

Código para display y DMA



```

✱
tengo el siguiente reto , dime que cosas debo borrar de mi codigo para que se adapte a lo que pide
. el display debe quedar , como el dma , el led rgb y la comunicació usart : y dime que me falta para
cumplir : /* USER CODE BEGIN Header */
/**
 * *****
 * @file      : main.c
 * @brief     : Main program body
 * *****
 * @attention
 *
 * Copyright (c) 2025 STMicroelectronics.
 * All rights reserved.
 *
 * This software is licensed under terms that can be found in the LICENSE file
 * in the root directory of this software component.
 * If no LICENSE file comes with this software, it is provided AS-IS.
 *
 * *****
 */
/* USER CODE END Header */
/* Includes -----*/
#include "main.h"

/* Private includes -----*/
/* USER CODE BEGIN Includes */
#include "fsm.h"
#include "stdint.h"
#include "stdio.h"
#include "string.h"
#include <stdlib.h>
#include "arm_math.h"
/* USER CODE END Includes */

/* Private typedef -----*/
/* USER CODE BEGIN PTD */

/* USER CODE END PTD */

/* Private define -----*/
/* USER CODE BEGIN PD */

/* USER CODE END PD */

```

```

/* Private macro -----*/
/* USER CODE BEGIN PM */

/* USER CODE END PM */

/* Private variables -----*/
ADC_HandleTypeDef hadc1;
DMA_HandleTypeDef hdma_adc1;

TIM_HandleTypeDef htim2;
TIM_HandleTypeDef htim3;
TIM_HandleTypeDef htim4;
DMA_HandleTypeDef hdma_tim3_ch2;

UART_HandleTypeDef huart2;

/* USER CODE BEGIN PV */
#define FFT_SIZE_MAX 2048
const uint32_t timer_clk = 84000000UL;
static uint16_t adc_buffer[FFT_SIZE_MAX];
static uint16_t fft_size = 1024;
static uint16_t display_value = 0;
static uint8_t nextDigit_FSM = 1;
static uint8_t tx_buffer[256] = {0};
#define FREQ_BUFFER_SIZE 16
float freq_buffer[FREQ_BUFFER_SIZE] = {0};
#define NUM_MUESTRAS 100
uint16_t adc_buffer[2 * NUM_MUESTRAS];
uint16_t vrx_buffer[NUM_MUESTRAS];
uint16_t vry_buffer[NUM_MUESTRAS];
uint8_t freq_index = 0;
uint8_t freq_full = 0;
char rx_buffer[128] = {0};
uint8_t rx_char = 0;
uint8_t rx_index=0;
volatile e_PosiblesEvents pending_event = IDLE;
volatile uint8_t data_snapshot = 0;

volatile uint8_t flagCapture = 0;
volatile uint32_t firstCapture = 0;
volatile uint32_t secondCapture = 0;
volatile uint32_t elapsedTicks = 0;
volatile uint32_t counterOverflow = 0;
/* USER CODE END PV */

/* Private function prototypes -----*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_DMA_Init(void);
static void MX_USART2_UART_Init(void);
static void MX_TIM2_Init(void);

