conception et développement d’un système temps réel et multitâche

Membre du groupe :

Isselmou el werie mbarka

Ethmane sidi cheikh fatimetou

El boukhary mohamed

I. Analyse fonctionnelle

1. Gestion du Système de Temps Réel et Multitâche

- Ce système assure le bon fonctionnement simultané des différentes fonctionnalités des téléviseurs en attribuant des priorités aux tâches et en gérant les interruptions de manière efficace. Il garantit également une allocation appropriée du temps de traitement pour chaque tâche, en tenant compte de leur importance et de leur urgence.

2.Fonction Principale : Allumer le Téléviseur :

- Lorsqu'un signal de commande d'allumage est détecté, le système active les composants internes nécessaires pour démarrer le téléviseur, ce qui inclut notamment l'alimentation électrique, le circuit de démarrage et le chargement du système d'exploitation. Une fois que le téléviseur est démarré, une image est affichée sur l'écran pour indiquer son état allumé.

3. Allumage Simultané de Deux Téléviseurs :

- Les signaux d'allumage provenant de deux téléviseurs sont synchronisés et traités simultanément par le système. Cela garantit que les deux téléviseurs démarrent en même temps, ce qui est important pour une expérience utilisateur cohérente.

4. Allumage d'une Télévision Uniquement le Jour et Pendant Deux Heures :

- Avant d'allumer le téléviseur, le système vérifie d'abord s'il fait jour en utilisant un capteur de luminosité. Si la luminosité ambiante est suffisante pour indiquer qu'il fait jour, le système procède à l'allumage. De plus, une fois allumé, le téléviseur est programmé pour s'éteindre automatiquement après deux heures d'utilisation, ce qui contribue à économiser l'énergie.

5. Affichage d'un Message d'Urgence en Cas de Mouvement Détecté :

- Un capteur de mouvement surveille en permanence l'environnement pour détecter tout mouvement. Si un mouvement est détecté, le système affiche immédiatement un message d'urgence sur l'écran principal 1 du téléviseur. Ce message reste affiché pendant cinq minutes avant de disparaître automatiquement, ce qui permet d'attirer l'attention des utilisateurs sur la situation d'urgence détectée.

II. Analyse des contraintes de temps

- Contraintes de Temps Réel :

Les signaux d'allumage doivent être détectés et traités en temps réel pour garantir une réponse rapide lorsque l'utilisateur souhaite allumer un téléviseur.

La détection de mouvement et l'affichage du message d'urgence doivent également être réalisés en temps réel pour assurer une réaction immédiate en cas d'urgence.

Le système doit être capable de gérer plusieurs tâches simultanément tout en respectant des délais stricts pour garantir une expérience utilisateur fluide.

- Contraintes de Temps d'Allumage :

L'allumage d'une télévision uniquement pendant la journée impose une contrainte temporelle. Le système doit être capable de détecter le moment où il fait jour et d'allumer le téléviseur en conséquence.

De plus, une fois que le téléviseur est allumé, il doit être programmé pour s'éteindre automatiquement après deux heures, ce qui nécessite une gestion précise du temps pour suivre la durée d'utilisation.

- Contraintes de Durée d'Affichage du Message d'Urgence :

L'affichage du message d'urgence en cas de détection de mouvement doit respecter une durée prédéterminée de cinq minutes. Le système doit être capable d'activer et de désactiver l'affichage du message en respectant cette contrainte de temps.

