```
#include "stm32f10x.h"
#include <stdint.h>
#include <stdbool.h>
#include <stdio.h>

#define MAX_ALARMS 5
#define BRIGHT_STATE 1  // When PA0 reads HIGH, it is considered "bright"

// Global time variables (Day: 0–6; Hour: 0–23; Minute, Second: 0–59)
// User modifies this snippet of code (1). Set time to real time before flashing code
uint8_t day  = 1;  // 0 = Sunday, 1 = Monday, …, 6 = Saturday
uint8_t hour = 11;
uint8_t min  = 30;
uint8_t sec  = 45;

// Tick counter (accumulate seconds from TIM2)
volatile uint32_t tick_count = 0;

// Alarm storage arrays
uint8_t alarmDay[MAX_ALARMS];    // Day of week for the alarm (0-6)
uint8_t alarmH[MAX_ALARMS];      // Hour (0-23)
uint8_t alarmM[MAX_ALARMS];      // Minute (0-59)
uint8_t alarmS[MAX_ALARMS];      // Second (0-59)
uint8_t alarmTriggered[MAX_ALARMS];
uint8_t alarmCount = 0;

// USART1 (Debug Serial) Functions
void USART1_Init(void) {
    RCC->APB2ENR |= RCC_APB2ENR_IOPAEN | RCC_APB2ENR_USART1EN;
    //PA9 for USART TX
    GPIOA->CRH &= ~(0xF << 4);
    GPIOA->CRH |=  (0xB << 4);
    USART1->BRR = 0x1D4C;
    USART1->CR1 |= USART_CR1_TE | USART_CR1_RE | USART_CR1_UE;
}

void USART1_SendChar(char c) {
    while (!(USART1->SR & USART_SR_TXE));
    USART1->DR = c;
}

void send_string(const char *s) {
    while (*s)
        USART1_SendChar(*s++);
}

void send_time(uint8_t d, uint8_t h, uint8_t m, uint8_t s) {
    char buf[25];
```

```
    sprintf(buf, "Day %d %02d:%02d:%02d\r\n", d, h, m, s);
    send_string(buf);
}

//PWM and Motor Control Functions
void PWM_Init(void) {
    RCC->APB2ENR |= RCC_APB2ENR_IOPBEN;
    RCC->APB1ENR |= RCC_APB1ENR_TIM4EN;
    // Configure PB6 for PWM output (TIM4_CH1)
    GPIOB->CRL &= ~(0xF << 24);
    GPIOB->CRL |=  (0xB << 24);
    TIM4->PSC = 72 - 1;
    TIM4->ARR = 1000 - 1;
    TIM4->CCR1 = 0;
    TIM4->CCMR1 |= (6 << 4);  // PWM mode 1
    TIM4->CCMR1 |= (1 << 3);  // Preload enable
    TIM4->CCER |= (1 << 0);   // Enable channel 1
    TIM4->CR1 |= (1 << 7);    // Auto-reload preload enable
    TIM4->CR1 |= (1 << 0);    // Start timer
}

void pause(uint32_t ms) {
    for (uint32_t i = 0; i < ms; i++) {
        for (uint32_t j = 0; j < 8000; j++) {
            __asm volatile("");
        }
    }
}

void run_motor(uint16_t duty_percent, uint32_t duration_ms) {
    TIM4->CCR1 = (TIM4->ARR + 1) * duty_percent / 100;
    // Blocking motor run: toggle LED indicators on PA1 and PA2 for visual feedback
    for (uint32_t i = 0; i < duration_ms / 200; i++) {
        GPIOA->ODR ^= (1 << 1);  // Toggle LED1 on PA1
        pause(100);
    }
    GPIOA->ODR &= ~(1 << 1);  // Turn off LEDs
    TIM4->CCR1 = 0;  // Stop motor
}

// Alarms
void setAlarm(uint8_t alarm_day, uint8_t h, uint8_t m, uint8_t s) {
    if (alarmCount < MAX_ALARMS) {
        alarmDay[alarmCount] = alarm_day;
        alarmH[alarmCount] = h;
        alarmM[alarmCount] = m;
        alarmS[alarmCount] = s;
        alarmTriggered[alarmCount] = 0;
```

```
        alarmCount++;
    }
}

// TIM2 (1Hz Timer for Clock/Alarms)
void TIM2_IRQHandler(void) {
    if (TIM2->SR & TIM_SR_UIF) {
        TIM2->SR &= ~TIM_SR_UIF;
        tick_count++;  // Increment by 1 each second
    }
}

void TIM2_Init(void) {
    RCC->APB1ENR |= RCC_APB1ENR_TIM2EN;
    TIM2->PSC = 7200 - 1;   // 72MHz/7200 = 10kHz
    TIM2->ARR = 10000 - 1;  // 10kHz/10000 = 1Hz
    TIM2->DIER |= TIM_DIER_UIE;
    TIM2->CR1  |= TIM_CR1_CEN;
    NVIC_EnableIRQ(TIM2_IRQn);
}

// GPIO Initialization
// Initializes the manual feed button and LED indicators used for motor feedback.