```

```

static void MX_TIM4_Init(void);
static void MX_ADC1_Init(void);
static void MX_TIM3_Init(void);
/* USER CODE BEGIN PFP */
void displayNumber(uint8_t digitValue);
void HandleLEDDelayCmd(const char *arg);
void HandleSampleFreqCmd(const char *arg);
void HandleRGBCmd(const char *arg);
void HandlePWMFreqCmd(const char *arg);
void HandleUnknownCmd(void);
void HandleFFTSIZECmd(const char *arg);
void HandleStatusCmd(void);
void HandlePrintADC(void);
void HandleFreqDisplayCmd(void);
void HandleClearCmd(void);
/* USER CODE END PFP */

/* Private user code -----*/
/* USER CODE BEGIN 0 */

e_PosiblesStates state_machine_action(e_PosiblesEvents event) {
    switch (event) {
        case IDLE:
            break;
        case EVENT_ENCODER:
            if (data_snapshot) // DATA snapshot
                display_value = (display_value == 0) ? 4095 : display_value - 1;
            else
                display_value = (display_value == 4095) ? 0 : display_value + 1;
            break;

        case EVENT_SW:
            display_value = 0;
            nextDigit_FSM = 1;
            break;

        case EVENT_TIMER_TICK:{
            // Apagar todos los dígitos
            HAL_GPIO_WritePin(GPIOA, GPIO_PIN_0, GPIO_PIN_RESET); // userDis1
            HAL_GPIO_WritePin(GPIOA, GPIO_PIN_1, GPIO_PIN_RESET); // userDis2
            HAL_GPIO_WritePin(GPIOB, GPIO_PIN_0, GPIO_PIN_RESET); // userDis3
            HAL_GPIO_WritePin(GPIOA, GPIO_PIN_4, GPIO_PIN_RESET); // userDis4

            switch (nextDigit_FSM) {
                case 1:
                    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_0, GPIO_PIN_SET);
                    displayNumber(display_value % 10);
                    break;
                case 2:
                    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_1, GPIO_PIN_SET);
                    displayNumber((display_value / 10) % 10);
            }
        }
    }
}

```

```

        break;
    case 3:
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_0, GPIO_PIN_SET);
        displayNumber((display_value / 100) % 10);
        break;
    case 4:
        HAL_GPIO_WritePin(GPIOA, GPIO_PIN_4, GPIO_PIN_SET);
        displayNumber((display_value / 1000) % 10);
        break;
}
nextDigit_FSM = (nextDigit_FSM < 4) ? nextDigit_FSM + 1 : 1;
break;}
case EVENT_USART: {
    // eco
    int len = snprintf((char*)tx_buffer, sizeof(tx_buffer),
        "Comando recibido: %s\r\n", rx_buffer);
    HAL_UART_Transmit(&huart2, tx_buffer, len, 1000);

    // dispatch
    if (strncmp((char*)rx_buffer,"led=",4)==0)  HandleLEDDelayCmd(rx_buffer+4);
    else if (strncmp((char*)rx_buffer,"fmuestreo=",10)==0)
        HandleSampleFreqCmd(rx_buffer+10);
    else if (strncmp((char*)rx_buffer,"rgb=",4)==0)  HandleRGBCmd(rx_buffer+4);
    else if (strncmp((char*)rx_buffer,"fftSize=",8)==0) HandleFFTSizeCmd(rx_buffer+8);
    else if (strncmp((char*)rx_buffer,"status",6)==0) HandleStatusCmd();
    else if (strncmp((char*)rx_buffer,"print",5)==0) HandlePrintADC();
    else if (strncmp((char*)rx_buffer,"freq",4)==0) HandleFreqDisplayCmd();
    else if (strncmp((char*)rx_buffer,"fft",3)==0) HandlePrintFFT();
    else if (strncmp((char*)rx_buffer,"info",4)==0) HandleFFTInfo();
    else if (strncmp((char*)rx_buffer,"help",4)==0) HandleHelpCmd();
    else if (strncmp((char*)rx_buffer, "clear", 5) == 0) HandleClearCmd();
    else
        ;

    memset(rx_buffer,0,sizeof(rx_buffer));
    break;
}
case EVENT_IC_CAPTURE:{
    float periodo_ms = elapsedTicks * (1.0f / 84000000.0f) * 1000.0f; // En ms
    float frecuencia_hz = 84000000.0f / elapsedTicks;
    // Ejemplo: enviar por serial
    freq_buffer[freq_index++] = frecuencia_hz;
    if (freq_index >= FREQ_BUFFER_SIZE) {
        freq_index = 0;
        freq_full = 1;
    }
    break;
}
return event;
}

void displayNumber(uint8_t digitValue) {
    uint32_t start = DWT->CYCCNT;
    static uint8_t contador_muestras = 0;