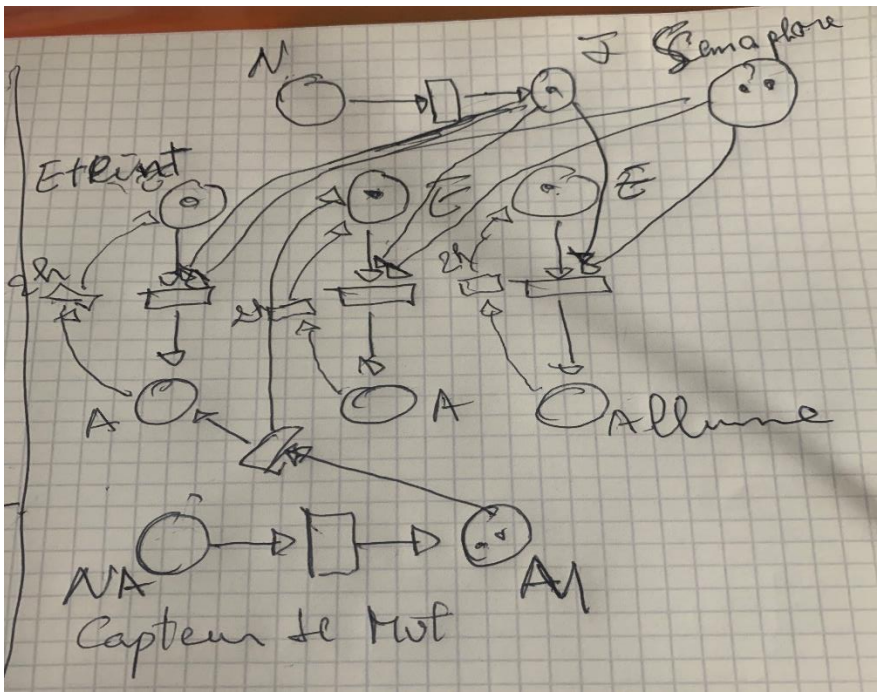
- Contraintes de Synchronisation :

L'allumage simultané de deux téléviseurs nécessite une synchronisation précise des signaux d'allumage. Le système doit garantir que les deux téléviseurs démarrent en même temps, ce qui impose une contrainte de temps sur le processus de démarrage.

- Contraintes de Latence :

La latence, c'est-à-dire le délai entre la détection d'un événement et la réaction du système, doit être maintenue à un niveau acceptable pour assurer une expérience utilisateur réactive. Toute latence excessive peut entraîner une expérience utilisateur médiocre, en particulier dans des situations critiques telles que l'affichage de messages d'urgence.

III. Conception par réseau de pétri



IV. La liste des taches et les priorités

- Détection du Signal d'Allumage (Priorité : Haute)

Recevoir et traiter les signaux d'allumage pour démarrer le téléviseur.

- Gestion du Système de Temps Réel et Multitâche (Priorité : Haute)

Coordination et exécution des tâches simultanées pour assurer le bon fonctionnement du système.

- Détection de Mouvement (Priorité : Haute)

Détecter les mouvements et déclencher l'affichage d'un message d'urgence en cas de besoin.

- Détection de la Luminosité Ambiante (Priorité : Moyenne)

Mesurer la luminosité pour déterminer si le téléviseur doit être allumé.

- Affichage du Message d'Urgence (Priorité : Moyenne)

Afficher un message d'urgence sur l'écran en cas de détection de mouvement.

- Activation des Composants Internes (Priorité : Basse)

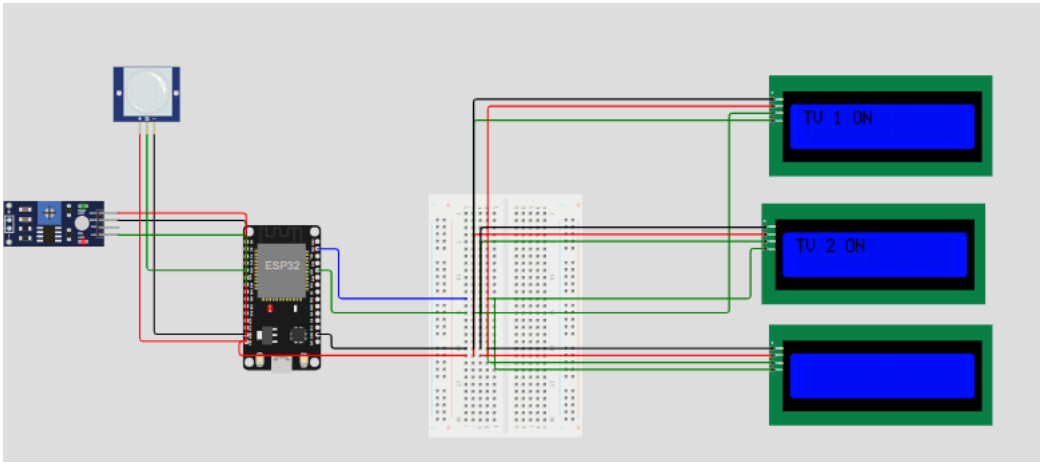
Activer les composants internes du téléviseur une fois le signal d'allumage reçu.

