Université de Douala
École Supérieure Technique La Salle

Systèmes Temps Réel

Travaux pratiques : asservissement de température
d'un four sous RTOS (PIC18F45K22 — CCS C — Proteus)

Fiches de TP

Classe : **IIA5**

Pr Moffo Lonla Bertrand

Année académique 2026–2027

Table des matières

Organisation des travaux pratiques **2**

TP 1 — Prise en main du RTOS de CCS et de Proteus **3**

TP 2 — Projet, phase 1 : acquisition, identification et réglage **6**

TP 3 — Projet, phase 2 : régulation multitâche et validation **11**

Annexe — Rapport de projet et grille d'évaluation **13**

Organisation des travaux pratiques

Les 8 heures de TP sont consacrées à un projet unique, réalisé en binôme : l'asservissement de température d'un four à micro-ondes (modèle `OVEN` de Proteus) par un PIC18F45K22 programmé avec le RTOS du compilateur CCS C (chapitre 5 du cours).

Fiche	Contenu	Séance	Durée
TP 1	Prise en main : RTOS de CCS, Proteus, mesure des tâches	7 (16/01/2027)	1 h + préparation
TP 2	Projet, phase 1 : matériel, acquisition, identification, réglage du PID	8 (23/01/2027)	4 h
TP 3	Projet, phase 2 : régulation multitâche, sécurité, validation	9 (30/01/2027)	3 h
Annexe	Rapport de projet et grille d'évaluation	après la séance 9	

Matériel et logiciels. CCS C (IDE PCWH, compilateur PCH pour PIC18), Proteus 8 (ISIS), Python 3 (ou Matlab) pour l'identification et la simulation de la boucle fermée. Programmes fournis (dossier `code/ccs/`) : `tp1_mesure.c`, `four_phase1.c`, `four.c` ; outil `scripts/identif_broida.py`.

Préparation. Chaque séance se prépare : lire la fiche et la partie du cours indiquée, répondre aux questions de préparation *avant* la séance. Le compte rendu de chaque TP (mesures, captures d'écran, réponses) alimente le rapport final.

Règles de travail. Sauvegarder chaque version du programme sous un nom différent ; ne modifier qu'une chose à la fois ; noter systématiquement les conditions de chaque mesure (période, paramètres, version du programme).

TP 1 — Prise en main du RTOS de CCS et de Proteus

Séance 7 — 1 h en salle + préparation — Cours : chapitre 4

Objectifs. Créer un projet CCS multitâche, le simuler sous Proteus, **mesurer** la période, la durée et la gigue de chaque tâche, et observer les conséquences d'une tâche trop longue dans un ordonnanceur coopératif.

Préparation (à faire avant la séance).

1. Relire les directives `#use rtos` et `#task` et les fonctions `rtos_stats`, `rtos_overrun`.
2. Lire le programme `tp1_mesure.c` ci-dessous. Pour chaque tâche, prévoir la période du signal observé sur sa broche (RB0, RB1, RC0, RC1, RC2).

`tp1_mesure.c` *(CCS C — PIC18F45K22)*

```c
// tp1_mesure.c -- TP 1 : mesure des périodes et des durées des tâches
// Chaque tâche met une broche de test à 1 en entrée et à 0 en sortie :
// l'analyseur logique de Proteus donne directement période, durée et gigue.
#include <18F45K22.h>
#fuses NOWDT, NOPBADEN, NOLVP, NOXINST
#use delay(clock=64MHz, crystal=16MHz)
#use rs232(baud=115200, UART1, bits=8, parity=N) // 87 µs par caractère
#use rtos(timer=0, minor_cycle=10ms, statistics)

#define TP_RAPIDE  PIN_C0                    // broches de test (analyseur logique)
#define TP_CALCUL  PIN_C1
#define TP_LENTE   PIN_C2

struct rtos_stats st;
int8 charge = 20;                            // nombre d'itérations du calcul

#task(rate=250ms, max=1ms)
void Live() { output_toggle(PIN_B0); }

#task(rate=10ms, max=1ms)
void Rapide()
{
    output_high(TP_RAPIDE);
    output_toggle(PIN_B1);
    output_low(TP_RAPIDE);
}

#task(rate=50ms, max=5ms)
void Calcul()
{
    float x = 1.0;
    int8 i;
    output_high(TP_CALCUL);
    for (i = 0; i < charge; i++) x = x * 1.01 + 0.5;   // calcul flottant
```

```
    output_low(TP_CALCUL);
}

#task(rate=1s, max=5ms)
void Lente()                                    // question 5 : ajouter delay_ms(15)
{
    output_high(TP_LENTE);
    // delay_ms(15);
    output_low(TP_LENTE);
}

#task(rate=2s, max=5ms)
void Rapport()
{
    rtos_stats(Calcul, &st);
    printf("max=%lu t x%lu00ns ov=%u\r\n",          // < 30 caractères : < 3 ms
           st.task_max_ticks, st.hns_per_tick, rtos_overrun(Calcul));
    if (kbhit()) {                              // '+' / '-' : modifier la charge
        char c = getc();
        if (c == '+' && charge < 200) charge += 20;
        if (c == '-' && charge > 20)  charge -= 20;
        printf("charge=%u\r\n", charge);
    }
}

void main()
{
    set_tris_b(0x00);
    rtos_run();
}
```

