driver_temp.h

```
float TEMP_GetCurrentValue();
void TEMP_Init();
```

---

driver_temp.c

```
static TaskHandle_t TEMP_TaskHandle;
static QueueHandle_t TEMP_MailboxHandle;
void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef* hadc){
BaseType_t woken = pdFALSE;
vTaskNotifyGiveFromISR(TEMP_TaskHandle, &woken);
portYIELD_FROM_ISR(woken);
}
void TEMP_Task(void *parameters){
while(1){
HAL_ADC_Start_IT(&hadc1);
ulTaskNotifyTake(pdTRUE, portMAX_DELAY);
float value = HAL_ADC_GetValue(&hadc1);
value *= MAX_VOLTAGE/RESOLUTION;
value *= 100;
xQueueOverwrite(TEMP_MailboxHandle, &value);
vTaskDelay(pdMS_TO_TICKS(200));
}}
float TEMP_GetCurrentValue(){
float temp = 0.0;
xQueuePeek(TEMP_MailboxHandle, &temp, portMAX_DELAY);
return temp;
}
void TEMP_Init(){
TEMP_MailboxHandle = xQueueCreate(1, sizeof(float));
xTaskCreate(TEMP_Task, "TEMP_Task", 64, NULL, 2, &TEMP_TaskHandle);
}
```

---

driver_uart.c

```
// TRANSMIT
static TaskHandle_t UART_TransmitTaskHandle;
static QueueHandle_t UART_TransmitQueueHandle;
static SemaphoreHandle_t UART_TransmitMutexHandle;
static void UART_TransmitTask(void* parameters){
uint8_t buf;
while(1){
xQueueReceive(UART_TransmitQueueHandle, &buf, portMAX_DELAY);
HAL_UART_Transmit_IT(&huart1, &buf, sizeof(uint8_t));
ulTaskNotifyTake(pdTRUE, portMAX_DELAY);
}}
void HAL_UART_TxCpltCallback(UART_HandleTypeDef *huart){
if(huart->Instance == huart1.Instance){
BaseType_t woken = pdFALSE;
vTaskNotifyGiveFromISR(UART_TransmitTaskHandle, &woken);
portYIELD_FROM_ISR(woken);
}}
// GENERAL
void UART_Init(){
UART_TransmitMutexHandle = xSemaphoreCreateMutex();
UART_TransmitQueueHandle = xQueueCreate(64, sizeof(uint8_t));
xTaskCreate(UART_TransmitTask, "transmitTask", 64, NULL, 4, &UART_TransmitTaskHandle);
}
// TRANSMIT UTIL
void UART_AsyncTransmitCharacter(char character){
xSemaphoreTake(UART_TransmitMutexHandle, portMAX_DELAY);
xQueueSendToBack(UART_TransmitQueueHandle, &character, portMAX_DELAY);
xSemaphoreGive(UART_TransmitMutexHandle);
}
void UART_AsyncTransmitString(char const* string){
if(string != NULL){
xSemaphoreTake(UART_TransmitMutexHandle, portMAX_DELAY);
for(uint32_t i = 0; i < strlen(string); i++){
xQueueSendToBack(UART_TransmitQueueHandle, string + i, portMAX_DELAY);
}
xSemaphoreGive(UART_TransmitMutexHandle);
}}
```

---

homework.c

```
#include "gpio.h"
#include "timers.h"
TimerHandle_t GPIO_Timer;
TimerHandle_t ledTimer;
unsigned passedMs = 0;
float rainfall = 0;
uint8_t rainVal;
static uint32_t tempValue;
static char tempText[4];
static char rainText[4];
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin) {
if (GPIO_Pin == GPIO_PIN_10) {
homeworkOverflow();
}}
static void homeworkTask(void *parameters){
UNUSED(parameters);
char messageTemp[9] = "Temp: ";
char messageKisa[7] = "Kisa: ";
UART_AsyncTransmitString(messageTemp);
LCD_CommandEnqueue(LCD_INSTRUCTION, LCD_SET_DD_RAM_ADDRESS_INSTRUCTION | 0x00);
for(uint32_t i = 0; i < 6; i++){
LCD_CommandEnqueue(LCD_DATA, messageKisa[i]);
}
while (1){
tempValue = TEMP_GetCurrentValue();
itoa(tempValue, tempText, 10);
UART_AsyncTransmitString(tempText);
rainVal = rainfall + 1;
itoa(rainVal, rainText, 10);
LCD_CommandEnqueue(LCD_INSTRUCTION, LCD_SET_DD_RAM_ADDRESS_INSTRUCTION | (0x00 + strlen(messageKisa)));
for(int i = 0; i < strlen(rainText); i++){
LCD_CommandEnqueue(LCD_DATA, rainText[i]);
}

vTaskDelay(pdMS_TO_TICKS(200));
for (int i=0;i<strlen(tempText);i++){
UART_AsyncTransmitCharacter('\b');
}
LCD_CommandEnqueue(LCD_INSTRUCTION, LCD_SET_DD_RAM_ADDRESS_INSTRUCTION | (0x00 + strlen(messageKisa)));
for (int i = 0; i < strlen(rainText); i++) {
LCD_CommandEnqueue(LCD_DATA, ' ');
}}}
void ledCounter(TimerHandle_t xTimer) {
UNUSED(xTimer);
if (TEMP_GetCurrentValue() < 30) {
HAL_GPIO_WritePin(GPIOB, GPIO_PIN_9, GPIO_PIN_RESET);
HAL_GPIO_WritePin(GPIOB, GPIO_PIN_8, GPIO_PIN_SET);
} else {
HAL_GPIO_WritePin(GPIOB, GPIO_PIN_8, GPIO_PIN_RESET);
HAL_GPIO_TogglePin(GPIOB, GPIO_PIN_9);
}}
void homeworkOverflow() {
rainfall = 36000 / passedMs;
passedMs = 0;
}
void timerCallback(TimerHandle_t xTimer){
UNUSED(xTimer);
passedMs++;
}
void homeworkInit(){
LCD_Init();
UART_Init();
TEMP_Init();
xTaskCreate(homeworkTask, "homeworkTask", 64, NULL, 5, NULL);
GPIO_Timer = xTimerCreate("GPIO_Timer", pdMS_TO_TICKS(1), pdTRUE, NULL, timerCallback);
xTimerStart(GPIO_Timer, portMAX_DELAY);
ledTimer = xTimerCreate("ledTimer", pdMS_TO_TICKS(500), pdTRUE,NULL, ledCounter);
xTimerStart(ledTimer, portMAX_DELAY);
}
```