// PA1 and PA2 are configured as LED outputs; PC13 is the manual feed button.
void init_leds_and_button(void) {
    RCC->APB2ENR |= RCC_APB2ENR_IOPAEN | RCC_APB2ENR_IOPCEN;
    // Configure PA1 and PA2 as outputs
    GPIOA->CRL &= ~((0xF << (1 * 4)) | (0xF << (2 * 4)));
    GPIOA->CRL |= ((0x1 << (1 * 4)) | (0x1 << (2 * 4)));
    // Configure PC13 as input for the manual feed button
    GPIOC->CRH &= ~(0xF << 20);
    GPIOC->CRH |= (0x4 << 20);
    GPIOC->ODR |= (1 << 13);
}

// LDR (Digital) Initialization on PA0
// The LDR is connected directly between PA0 and GND. Internal pull-up on PA0 keeps it
HIGH normally.
void LDR_Digital_Init(void) {
    RCC->APB2ENR |= RCC_APB2ENR_IOPAEN;
    // Configure PA0 as a digital input with internal pull-up.
    GPIOA->CRL &= ~(0xF << (0 * 4));
    GPIOA->CRL |= (0x8 << (0 * 4));  // Input with pull-up/down
    GPIOA->ODR |= (1 << 0);          // Enable internal pull-up so PA0 reads HIGH by default.
}

//  Main Program
int main(void) {
```

```
    USART1_Init();
    PWM_Init();
    TIM2_Init();
    init_leds_and_button();
    LDR_Digital_Init();

    send_string("Motor Timer with Digital LDR Trigger\r\n");
    send_string("Scheduled Feeding Times (Format: Day HH:MM:SS)\r\n");

    // Set scheduled feed times
            // User modifies this snippet of code (2).
    setAlarm(1, 14, 30, 40);
    setAlarm(1, 11, 30, 50);
    setAlarm(1, 15, 31, 10);

    while (1) {
        // Manual feed: if the pushbutton on PC13 is pressed (active low)
        if (!(GPIOC->IDR & (1 << 13))) {
            send_string(">> MANUAL Food Dispense <<\r\n");
            run_motor(50, 10000);
            pause(300);
        }

        // Output for each tick
        while (tick_count > 0) {
            tick_count--;  // Process one tick (1 second)
            send_time(day, hour, min, sec);

            // Simultaneously check all alarms(regardless of order) against the current tick
            for (int i = 0; i < alarmCount; i++) {
                if (!alarmTriggered[i] &&
                    day == alarmDay[i] &&
                    hour == alarmH[i] &&
                    min == alarmM[i] &&
                    sec == alarmS[i])
                {
                    // In the morning (before 12:00 PM), LDR checks lighting condition.
                    if (hour < 12) {
                        uint8_t ldr_state = (GPIOA->IDR & (1 << 0)) ? 1 : 0;
                        if (ldr_state == BRIGHT_STATE) {
                            send_string(">> MOTOR START (Morning - Tank light on before 12pm)
<<\r\n");
                            run_motor(50, 10000);
                            send_string(">> MOTOR STOP <<\r\n");
                            alarmTriggered[i] = 1;
                        } else {
                            send_string(">> ALARM SKIPPED (Morning - tank light is not on) <<\r\n");
                        }
```

```
                } else {
                    // In the afternoon/evening, LDR reading isn't read.
                    send_string(">> MOTOR START (Afternoon/Evening) <<\r\n");
                    run_motor(50, 10000);
                    send_string(">> MOTOR STOP <<\r\n");
                    alarmTriggered[i] = 1;
                }
            }
        }
        // Increment time by one second
        sec++;
        if (sec >= 60) {
            sec = 0;
            min++;
            if (min >= 60) {
                min = 0;
                hour++;
                                                                    //Hour loops back to 0
after 23:59:59
                if (hour >= 24) {
                    hour = 0;
                    // When the day resets, it increments the day counter (goes back to 0 after 6)
                    day++;
                    if (day >= 7) {
                        day = 0;
                    }
                }
            }
        }
    }
}
```