

|  |
| --- |
| **Systèmes Embarqués II** |
| Multitâche |
| ISAT – EPHEC 2020-2021  Juan Alvarez et Olivier Grabenweger |

Table des matières

[1 Introduction 2](#_Toc60870045)

[2 Mindmapping - Logigramme 2](#_Toc60870046)

[a) Mindmap 2](#_Toc60870047)

[b) Logigramme 3](#_Toc60870048)

[3 Schéma de câblage 4](#_Toc60870049)

[a) Composants 4](#_Toc60870050)

[4 Code source 4](#_Toc60870051)

[a) Code ESP32 : 4](#_Toc60870052)

[5 Conclusion 8](#_Toc60870053)

[6 Annexes, bibliographie et illustrations 8](#_Toc60870054)

[a) Annexes 8](#_Toc60870055)

[b) Bibliographie 8](#_Toc60870056)

# Introduction

Ce TP sert d’introduction au multitâche avec l’ESP32 et à nous montrer la puissance de ce dernier après avoir passé tout le reste du quadrimestre avec l’ESP8266.

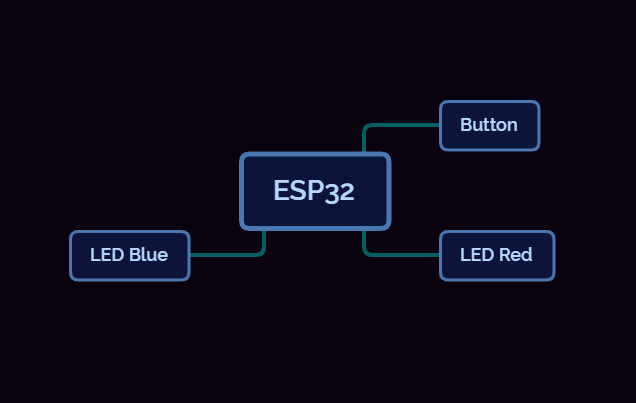
Nous allons utiliser free RTOS, un système d’exploitation temps réel qui permet l’ordonnance de tâches.

L’objectif est d’écrire un programme pour piloter trois périphériques différents : 2 LED et un bouton-poussoir.

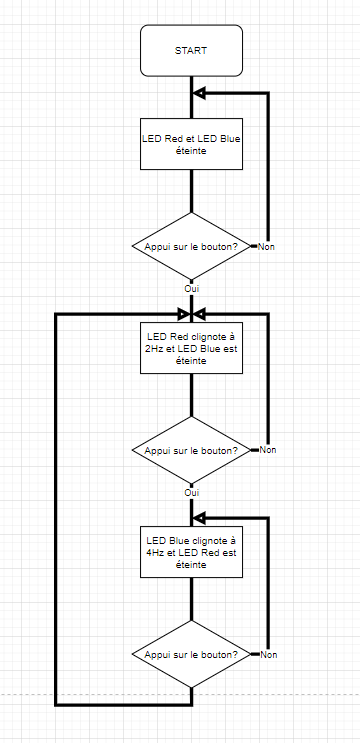
Au démarrage, les LED sont éteintes. Quand on appuie sur le bouton pour la première fois, la LED Red devra clignoter avec une période d’une demi seconde (500ms) tandis que la LED Blue devra rester éteinte. Au deuxième appui, c’est la LED Blue qui devra clignoter avec une période d’un quart de seconde cette fois (250ms) alors que la LED Red sera éteinte. Et le cycle se répètera indéfiniment.