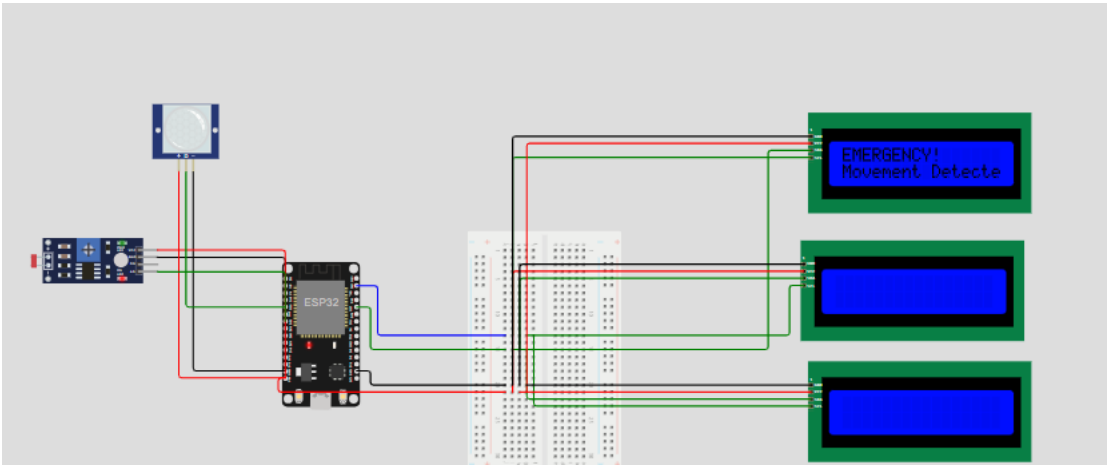
- Affichage de l'Image sur l'Écran (Priorité : Basse)

Afficher une image sur l'écran une fois que le téléviseur est démarré.