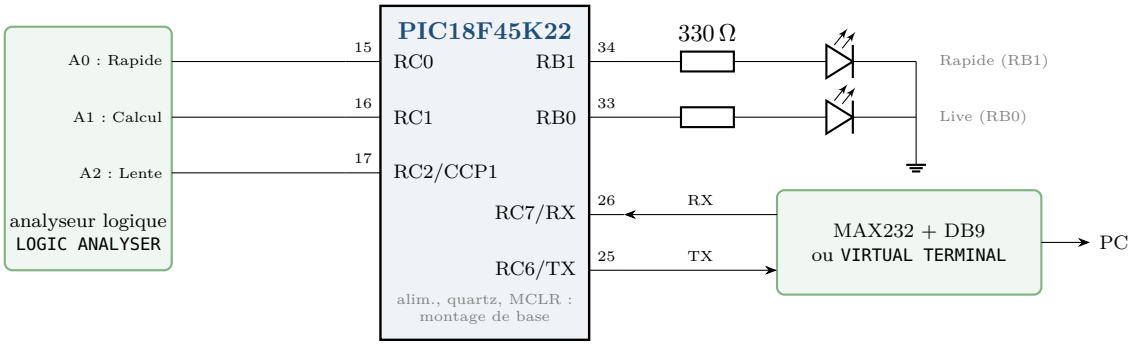


FIGURE 1.1 – Montage du TP 1 : LED sur RB0 et RB1, broches de test RC0–RC2 vers l'analyseur logique, liaison série (montage de base non représenté).

Exercice 1 — Création du projet et du schéma

1. Dans PCWH : créer le projet `tp1_mesure` pour le PIC18F45K22, copier le programme, compiler. Relever la taille du code (ROM) et de la RAM utilisées (fenêtre de compilation).
2. Dans Proteus : placer le PIC18F45K22, deux LED (RB0, RB1) avec résistances de 330 Ω, un `VIRTUAL TERMINAL` (RC6/TX → RXD, RC7/RX ← TXD, 115 200 bauds 8N1) et un analyseur logique (`LOGIC ANALYSER`) relié à RC0, RC1, RC2 et RB0.
3. Propriétés du PIC : *Program File* = `tp1_mesure.cof`, *Processor Clock Frequency* = 64 MHz. Lancer la simulation.

Exercice 2 — Mesure des périodes et des durées

1. Mesurer à l'analyseur logique la période de RB0 et de RC0. Comparer aux valeurs prévues.

2. Mesurer la durée des impulsions sur RC0 (tâche `Rapide`) et RC1 (tâche `Calcul`). Comparer à la valeur `max` déclarée et à la valeur affichée par `Rapport` sur le terminal (durée = ticks × hns_per_tick × 100 ns).
3. Dans un même cycle mineur, plusieurs tâches peuvent être échues. Zoomer à un instant multiple de 1 s et relever l'ordre d'exécution des tâches et le décalage (gigue) de `Rapide`.

> **Remarque — à consigner dans le compte rendu** Tableau : tâche, `rate`, `max`, période mesurée, durée mesurée (analyseur), durée `rtos_stats`, gigue maximale observée.