```

```
// Apagar todos los segmentos
HAL_GPIO_WritePin(GPIOA, GPIO_PIN_10, GPIO_PIN_RESET); // A
HAL_GPIO_WritePin(GPIOB, GPIO_PIN_10, GPIO_PIN_RESET); // B
HAL_GPIO_WritePin(GPIOB, GPIO_PIN_13, GPIO_PIN_RESET); // C
HAL_GPIO_WritePin(GPIOC, GPIO_PIN_0, GPIO_PIN_RESET); // D
HAL_GPIO_WritePin(GPIOC, GPIO_PIN_1, GPIO_PIN_RESET); // E
HAL_GPIO_WritePin(GPIOB, GPIO_PIN_1, GPIO_PIN_RESET); // F
HAL_GPIO_WritePin(GPIOB, GPIO_PIN_5, GPIO_PIN_RESET); // G

switch (digitValue) {
    case 0:
        HAL_GPIO_WritePin(GPIOA, GPIO_PIN_10, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_10, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_13, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOC, GPIO_PIN_0, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOC, GPIO_PIN_1, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_1, GPIO_PIN_SET);
        break;
    case 1:
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_10, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_13, GPIO_PIN_SET);
        break;
    case 2:
        HAL_GPIO_WritePin(GPIOA, GPIO_PIN_10, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_10, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOC, GPIO_PIN_0, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOC, GPIO_PIN_1, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_5, GPIO_PIN_SET);
        break;
    case 3:
        HAL_GPIO_WritePin(GPIOA, GPIO_PIN_10, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_10, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_13, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOC, GPIO_PIN_0, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_5, GPIO_PIN_SET);
        break;
    case 4:
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_10, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_13, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_1, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_5, GPIO_PIN_SET);
        break;
    case 5:
        HAL_GPIO_WritePin(GPIOA, GPIO_PIN_10, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_13, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOC, GPIO_PIN_0, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_1, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_5, GPIO_PIN_SET);
        break;
    case 6:
        HAL_GPIO_WritePin(GPIOA, GPIO_PIN_10, GPIO_PIN_SET);
```

```

    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_13, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOC, GPIO_PIN_0, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOC, GPIO_PIN_1, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_1, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_5, GPIO_PIN_SET);
    break;
case 7:
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_10, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_10, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_13, GPIO_PIN_SET);
    break;
case 8:
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_10, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_10, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_13, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOC, GPIO_PIN_0, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOC, GPIO_PIN_1, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_1, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_5, GPIO_PIN_SET);
    break;
case 9:
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_10, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_10, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_13, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_1, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_5, GPIO_PIN_SET);
    break;
}
uint32_t elapsed = DWT->CYCCNT - start;

if (contador_muestras < 10) {
    printf("Ciclos para mostrar digito %d: %lu\n", digitValue, (unsigned long)elapsed);
    contador_muestras++;
}
}

void HandleLEDDelayCmd(const char *arg) {
    uint32_t nuevo = atoi(arg);
    // Ajusta TIM2 para el blinky
    __HAL_TIM_SET_AUTORELOAD(&htim2, nuevo);
    __HAL_TIM_SET_COUNTER(&htim2, 0);
    // Feedback
    int len = snprintf((char*)tx_buffer, sizeof(tx_buffer),
        "LED delay = %lu ms\r\n", (unsigned long)nuevo);
    HAL_UART_Transmit(&huart2, tx_buffer, len, 1000);
}

void HandleSampleFreqCmd(const char *arg) {
    uint32_t fs;
    char option = arg[0];
    // Verifica la opción seleccionada (1, 2, 3, 4)
    switch(option) {