V. Codage avec freeRTOS

https://wokwi.com/projects/399617713236745217






```
#include <LiquidCrystal_I2C.h>
#include <Wire.h>
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "freertos/queue.h"
#include "freertos/semphr.h"

// Constants
```

```
#define LUMINOSITY_SENSOR_PIN 33
#define MOTION_SENSOR_PIN 27
#define TV_PIN1 25
#define TV_PIN2 26
#define TV_PIN3 27
#define SEUIL_LUMINOSITE 500

// I2C LCD display addresses
const int LCD1_ADDRESS = 0x27;
const int LCD2_ADDRESS = 0x28;
const int LCD3_ADDRESS = 0x29;

// I2C LCD displays
LiquidCrystal_I2C lcd1(LCD1_ADDRESS, 16, 2);
LiquidCrystal_I2C lcd2(LCD2_ADDRESS, 16, 2);
LiquidCrystal_I2C lcd3(LCD3_ADDRESS, 16, 2);

// Queues and Semaphores
QueueHandle_t luminosityEventQueue;
QueueHandle_t motionEventQueue;
SemaphoreHandle_t tvSemaphore;

// Timer Handles
TimerHandle_t screenOffTimer1;
TimerHandle_t emergencyTimer;

// Variables
bool isDaytime = false;
bool emergencyActive = false;
bool tv1On = false;
bool tv2On = false;
bool tv3On = false;
unsigned long emergencyStartTime = 0;

// Task prototypes
void taskLuminosity(void *pvParameters);
void taskTelevision1(void *pvParameters);
void taskTelevision2(void *pvParameters);
void taskTelevision3(void *pvParameters);
void taskEmergencyHandler(void *pvParameters);
void IRAM_ATTR motionSensorISR();
void turnOffScreen1(TimerHandle_t xTimer);
void emergencyTimeout(TimerHandle_t xTimer);
```

```
void setup() {
  Serial.begin(115200);

  // Initialize the I2C LCD displays
  lcd1.init();
  lcd2.init();
  lcd3.init();
  lcd1.backlight();
  lcd2.backlight();
  lcd3.backlight();
  lcd1.clear();
  lcd2.clear();
  lcd3.clear();

  // Initialize pins
  pinMode(LUMINOSITY_SENSOR_PIN, INPUT);
  pinMode(MOTION_SENSOR_PIN, INPUT);
  pinMode(TV_PIN1, OUTPUT);
  pinMode(TV_PIN2, OUTPUT);
  pinMode(TV_PIN3, OUTPUT);
  digitalWrite(TV_PIN1, LOW);
  digitalWrite(TV_PIN2, LOW);
  digitalWrite(TV_PIN3, LOW);

  // Create queues and semaphores
  luminosityEventQueue = xQueueCreate(5, sizeof(int));
  motionEventQueue = xQueueCreate(5, sizeof(int));
  tvSemaphore = xSemaphoreCreateCounting(2, 0); // Allow up to 2 TVs to be on at
the same time

  // Create timers
  screenOffTimer1 = xTimerCreate("Screen Off Timer 1", pdMS_TO_TICKS(5000),
pdFALSE, (void*)0, turnOffScreen1);
  emergencyTimer = xTimerCreate("Emergency Timer", pdMS_TO_TICKS(5 * 60 *
1000), pdFALSE, (void*)0, emergencyTimeout);

  // Attach ISR for motion sensor
  attachInterrupt(digitalPinToInterrupt(MOTION_SENSOR_PIN), motionSensorISR,
RISING);

  // Create tasks
  xTaskCreatePinnedToCore(taskLuminosity, "Luminosity Task", 2048, NULL, 1, NULL,
1);
```

```
 xTaskCreatePinnedToCore(taskTelevision1, "Television 1 Task", 2048, NULL, 1,
NULL, 1);
 xTaskCreatePinnedToCore(taskTelevision2, "Television 2 Task", 2048, NULL, 1,
NULL, 1);
 xTaskCreatePinnedToCore(taskTelevision3, "Television 3 Task", 2048, NULL, 1,
NULL, 1);
 xTaskCreatePinnedToCore(taskEmergencyHandler, "Emergency Handler Task",
2048, NULL, 2, NULL, 1);
}

void loop() {
 // Empty. Tasks and timers handle the logic.
}

void taskLuminosity(void *pvParameters) {
 int luminosity;
 while (1) {
  luminosity = analogRead(LUMINOSITY_SENSOR_PIN);
  float voltage = luminosity / 1024.0 * 5.0;
  float resistance = 2000 * voltage / (1 - voltage / 5);
  float luminosityValue = pow(50 * 1e3 * pow(10, 0.7) / resistance, (1 / 0.7));

  Serial.print("Luminosity: ");
  Serial.println(luminosityValue);

  if (luminosityValue >= SEUIL_LUMINOSITE) {