Exercice 3 — Dépassements

1. Taper plusieurs fois `+` dans le terminal pour augmenter la charge de `Calcul`. À partir de quelle valeur `rtos_overrun(Calcul)` devient-il vrai ? Relier à la durée mesurée et au paramètre `max`.
2. Continuer jusqu'à ce que la durée de `Calcul` dépasse 10 ms. Observer la période de RC0 (`Rapide`). Expliquer.
3. Revenir à la charge initiale, décommenter `delay_ms(15)` dans `Lente`, recompiler. Observer RC0 au voisinage de chaque seconde. Expliquer et proposer une écriture correcte d'une tâche qui doit « attendre ».

TP 2 — Projet, phase 1 : acquisition, identification et réglage

Séance 8 — 4 h — Cours : chapitre 5, §§ 5.2 à 5.5 — TD 7

Objectifs. Réaliser le matériel simulé du régulateur, mettre en œuvre les tâches d'acquisition, d'affichage et de communication en **boucle ouverte**, enregistrer la réponse indicielle du four, identifier son modèle et calculer les réglages du PID.

Préparation.

1. Relire la méthode de Broïda et les règles de réglage (cours, § 5.4–5.5) ; refaire l'exercice 2 du TD 7.
2. Calculer le code CAN correspondant à 25 °C, 100 °C et 250 °C (chaîne de mesure à 10 mV/°C, $V_{\text{ref}} = 5$ V).
3. Lire `four_phase1.c` (ci-dessous) et répondre : quelles tâches accèdent à `mesure` et `commande` ? Pourquoi la tâche `Terminal` n'envoie-t-elle qu'un caractère par activation ?

Exercice 1 — Schéma sous Proteus

45 min

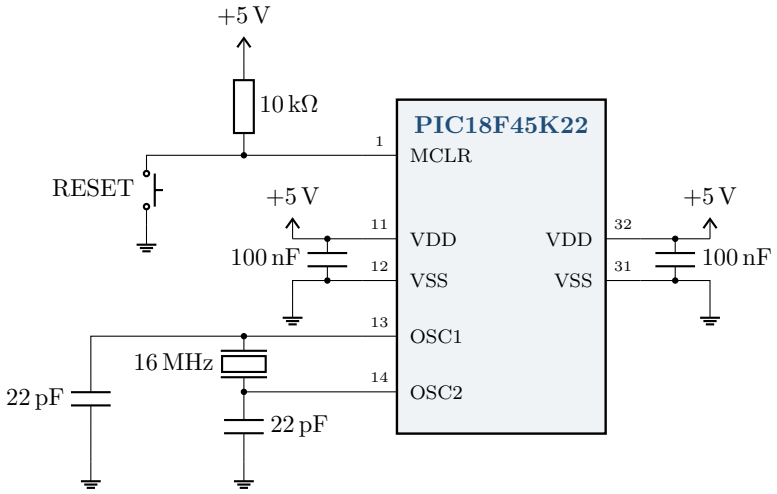


FIGURE 2.1 – Montage de base du PIC18F45K22 (sous Proteus, alimentation implicite ; quartz facultatif).