# Mindmapping - Logigramme

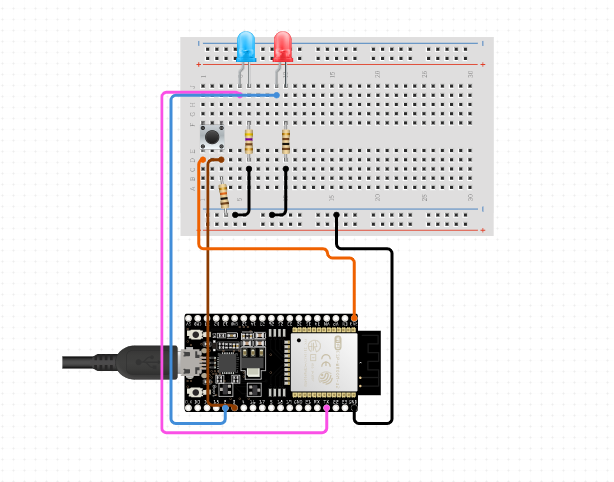
## Mindmap



## Logigramme



# Schéma de câblage



## Composants

* 2 LEDs => 1 rouge et 1 bleue
* 1 Bouton-poussoir
* 3 résistances => 1 de 10kΩ et 2 autres de 330Ω
* 1 bred board
* 1 ESP32

# Code source

## Code ESP32 :

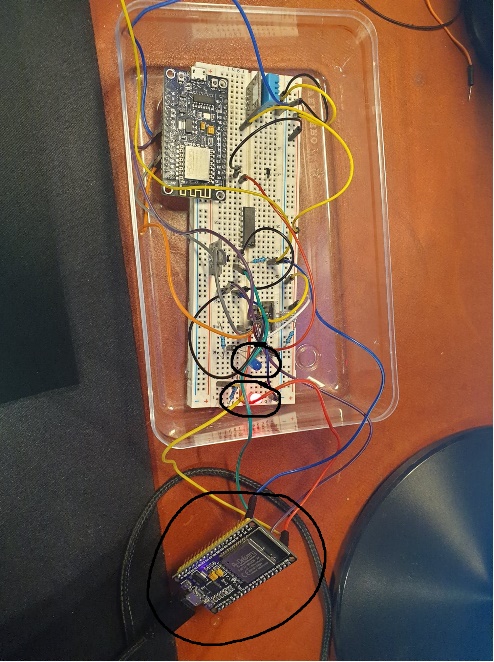
|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116  117  118  119  120  121  122  123  124  125  126  127  128  129  130  131  132  133  134  135  136  137  138  139  140  141  142  143  144  145  146  147  148  149  150  151  152  153  154  155  156  157  158  159  160  161  162  163  164  165  166  167  168  169  170  171  172  173  174  175  176  177  178  179  180  181  182  183  184  185  186  187  188  189  190  191  192  193  194  195  196  197  198  199  200 | /\* bibliothèques \*/  #include <Arduino.h>  /\* Variables \*/  **const** **int** BUTTON = GPIO\_NUM\_21; //Pin du bouton  **const** **int** LEDred = GPIO\_NUM\_22; //Pin de la LED rouge  **const** **int** LEDblue = GPIO\_NUM\_23; //Pin de la LED bleu    **int** flag\_button; //flag du bouton  **int** state\_button; //état du bouton  **int** count; //compteur    **static** QueueHandle\_t qh; //Création de notre file qh  /\* Tâche Bouton \*/  **void** **TaskButton**(**void** \*arg)  {  **int** bufferSendData; //variable de la donnée qui sera envoyée  **for**(;;)  {  state\_button = digitalRead(BUTTON); //lecture de l'état du bouton  **if** (state\_button == LOW) // Système avec flag qui permet d'éviter le rebond du bouton  {  flag\_button = **1**;  }  **if** (flag\_button == **1** && state\_button == HIGH)  {  count += **1**; //incrémentation du compteur  flag\_button = **0**;  }  **if** (count == **0**)  {  bufferSendData = **2**; //Valeur envoyée aux autres tâches pour que les 2 LED restent éteintes au démarrage  }  **else**  {  bufferSendData = count % **2**; //Valeur envoyée aux autres tâches (1 si count est impair et 0 si count est pair)  }  //(file, buffer valeur envoyée, Timeout)  xQueueSendToBack(qh, &bufferSendData, portMAX\_DELAY); //fonction pour envoyer la valeur dans la file  }  }  /\* Tâche LED Red \*/  **void** **TaskLEDRed**(**void** \*arg)  {  **int** bufferGetDataRed; //variable de la donnée reçue  **for**(;;)  {  //(file, buffer valeur reçue, Timeout)  xQueueReceive(qh,&bufferGetDataRed,portMAX\_DELAY); //fonction pour recevoir la valeur de la file  **if** (bufferGetDataRed == **1**) //si la valeur est égale à 1, la LED rouge clignote à une fréquence de 2Hz  {  digitalWrite(LEDred,**1**); //LED rouge s'allume  delay(**500**); //500ms de délai  digitalWrite(LEDred,**0**); //LED rouge s'éteind  delay(**500**); //500ms de délai  }  **else**  {  digitalWrite(LEDred,**0**); //LED rouge s'éteind  }  }  }  /\* Tâche LED Bleu \*/  **void** **TaskLEDBlue**(**void** \*arg)  {  **int** bufferGetDataBlue; //variable de la donnée reçue  **for**(;;)  {  //(file, buffer valeur reçue, Timeout)  xQueueReceive(qh,&bufferGetDataBlue,portMAX\_DELAY);  //fonction pour recevoir la valeur de la file  //si la valeur est égale à 0, la LED bleu clignote à une fréquence de 4Hz  **if** (bufferGetDataBlue == **0**)  {  digitalWrite(LEDblue,**1**); //LED bleu s'allume  delay(**250**); //250ms de délai  digitalWrite(LEDblue,**0**); //LED bleu s'éteind  delay(**250**); //250ms de délai  }  **else**  {  digitalWrite(LEDblue,**0**); //LED bleu s'éteind  }  }  }  /\* fonction setup \*/  **void** **setup**()  {  pinMode(LEDred, OUTPUT); // Initialise la LED rouge en output  digitalWrite(LEDred, LOW); //Eteind la LED rouge  pinMode(LEDblue, OUTPUT); // Initialise la LED Bleu en output  digitalWrite(LEDblue, LOW); //Eteind la LED bleu  pinMode(BUTTON, INPUT); // Initialise le bouton en input    //mise à zéro de quelques variables  flag\_button = **0**;  state\_button = **0**;  count = **0**;  qh = xQueueCreate(**3**,**8**); //paramètrage/configuration de la file qh  //(la) Tâche, le nom de la tâche, taille, paramètre, priorité, handle, CPU  xTaskCreatePinnedToCore(TaskButton,"TaskButton",**2048**,NULL,**1**,NULL,**1**);  xTaskCreatePinnedToCore(TaskLEDRed,"TaskLEDRed",**2048**,NULL,**1**,NULL,**1**);  xTaskCreatePinnedToCore(TaskLEDBlue,"TaskLEDBlue",**2048**,NULL,**1**,NULL,**1**);  }  **void** **loop**()  {  //Nothing else matters  } |