   isDaytime = true;
   xQueueSend(luminosityEventQueue, &luminosity, portMAX_DELAY);
  } else {
   isDaytime = false;
   xSemaphoreGive(tvSemaphore); // Ensure semaphore is released when not
daytime
   xSemaphoreGive(tvSemaphore); // Ensure semaphore is released when not
daytime
  }

  vTaskDelay(pdMS_TO_TICKS(1000));
 }
}

void taskTelevision1(void *pvParameters) {
 int luminosity;
 while (1) {
```

```
   if (xQueueReceive(luminosityEventQueue, &luminosity, portMAX_DELAY) ==
pdPASS) {
     if (xSemaphoreTake(tvSemaphore, portMAX_DELAY) == pdTRUE) {
      if (isDaytime && !emergencyActive) {
       digitalWrite(TV_PIN1, HIGH);
       lcd1.clear();
       lcd1.print("TV 1 ON");
       tv1On = true;
       vTaskDelay(pdMS_TO_TICKS(2 * 60 * 60 * 1000)); // TV on for 2 hours
       digitalWrite(TV_PIN1, LOW);
       lcd1.clear();
       tv1On = false;
       xSemaphoreGive(tvSemaphore);
      }
     }
   }
  }
}

void taskTelevision2(void *pvParameters) {
  int luminosity;
  while (1) {
   if (xQueueReceive(luminosityEventQueue, &luminosity, portMAX_DELAY) ==
pdPASS) {
     if (xSemaphoreTake(tvSemaphore, portMAX_DELAY) == pdTRUE) {
      if (isDaytime && !emergencyActive) {
       digitalWrite(TV_PIN2, HIGH);
       lcd2.clear();
       lcd2.print("TV 2 ON");
       tv2On = true;
       vTaskDelay(pdMS_TO_TICKS(2 * 60 * 60 * 1000)); // TV on for 2 hours
       digitalWrite(TV_PIN2, LOW);
       lcd2.clear();
       tv2On = false;
       xSemaphoreGive(tvSemaphore);
      }
     }
   }
  }
}

void taskTelevision3(void *pvParameters) {
  int luminosity;
  while (1) {
```

```
    if (xQueueReceive(luminosityEventQueue, &luminosity, portMAX_DELAY) ==
pdPASS) {
      if (xSemaphoreTake(tvSemaphore, portMAX_DELAY) == pdTRUE) {
        if (isDaytime && !emergencyActive) {
          digitalWrite(TV_PIN3, HIGH);
          lcd3.clear();
          lcd3.print("TV 3 ON");
          tv3On = true;
          vTaskDelay(pdMS_TO_TICKS(2 * 60 * 60 * 1000)); // TV on for 2 hours
          digitalWrite(TV_PIN3, LOW);
          lcd3.clear();
          tv3On = false;
          xSemaphoreGive(tvSemaphore);
        }
      }
    }
  }
}

void taskEmergencyHandler(void *pvParameters) {
  int danger;
  while (1) {
    if (xQueueReceive(motionEventQueue, &danger, portMAX_DELAY) == pdPASS) {
      emergencyActive = true;
      xTimerStart(emergencyTimer, 0);

      lcd1.clear();
      lcd1.backlight();
      lcd1.setCursor(0, 0);
      lcd1.print("EMERGENCY!");
      lcd1.setCursor(0, 1);
      lcd1.print("Movement Detected");

      if (tv1On) {
        digitalWrite(TV_PIN1, LOW);
        tv1On = false;
        xSemaphoreGive(tvSemaphore);
      }
      if (tv2On) {
        digitalWrite(TV_PIN2, LOW);
        tv2On = false;
        xSemaphoreGive(tvSemaphore);
      }
      if (tv3On) {
```

```
      digitalWrite(TV_PIN3, LOW);
      tv3On = false;
      xSemaphoreGive(tvSemaphore);
     }
    }
   }
}

void IRAM_ATTR motionSensorISR() {
 int danger = 1;
 BaseType_t xHigherPriorityTaskWoken = pdFALSE;
 xQueueSendFromISR(motionEventQueue, &danger, &xHigherPriorityTaskWoken);
 portYIELD_FROM_ISR(xHigherPriorityTaskWoken);
}

void turnOffScreen1(TimerHandle_t xTimer) {
 lcd1.clear();
 lcd1.noBacklight();
 xSemaphoreGive(tvSemaphore);
}

void emergencyTimeout(TimerHandle_t xTimer) {
 emergencyActive = false;
 lcd1.clear();
 lcd1.noBacklight();
 isDaytime = false; // Reset to prevent immediate TV turn on after emergency
}
```