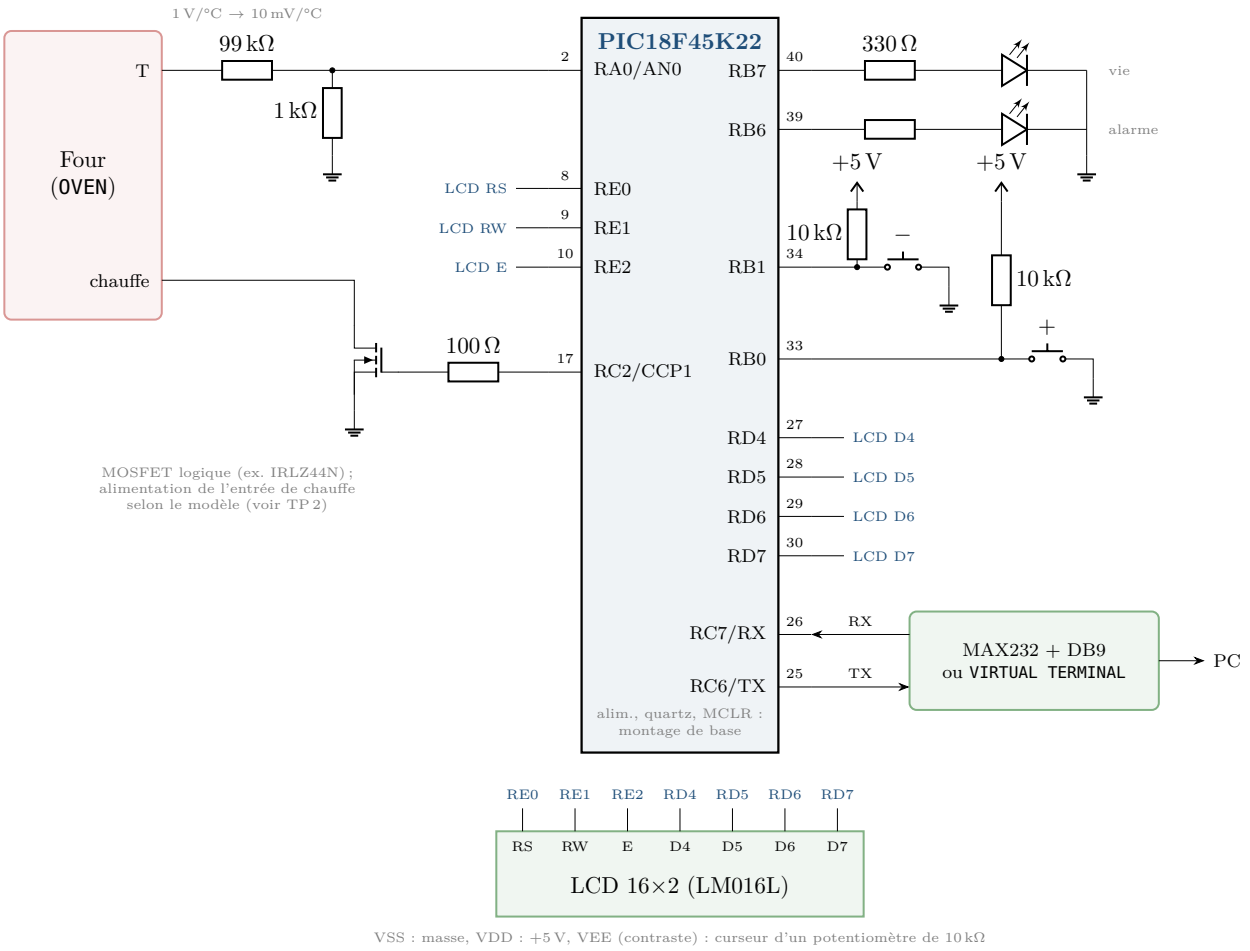


FIGURE 2.2 – Schéma du régulateur de température. En phase 1, les boutons et la LED d'alarme sont câblés mais pas encore utilisés.

1. Saisir le schéma des figures ci-dessus (montage de base, puis régulateur) : PIC18F45K22 (64 MHz) avec quartz, LCD `LM016L` (RS = RE0, RW = RE1, E = RE2, D4–D7 = RD4–RD7), LED de vie (RB7) et d'alarme (RB6), boutons sur RB0 et RB1 avec résistances de tirage, `VIRTUAL TERMINAL` sur l'UART1, modèle `OVEN`.
2. Ouvrir les propriétés du modèle `OVEN` et noter tous ses paramètres. Appliquer les valeurs imposées par l'enseignant (identiques pour tous les binômes).
3. **Étalonnage de la sortie T.** Relever la tension de la broche `T` du four à la température ambiante, puis après quelques minutes de chauffe (comparer à la température affichée par le modèle). En déduire l'échelle de la sortie (V/°C) et dimensionner l'adaptation (pont diviseur ou amplificateur) qui fournit **10 mV/°C** à l'entrée AN0, sans dépasser 5 V jusqu'à 500 °C.
4. Réaliser l'étage de commande de l'entrée de chauffe à partir de la sortie PWM RC2/CCP1 (transistor de commutation ou source commandée, selon l'entrée du modèle).

> **Remarque — à consigner dans le compte rendu** Schéma complet (capture), paramètres du four, calcul de l'adaptation de la mesure, schéma de l'étage de puissance.