# Conclusion

En observant la puissance de l’ESP32 avec son dual core et la puissance du multitâche, on s’est rendu compte que le retour en arrière vers l’ESP8266 ne sera sans doute plus possible pour nous.   
Mais c’est loin d’être une perte. Avec le free RTOS et sa gestion des tâches en envoyant des données dans une file, on arrive aisément à faire tourner plusieurs tâches en même temps.   
Ceci nous a permis de nous passer de la fonction « Millis » que nous utilisions dans nos précédents TP pour ne pas être bloqué par des délais.

# Annexes, bibliographie et illustrations

## Annexes



## Bibliographie

* <https://www.circuito.io/> : schéma de câblage
* <http://draw.io/> : organigramme
* Xmind : Mindmap
* <http://www.esp32learning.com/code/esp32-and-freertos-example-create-a-task.php>
* <https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/system/freertos.html>
* <file:///C:/Users/olivi/Downloads/Multit%C3%A2che%20en%20pratique%20avec%20lESP32%20(Elektor%202020)%20(1).pdf>
* <https://cdn.shopify.com/s/files/1/1509/1638/files/ESP_-_32_NodeMCU_Developmentboard_Datenblatt_AZ-Delivery_Vertriebs_GmbH_10f68f6c-a9bb-49c6-a825-07979441739f.pdf?v=1598356497>