```

```

        case '1': fs = 44100; break;
        case '2': fs = 48000; break;
        case '3': fs = 96000; break;
        case '4': fs = 128000; break;
        default: {
            const char *msg = "Opciones válidas para 'fmuestreo=' son: 1, 2, 3, 4\r\n";
            HAL_UART_Transmit(&huart2, (uint8_t*)msg, strlen(msg), 1000);
            return;
        }
    }
}

// calcular ARR y PSC
uint32_t arr = timer_clk / fs - 1;
uint32_t psc = 0;
if (arr > 0xFFFF) {
    // escalar PSC si supera 16 bits
    psc = (arr / 0x10000) + 1;
    arr = (timer_clk / (psc+1) / fs) - 1;
}
// aplicar a TIM3 y reiniciar
__HAL_TIM_SET_PRESCALER(&htim3, psc);
__HAL_TIM_SET_AUTORELOAD(&htim3, arr);
__HAL_TIM_SET_COUNTER(&htim3, 0);
HAL_TIM_Base_Stop(&htim3);
HAL_TIM_Base_Start(&htim3);
// Feedback
int len = snprintf((char*)tx_buffer, sizeof(tx_buffer),
    "Sample TIM3 @ %lu Hz (PSC=%lu, ARR=%lu)\r\n",
    (unsigned long)fs, (unsigned long)psc, (unsigned long)arr);
HAL_UART_Transmit(&huart2, tx_buffer, len, 1000);
}

void HandleRGBCmd(const char *arg) {
    // R, G, B si aparecen en arg
    GPIO_PinState R = (strchr(arg, 'R') ? GPIO_PIN_SET : GPIO_PIN_RESET);
    GPIO_PinState G = (strchr(arg, 'G') ? GPIO_PIN_SET : GPIO_PIN_RESET);
    GPIO_PinState B = (strchr(arg, 'B') ? GPIO_PIN_SET : GPIO_PIN_RESET);
    HAL_GPIO_WritePin(GPIOC, GPIO_PIN_3, R);
    HAL_GPIO_WritePin(GPIOC, GPIO_PIN_2, G);
    HAL_GPIO_WritePin(GPIOC, GPIO_PIN_13, B);
    // Feedback
    int len = snprintf((char*)tx_buffer, sizeof(tx_buffer),
        "RGB -> R:%c G:%c B:%c\r\n",
        R==GPIO_PIN_SET?'1':'0',
        G==GPIO_PIN_SET?'1':'0',
        B==GPIO_PIN_SET?'1':'0');
    HAL_UART_Transmit(&huart2, tx_buffer, len, 1000);
}

void HandleFFTSIZECmd(const char *arg) {
    char option = arg[0];

```

```

if (option == '1') {
    fft_size = 1024;
} else if (option == '2') {
    fft_size = 2048;
} else {
    const char *msg = "Opciones válidas para 'fftSize=' son:\r\n"
        "1 -> 1024 puntos\r\n"
        "2 -> 2048 puntos\r\n";
    HAL_UART_Transmit(&huart2, (uint8_t*)msg, strlen(msg), 1000);
    return;
}

// reiniciar DMA con nuevo tamaño
HAL_ADC_Stop_DMA(&hadc1);
HAL_ADC_Start_DMA(&hadc1, (uint32_t*)adc_buffer, fft_size);

int len = snprintf((char*)tx_buffer, sizeof(tx_buffer),
    "FFT size set to %u\r\n", fft_size );
HAL_UART_Transmit(&huart2, tx_buffer, len, 1000);
}

void HandleStatusCmd(void) {
    uint32_t psc = htim3.Init.Prescaler;
    uint32_t arr = htim3.Init.Period;
    float fs = 84000000.0f / ((psc + 1) * (arr + 1));
    float bin_res = fs / fft_size;

    int len = snprintf((char*)tx_buffer, sizeof(tx_buffer),
        "Config:\r\nSample TIM3 @ %.2f Hz (PSC=%lu, ARR=%lu)\r\nFFT size: %u\r\nResolucion
espectral: %.2f Hz/bin\r\nCanal ADC: 6\r\nTrigger ADC: TIM3_TRGO\r\n",
        fs, (unsigned long)psc, (unsigned long)arr, fft_size, bin_res);
    HAL_UART_Transmit(&huart2, tx_buffer, len, 1000);
}

void HandlePrintADC(void) {
    static uint8_t continuous_mode = 0;
    char msg[32];

    // Toggle del modo continuo
    if (continuous_mode) {
        continuous_mode = 0;
        return;
    }

    continuous_mode = 1;

    while (continuous_mode) {
        for (int i = 0; i < fft_size; i++) {
            // Enviar valor ADC normalizado (0.0-3.3V) como float
            float voltage = adc_buffer[i] * (3.3f / 4095.0f);
            int len = snprintf(msg, sizeof(msg), "%.4f\r\n", voltage);

            if (HAL_UART_Transmit(&huart2, (uint8_t*)msg, len, 10) != HAL_OK) {