Exercice 2 — Programme de la phase 1 (boucle ouverte) 45 min

`four_phase1.c` *(CCS C — PIC18F45K22)*

```
1 // four_phase1.c -- TP 2 (phase 1) : boucle ouverte, acquisition et identification
2 // Tâches : Live, Mesure, Afficheur, Terminal.  Commande "U<valeur><CR>" (0..100 %)
```

```
// Trame envoyée chaque seconde : "t;T;U"  (secondes ; °C ; %) -> fichier CSV
#include <18F45K22.h>
#device ADC=10
#fuses NOWDT, NOPBADEN, NOLVP, NOXINST, NOPROTECT
#use delay(clock=64MHz, crystal=16MHz)
#use rs232(baud=9600, UART1, bits=8, parity=N, errors)
#use rtos(timer=0, minor_cycle=10ms, statistics)
#include <stdlib.h>
#include <string.h>

#define LCD_ENABLE_PIN PIN_E2
#define LCD_RS_PIN     PIN_E0
#define LCD_RW_PIN     PIN_E1
#define LCD_DATA4      PIN_D4
#define LCD_DATA5      PIN_D5
#define LCD_DATA6      PIN_D6
#define LCD_DATA7      PIN_D7
#include <lcd.c>

#define LED_VIE  PIN_B7

float mesure = 25.0;       // °C
float commande = 0.0;      // % (boucle ouverte)
int8  sem_etat;
int32 secondes = 0;

#define RXN 8
char rx_buf[RXN];
int8 rx_w = 0, rx_r = 0;

#int_rda
void isr_rx(void)
{
    rx_buf[rx_w] = getc();
    rx_w = (rx_w + 1) % RXN;
}

char tx_buf[32];
int8 tx_i = 0, tx_n = 0;

#task(rate=200ms, max=1ms)
void Live() { output_toggle(LED_VIE); }

#task(rate=100ms, max=1ms)
void Mesure()
{
    int16 somme = 0;
    int8 i;
    for (i = 0; i < 4; i++) { delay_us(10); somme += read_adc(); }
    rtos_wait(sem_etat);
    mesure = (float)somme * (500.0 / 1023.0 / 4.0);   // 10 mV/°C
    rtos_signal(sem_etat);
}

#task(rate=250ms, max=5ms)
void Afficheur()
{
    static int1 ligne = 0;
    float y, u;
    rtos_wait(sem_etat);  y = mesure;  u = commande;  rtos_signal(sem_etat);
    if (ligne == 0) { lcd_gotoxy(1, 1); printf(lcd_putc, "T = %5.1f%cC    ", y, 0xDF); }
    else            { lcd_gotoxy(1, 2); printf(lcd_putc, "U = %3.0f %% (BO) ", u); }
    ligne = !ligne;
}

```

```
#task(rate=10ms, max=1ms)
void Terminal()
{
    static char cmd[6];
    static int8 n = 0, compteur = 0;
    static int1 prete = 0;
    char c;
    int16 val;
    float y, u;
    while (rx_r != rx_w) {                    // commandes reçues
        c = rx_buf[rx_r];
        rx_r = (rx_r + 1) % RXN;
        if (c == '\r') { cmd[n] = 0; n = 0; prete = 1; }
        else if (n < 5) cmd[n++] = c;
    }
    if (prete) {
        prete = 0;
        if (cmd[0] == 'U') {
            val = atol(cmd + 1);
            if (val >= 0 && val <= 100) {
                rtos_wait(sem_etat);  commande = val;  rtos_signal(sem_etat);
                set_pwm1_duty(val * 10);      // application immédiate de l'échelon
            }
        }
    }
    if (tx_i < tx_n) {
        putc(tx_buf[tx_i++]);                 // un caractère par activation
    } else if (++compteur >= 100) {           // chaque seconde
        compteur = 0;
        secondes++;
        rtos_wait(sem_etat);  y = mesure;  u = commande;  rtos_signal(sem_etat);
        sprintf(tx_buf, "%lu;%.1f;%.0f\r\n", secondes, y, u);
        tx_n = strlen(tx_buf);
        tx_i = 0;
    }
}

void main()
{
    setup_adc_ports(sAN0);
    setup_adc(ADC_CLOCK_DIV_64);
    set_adc_channel(0);
    setup_timer_2(T2_DIV_BY_16, 249, 1);      // PWM 4 kHz, 1000 pas
    setup_ccp1(CCP_PWM);
    set_pwm1_duty(0);
    lcd_init();
    sem_etat = 1;
    enable_interrupts(INT_RDA);
    enable_interrupts(GLOBAL);
    rtos_run();
}
```

1. Compiler et simuler. Vérifier l'affichage de la température ambiante sur le LCD et l'arrivée des trames `t;T;U` chaque seconde sur le terminal.
2. Envoyer `U30` puis `U0` : vérifier le rapport cyclique de la PWM à l'oscilloscope (période, largeur d'impulsion) et le comportement de la température.
3. Mesurer à l'analyseur logique la durée de la tâche `Afficheur` (ajouter une broche de test). Vérifier qu'elle reste inférieure à `max`.

