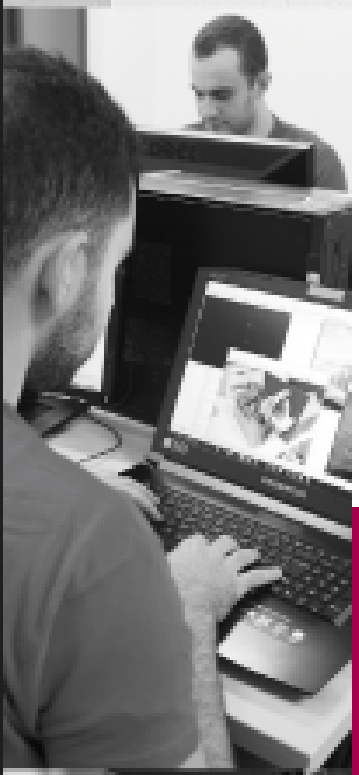


ENSEA

Beyond Engineering

Compte rendu TP3

Un framework multithreadé minimaliste

derridj mellyna weber loic

24 octobre 2025

7.1. Introduction - préparation

7.1.1. Programme en polling

```
void loop_TP3(void){

    while(1){
        int speed=0;// vitesse initialisée
        char string[32];
        int motif= ReadBTNMotif();
        if (ReadBTNSpeedUp()==1 ){ //augmentation de la vitesse du chenillard
            speed++;
        }
        if (ReadBTNSpeedDown()==1){// diminution de la vitesse du chenillard
            speed=speed-1;
        }

        sprintf(string, "Motif:%d | Vitesse:%d", motif, speed); // on stocke dans string le motif et la vitesse
        displayOnScreen(string);// affichage sur l'écran la vitesse et du motif


        displayLED( motif);//chenillard fait clignoter les leds selon le motif souhaité


    }

}
```

L'utilisation d'un programme en mode polling monopolise le microcontroleur (l'exécution de la tache dans la boucle while(1)). Si l'une des taches ne se termine jamais, les suivantes ne pourront jamais s'exécuter.

Remarque on aurait pu utiliser aussi : while (ReadBTNSpeedUp()==1) speed++. Autrement dit : tant qu'on appuie sur le bouton ; on augmente la vitesse.

7.3. Le scheduler, usage du timer

7.3.1. Programme en polling

fréquence d'horloge alimentant le timer 4 : 80 Mhz
Pour obtenir : PSC=79 ARR=9999 on génère un évènement toutes les 10ms.

Plus grande chaine de caractères que l'on peut obtenir : sachant qu'on génère 115200 bits par secondes (soit 1152 pour 10ms). Puisque un caractère équivaut à un octet, on a besoin de 1152/10=115.2 donc 115 octets.
En considérant que l'on a 3 taches, chacune des taches a une période de 3.3ms (10/3)

```
*/
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
  /* USER CODE BEGIN Callback 0 */
    static int taskNumber=3;
    if(htim->Instance == TIM4){
        taskNumber=(taskNumber+1)%3;

        switch (taskNumber){
            case 0 : taskLED();
            break;
            case 1: taskButton();
            break;
            case 2: taskScreen();
            break;
            default :
            break;
        }
    }
```

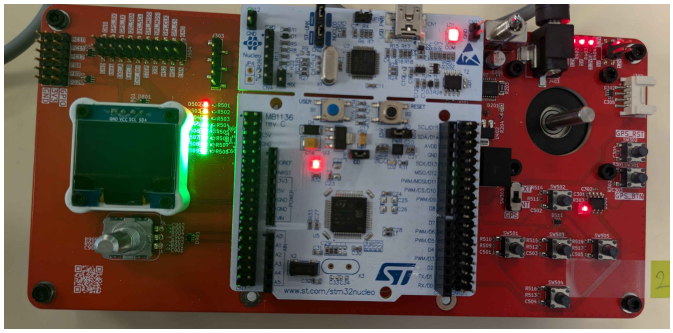
7.4. Tache de la gestion des Leds

7.4.1. Le driver logiciel des LEDs.

fonction Led update : (la fonction SHAL GPIO WRITEPIN permet d'allumer ou éteindre les Leds. Elle prend comme argument le port de la led, le numéro puis l'état (allumé ou éteint))

```
void LED_Update(){
    for (int i=0; i<=LED_BAR_SIZE; i++){
        if ( led_bar[i].isOn){
        HAL_GPIO_WritePin(led_bar[i].port,led_bar[i].pinNumber,GPIO_PIN_SET);
    }
        else{
        HAL_GPIO_WritePin(led_bar[i].port,led_bar[i].pinNumber,0);
        }
    }

}
```



7.4.2. La tache taskLED()

Code de la fonction taskLED :

```
void taskLED(){
    int numero_motif=(HAL_GetTick()/globalDelayInMs)%(tableau_motif[index_tableau_motif]->size);
    int moti= tableau_motif[index_tableau_motif]->motif[numero_motif];

    LED_Set_Value_With_Int(moti);
    LED_Update();


}
```

Dans ce programme, la variable `numero_motif` augmente en raison de la fonction `HAL_GetTick` (variable de temps qui augmente tandis que le reste est fixe), puis revient à 0 lorsqu'on a atteint la fin du motif, à cause de l'utilisation de l'opérateur `%`.

7.5 La tache de gestion des boutons.

```
void BUTTON_Update(){
    int val;

    for (int i=0; i<5;i++){
        if ( HAL_GPIO_ReadPin( joystick[i].port, joystick[i].pinNumber)==GPIO_PIN_RESET){

            // on regarde pour chaque interrupteur si on appuie dessus si c'est le cas
            // on change l'état de la led
            val=1;
            joystick[i].hasBeenPressed= (1-joystick[i].isOn)*val;
            joystick[i].isOn=val;
        }
        else{
            // si on appuie pas, l'état de la Led reste inchangé
            val=0;
            joystick[i].hasBeenPressed=(1-joystick[i].isOn)*val;
            joystick[i].isOn=val;
        }
    }

}
```

plusieurs cas de figures se présentent pour l'état d'un boutton. On représente ces différents états dans le tableau ci-contre :

GET_Pressed	isOn	état
0	0	led éteinte et on a toujours pas appuyé
1	0	on vient de relâcher (la led était allumée)
1	1	on a pas appuyé, la led reste allumée
0	1	on vient d'appuyer (la led était éteinte)

TABLE 1 – Etat de la LED