```



```

        continuous_mode = 0;
        break;
    }

    // Verificar comando de parada sin bloquear
    if (rx_index > 0 && strstr(rx_buffer, "print") != NULL) {
        continuous_mode = 0;
        memset(rx_buffer, 0, sizeof(rx_buffer));
        rx_index = 0;
        break;
    }
}
}
}

void HandleFreqDisplayCmd(void) {
    char msg[64];
    float suma = 0.0f;
    int count = freq_full ? FREQ_BUFFER_SIZE : freq_index;

    HAL_UART_Transmit(&huart2, (uint8_t*)"Frecuencias IC (Hz):\r\n", 24, 100);

    for (int i = 0; i < count; i++) {
        int idx = (freq_index + i) % FREQ_BUFFER_SIZE;
        suma += freq_buffer[idx];
        int len = snprintf(msg, sizeof(msg), "%.2f\r\n", freq_buffer[idx]);
        HAL_UART_Transmit(&huart2, (uint8_t*)msg, len, 100);
    }

    float promedio = (count > 0) ? suma / count : 0.0f;
    int len = snprintf(msg, sizeof(msg), "Promedio: %.2f Hz\r\n", promedio);
    HAL_UART_Transmit(&huart2, (uint8_t*)msg, len, 100);
}

void HandlePrintFFT(void) {
    float input_f32[FFT_SIZE_MAX];
    float output_fft[FFT_SIZE_MAX];

    // Convertir y normalizar datos ADC
    for (int i = 0; i < fft_size; i++) {
        input_f32[i] = (float)adc_buffer[i] - 2048.0f; // Eliminar offset DC
    }

    // Configurar y calcular FFT
    arm_rfft_fast_instance_f32 S;
    arm_rfft_fast_init_f32(&S, fft_size);
    arm_rfft_fast_f32(&S, input_f32, output_fft, 0);

    // Calcular parámetros de frecuencia
    float fs = 84000000.0f / ((htim3.Init.Prescaler + 1) * (htim3.Init.Period + 1));
    float bin_res = fs / fft_size;