Exercice 3 — Essai indiciel et identification 1 h

1. Laisser le four se stabiliser à l'ambiante (`U0`), puis appliquer l'échelon `U50`. Enregistrer les trames jusqu'à la stabilisation (au moins 5 constantes de temps). Copier le contenu du terminal

dans un fichier `essai.csv`.

2. Tracer la réponse (tableur, Python ou Matlab). Déterminer graphiquement $\Delta\theta_\infty$, t_{28} et t_{40} et en déduire K, τ, θ.
3. Vérifier avec l'outil fourni : `python3 scripts/identif_broida.py essai.csv`. Comparer.
4. Pourquoi faut-il partir d'un état d'équilibre ? Pourquoi ne pas choisir un échelon de 100 % ?

> **Remarque — à consigner dans le compte rendu** Courbe de réponse annotée (t_{28}, t_{40}, valeurs finales), calculs de Broïda, sortie de l'outil, modèle retenu.

Exercice 4 — Réglage du PID et simulation 1 h

1. Calculer les réglages PID de Ziegler-Nichols et de Chien-Hrones-Reswick (0 %) pour votre modèle.
2. Choisir la période d'échantillonnage T_e de la tâche PID et vérifier qu'elle est un multiple du cycle mineur.
3. Simuler la boucle fermée (modèle identifié + PID discret saturé à $[0, 100]$ %, avec et sans anti-windup) pour un échelon de consigne de 25 à 150 °C, sous Simulink ou en adaptant `scripts/pid_sim.py` (variables `K`, `TAU`, `THETA`, `TE`). Relever dépassement et temps de réponse à 5 %.
4. Choisir le réglage qui sera implanté en phase 2 et justifier au regard du cahier des charges.

> **Remarque — à consigner dans le compte rendu** Tableau des réglages, courbes simulées, choix motivé de K_p, T_i, T_d, T_e.

TP 3 — Projet, phase 2 : régulation multitâche et validation

Séance 9 — 3 h — Cours : chapitre 5, §§ 5.6 à 5.8

Objectifs. Intégrer la régulation PID, le réglage de la consigne et la surveillance de sécurité dans l'application multitâche ; valider les performances de la régulation et le comportement temporel ; préparer le rapport.

Préparation. Lire le programme complet `four.c` (cours, § 5.7). Le montage est celui du TP 2 (figure 2.2). Y reporter *vos* réglages de la phase 1 (`KP`, `TI`, `TD`, `TE`) et vérifier que `rate` de la tâche `PID` est égal à T_e. Compléter la table des tâches (période, rôle, ressources partagées utilisées).

Exercice 1 — Intégration et premiers essais

45 min

1. Compiler `four.c` avec vos paramètres et le simuler sur le schéma de la phase 1.
2. Vérifier les fonctions : LED de vie, réglage de la consigne par les boutons (pas de 5 °C, bornes 30 et 250 °C), commande `S120` depuis le terminal, affichage des deux lignes du LCD, trames `W=... T=... U=...` chaque seconde.
3. Vérifier que deux appuis rapprochés sur un bouton et une commande série simultanée ne corrompent pas la consigne. Quel mécanisme le garantit ?

Exercice 2 — Performances de la régulation

1 h

1. Depuis l'équilibre à l'ambiante, appliquer la consigne 150 °C. Enregistrer les trames (fichier CSV) et tracer $\theta(t)$ et $u(t)$. Mesurer le dépassement, le temps de montée et le temps de réponse à 5 %, l'erreur statique.
2. Comparer avec la simulation de la phase 1. Expliquer les écarts éventuels.
3. Appliquer ensuite des échelons de consigne 150 → 200 → 100 °C. Commenter la descente : pourquoi est-elle plus lente que la montée ?
4. Désactiver l'anti-windup (remplacer la condition d'intégration par `1`) et refaire l'essai 25 → 150 °C. Conclure.

> **Remarque — à consigner dans le compte rendu** Courbes $\theta(t)$ et $u(t)$ mesurées superposées à la simulation, tableau des indicateurs, analyse.