```

```
// Cabecera simple para CoolTerm
HAL_UART_Transmit(&huart2, (uint8_t*)"FFT_DATA_START\n", 15, 100);

// Enviar magnitudes positivas
for (int i = 1; i < fft_size / 2; i++) {
    float real = output_fft[2 * i];
    float imag = output_fft[2 * i + 1];
    float mag = sqrtf(real * real + imag * imag) / (fft_size/2);

    // Convertir a valor positivo absoluto (para histograma)
    float positive_mag = fabsf(mag);

    char msg[32];
    // Formato: "frecuencia,magnitud\n" (sin texto adicional)
    int len = snprintf(msg, sizeof(msg), "%.1f,%.4f\n", i * bin_res, positive_mag);
    HAL_UART_Transmit(&huart2, (uint8_t*)msg, len, 100);
}

// Finalización
HAL_UART_Transmit(&huart2, (uint8_t*)"FFT_DATA_END\n", 13, 100);
}

void HandleFFTInfo(void) {
    float input_f32[FFT_SIZE_MAX];
    float output_fft[FFT_SIZE_MAX];
    float mag_fft[FFT_SIZE_MAX / 2];

    for (int i = 0; i < fft_size; i++) {
        input_f32[i] = (float)adc_buffer[i] - 2048.0f;
    }

    arm_rfft_fast_instance_f32 S;
    arm_rfft_fast_init_f32(&S, fft_size);
    arm_rfft_fast_f32(&S, input_f32, output_fft, 0);

    for (int i = 0; i < fft_size / 2; i++) {
        float real = output_fft[2 * i];
        float imag = output_fft[2 * i + 1];
        mag_fft[i] = sqrtf(real * real + imag * imag);
    }

    uint32_t max_index = 0;
    float max_val = 0.0f;
    arm_max_f32(&mag_fft[1], (fft_size / 2) - 1, &max_val, &max_index);
    max_index += 1; // porque empezamos en bin 1

    float fs = 84000000.0f / ((htim3.Init.Prescaler + 1) * (htim3.Init.Period + 1));
    float freq_bin = fs / fft_size;
    float freq_detected = freq_bin * max_index;
}
```

```

float sum = 0.0f, sum_sq = 0.0f;
for (int i = 0; i < fft_size; i++) {
    float x = (float)adc_buffer[i];
    sum += x;
}
float offset = sum / fft_size;
for (int i = 0; i < fft_size; i++) {
    float x = (float)adc_buffer[i] - offset;
    sum_sq += x * x;
}
float rms = sqrtf(sum_sq / fft_size);

float db_val = 20.0f * log10f(max_val + 1e-6f);

char msg[160];
int len = snprintf(msg, sizeof(msg),
    "Info de FFT:\r\nFrecuencia dominante: %.2f Hz\r\nMagnitud: %.2f dB\r\nOffset: %.2f niveles
ADC\r\nRMS: %.2f niveles ADC\r\n",
    freq_detected, db_val, offset, rms);
HAL_UART_Transmit(&huart2, (uint8_t*)msg, len, 100);
}

void HandleClearCmd(void) {
    const char *clear_screen = "\033[2J\033[H"; // Secuencia ANSI para limpiar la pantalla
    HAL_UART_Transmit(&huart2, (uint8_t*)clear_screen, strlen(clear_screen), 1000);
}

void HandleHelpCmd(void) {
    const char *help_msg =
        "\r\n===== AYUDA =====\r\n"
        "\r\nComandos disponibles:\r\n"
        " led=<ms>@    - Cambia la frecuencia del LED Blinky\r\n"
        " fmuestreo=<1|2|3|4>@ - Frecuencia de muestreo del ADC:\r\n"
        "     - 1 -> 44100 Hz\r\n"
        "     - 2 -> 48000 Hz\r\n"
        "     - 3 -> 96000 Hz\r\n"
        "     - 4 -> 128000 Hz\r\n"
        " rgb=<RGB>@    - Control de LED RGB, ej: rgb=RG\r\n"
        " fftSize=<size>@ - Tamaño de FFT: \r\n"
        "     - 1 -> 1024 \r\n"
        "     - 2 -> 2048 \r\n"
        " status@      - Mostrar configuración actual\r\n"
        " print@       - Imprimir datos crudos del ADC\r\n"
        " fft@         - Imprimir espectro FFT\r\n"
        " info@        - Frecuencia dominante, magnitud y offset\r\n"
        " freq@        - Historial de frecuencia medida por IC\r\n"
        " help @       - Mostrar esta ayuda\r\n"
        "===== \r\n";
    HAL_UART_Transmit(&huart2, (uint8_t*)help_msg, strlen(help_msg), 1000);
}

void imprimirBinario(uint8_t valor) {
    for (int i = 7; i >= 0; i--) {
        printf("%d", (valor >> i) & 1);
    }
}