Exercice 3 — Sécurité et validation temporelle

45 min

1. **Test de la sécurité.** Pour ne pas attendre 260 °C, fixer temporairement `T_MAX` à 160 °C et appliquer la consigne 200 °C. Vérifier la coupure de la PWM, l'allumage de la LED d'alarme, l'affichage `ALARME`, l'impossibilité de réarmer (`R`) au-dessus de $T_{\max} - 20$ °C, puis le réarmement en dessous.

2. Mesurer à l'analyseur logique le délai entre l'instant où la mesure dépasse le seuil et la coupure de la PWM (broche de test basculée par `Mesure` quand la valeur dépasse `T_MAX`, sortie RC2). Le cahier des charges ($\leq$ 200 ms) est-il respecté ?
3. **Temps d'exécution.** Ajouter une tâche `Diagnostic` (période 5 s) qui envoie, pour chaque tâche, la durée maximale mesurée par `rtos_stats` et l'indicateur `rtos_overrun`. Compléter le tableau de la table des tâches du cours (§ 5.6) avec les valeurs *mesurées* et recalculer la charge U et la somme des durées dans le pire cycle mineur.

> **Remarque — à consigner dans le compte rendu** Chronogramme du test de sécurité, délai mesuré, tableau des durées mesurées, charge et pire cycle mineur recalculés.

Exercice 4 — Synthèse 30 min

Rédiger, pour le rapport, une conclusion d'une demi-page répondant aux questions : le système respecte-t-il le cahier des charges (fonctionnel, performances, temps réel, sécurité) ? Quelles sont les limites de l'ordonnancement coopératif du RTOS de CCS pour cette application ? Que changerait le passage à un noyau préemptif (FreeRTOS) ?

Annexe — Rapport de projet et grille d'évaluation

À remettre en binôme après la séance 9 (date fixée par l'enseignant) — rapport PDF + projets CCS et Proteus

Plan du rapport (15 à 25 pages hors annexes)

1. **Introduction** : contexte, objectifs, cahier des charges reformulé.
2. **Architecture matérielle** : schéma Proteus commenté, chaîne de mesure (étalonnage, calcul de l'adaptation), chaîne d'action (PWM, étage de puissance).
3. **Modélisation et identification** : protocole de l'essai indiciel, courbe annotée, méthode de Broïda, modèle retenu.
4. **Synthèse du correcteur** : réglages ZN et CHR, choix de T_e, discrétisation, anti-windup, résultats de simulation.
5. **Conception multitâche** : table des tâches (rôle, période, durée estimée puis mesurée), graphe des flots de données, ressources partagées et sémaphores, choix du cycle mineur, analyse temporelle (charge, pire cycle mineur, délai de la sécurité).
6. **Réalisation** : extraits de code commentés (le code complet en annexe).
7. **Validation** : essais fonctionnels, performances mesurées comparées à la simulation, mesures temporelles, test de sécurité.
8. **Conclusion** : bilan au regard du cahier des charges, difficultés rencontrées et solutions, limites, perspectives (noyau préemptif, chien de garde, passage sur carte réelle).

Règles de forme : figures numérotées, légendées et citées dans le texte ; unités systématiques ; tableaux de mesures avec les conditions d'essai ; terminologie du cours (tâche, période, échéance, cycle mineur, section critique…).

Grille d'évaluation (sur 20)

Critère	Points
Schéma Proteus complet et fonctionnel ; adaptation de la mesure justifiée	2
Essai indiciel et identification (protocole, courbe, calculs, cohérence)	3
Synthèse du PID : réglages, choix de T_e, anti-windup, simulation	3
Conception multitâche : table des tâches, synchronisation, analyse temporelle	4
Code : structure, commentaires, respect des règles du RTOS (pas d'attente active, opérations lentes découpées)	2
Validation : performances mesurées vs simulées, mesures temporelles, test de sécurité	3
Qualité du rapport (rédaction, figures, unités, conclusion critique)	2
Démonstration et réponses aux questions	1
Total	**20**

Liste de contrôle avant remise

- ☐ Le projet CCS compile sans avertissement ; le `.cof` chargé dans Proteus est celui de la dernière version.
- ☐ Le fichier Proteus fourni s'ouvre et simule sans modification (chemin du programme relatif).
- ☐ Les paramètres du modèle `OVEN` et les réglages du PID figurent dans le rapport.
- ☐ Chaque courbe indique les conditions d'essai (consigne, réglage, T_e, version du programme).
- ☐ Le tableau des tâches contient les durées *mesurées*.
- ☐ `T_MAX` a été remis à 260 °C après le test de sécurité.