```

```

    }
}

void mostrarOperacion(const char *desc, uint8_t resultado) {
    printf("%s\n", desc);
    printf("Binario: "); imprimirBinario(resultado);
    printf("\nHex: 0x%02X\n", resultado);
    printf("ASCII: %c\n\n", (resultado >= 32 && resultado <= 126) ? resultado : '.');
}

void verificarOperacionesCrucigrama(void) {
    // 1. Caracter ASCII 'B'
    uint8_t op1 = 'B';
    mostrarOperacion("1. ASCII 'B'", op1);

    // 2. ((!!0xC) << 0b11) | (~0x45)
    uint8_t op2 = ((!!0xC) << 0b11) | (~0x45);
    mostrarOperacion("2. ((!!0xC) << 0b11) | (~0x45)", op2);

    // 3. (0b00111100 & 0xAC) << 0x3
    uint8_t op3 = (0b00111100 & 0xAC) << 0x3;
    mostrarOperacion("3. (0b00111100 & 0xAC) << 0x3)", op3);

    // 4. -121
    uint8_t op4 = (uint8_t)(-121);
    mostrarOperacion("4. -121 (uint8_t)", op4);

    // 5. (0xCAFE & 0xFF0) >> 7
    uint16_t val5 = (0xCAFE & 0xFF0) >> 7;
    uint8_t op5 = (uint8_t)(val5 & 0xFF);
    mostrarOperacion("5. (0xCAFE & 0xFF0) >> 7", op5);

    // 6. (1 << 4) + (0x3 << 4) - 10 + (0b11 << 0x6)
    uint8_t op6 = (1 << 4) + (0x3 << 4) - 10 + (0b11 << 6);
    mostrarOperacion("6. (1<<4)+(0x3<<4)-10+(0b11<<6)", op6);

    // 7. ('T' | '5')
    uint8_t op7 = 'T' | '5';
    mostrarOperacion("7. ('T' | '5')", op7);

    // 8. (2921 & 0xD6)
    uint8_t op8 = 2921 & 0xD6;
    mostrarOperacion("8. (2921 & 0xD6)", op8);

    // 9. 01100110 ^ 00101011
    uint8_t a9 = 0b01100110;
    uint8_t b9 = 0b00101011;
    uint8_t op9 = a9 ^ b9;
    mostrarOperacion("9. 01100110 ^ 00101011", op9);
}

/* USER CODE END 0 */

```

```

/**
 * @brief The application entry point.
 * @retval int
 */
int main(void)
{

    /* USER CODE BEGIN 1 */

    /* USER CODE END 1 */

    /* MCU Configuration-----*/

    /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
    HAL_Init();

    /* USER CODE BEGIN Init */
    CoreDebug->DEMCR |= CoreDebug_DEMCR_TRCENA_Msk; // Habilitar DWT
    DWT->CYCCNT = 0; // Reiniciar contador
    DWT->CTRL |= DWT_CTRL_CYCCNTENA_Msk; // Encender contador de ciclos
    /* USER CODE END Init */

    /* Configure the system clock */
    SystemClock_Config();

    /* USER CODE BEGIN SysInit */

    /* USER CODE END SysInit */

    /* Initialize all configured peripherals */
    MX_GPIO_Init();
    MX_DMA_Init();
    MX_USART2_UART_Init();
    MX_TIM2_Init();
    MX_TIM4_Init();
    MX_ADC1_Init();
    MX_TIM3_Init();
    /* USER CODE BEGIN 2 */
    HAL_TIM_Base_Start_IT(&htim2);
    HAL_TIM_Base_Start_IT(&htim3);
    HAL_TIM_Base_Start_IT(&htim4);
    HAL_UART_Receive_IT(&huart2,&rx_char,1);
    HAL_ADC_Start_DMA(&hadc1, (uint32_t*)adc_buffer, fft_size);
    HAL_TIM_IC_Start_IT(&htim3, TIM_CHANNEL_2);
    HandleSampleFreqCmd(3); //96000 hz muestreo
    HandleFFTSIZECmd(1); // 1-> 1024

    verificarOperacionesCrucigrama();
    medirTiempoEnvioUART();
    HandleHelpCmd();

```

```

/* USER CODE END 2 */

/* Infinite loop */
/* USER CODE BEGIN WHILE */
while (1)
{
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
    if (pending_event != IDLE){
        state_machine_action(pending_event);
        pending_event = IDLE;
    }
}
/* USER CODE END 3 */
}

/**
 * @brief System Clock Configuration
 * @retval None
 */
void SystemClock_Config(void)
{
    RCC_OscInitTypeDef RCC_OscInitStruct = {0};
    RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};

    /** Configure the main internal regulator output voltage
    */
    __HAL_RCC_PWR_CLK_ENABLE();
    __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE1);

    /** Initializes the RCC Oscillators according to the specified parameters
    * in the RCC_OscInitTypeDef structure.
    */
    RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI;
    RCC_OscInitStruct.HSISState = RCC_HSI_ON;
    RCC_OscInitStruct.HSICalibrationValue = RCC_HSICALIBRATION_DEFAULT;
    RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
    RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSI;
    RCC_OscInitStruct.PLL.PLLM = 16;
    RCC_OscInitStruct.PLL.PLLN = 336;
    RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV4;
    RCC_OscInitStruct.PLL.PLLQ = 4;
    if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
    {
        Error_Handler();
    }

    /** Initializes the CPU, AHB and APB buses clocks
    */

```

```

RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
                             |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV2;
RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;

if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_2) != HAL_OK)
{
    Error_Handler();
}
}

/**
 * @brief ADC1 Initialization Function
 * @param None
 * @retval None
 */
static void MX_ADC1_Init(void)
{

    /* USER CODE BEGIN ADC1_Init 0 */

    /* USER CODE END ADC1_Init 0 */

    ADC_ChannelConfTypeDef sConfig = {0};

    /* USER CODE BEGIN ADC1_Init 1 */

    /* USER CODE END ADC1_Init 1 */

    /** Configure the global features of the ADC (Clock, Resolution, Data Alignment and number of
    conversion)
    */
    hadc1.Instance = ADC1;
    hadc1.Init.ClockPrescaler = ADC_CLOCK_SYNC_PCLK_DIV4;
    hadc1.Init.Resolution = ADC_RESOLUTION_12B;
    hadc1.Init.ScanConvMode = ENABLE;
    hadc1.Init.ContinuousConvMode = DISABLE;
    hadc1.Init.DiscontinuousConvMode = DISABLE;
    hadc1.Init.ExternalTrigConvEdge = ADC_EXTERNALTRIGCONVEDGE_RISING;
    hadc1.Init.ExternalTrigConv = ADC_EXTERNALTRIGCONV_T3_TRGO;
    hadc1.Init.DataAlign = ADC_DATAALIGN_RIGHT;
    hadc1.Init.NbrOfConversion = 2;
    hadc1.Init.DMAContinuousRequests = ENABLE;
    hadc1.Init.EOCSelection = ADC_EOC_SINGLE_CONV;
    if (HAL_ADC_Init(&hadc1) != HAL_OK)
    {
        Error_Handler();
    }
}

```

```

/** Configure for the selected ADC regular channel its corresponding rank in the sequencer and its
sample time.
*/
sConfig.Channel = ADC_CHANNEL_6;
sConfig.Rank = 1;
sConfig.SamplingTime = ADC_SAMPLETIME_3CYCLES;
if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
{
    Error_Handler();
}

/** Configure for the selected ADC regular channel its corresponding rank in the sequencer and its
sample time.
*/
sConfig.Rank = 2;
if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
{
    Error_Handler();
}
/* USER CODE BEGIN ADC1_Init 2 */

/* USER CODE END ADC1_Init 2 */

}

/**
 * @brief TIM2 Initialization Function
 * @param None
 * @retval None
 */
static void MX_TIM2_Init(void)
{
    /* USER CODE BEGIN TIM2_Init 0 */

    /* USER CODE END TIM2_Init 0 */

    TIM_ClockConfigTypeDef sClockSourceConfig = {0};
    TIM_MasterConfigTypeDef sMasterConfig = {0};

    /* USER CODE BEGIN TIM2_Init 1 */

    /* USER CODE END TIM2_Init 1 */
    htim2.Instance = TIM2;
    htim2.Init.Prescaler = 42000-1;
    htim2.Init.CounterMode = TIM_COUNTERMODE_UP;
    htim2.Init.Period = 500;
    htim2.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
    htim2.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_ENABLE;
    if (HAL_TIM_Base_Init(&htim2) != HAL_OK)
    {

```



```

    Error_Handler();
}
sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
if (HAL_TIM_ConfigClockSource(&htim2, &sClockSourceConfig) != HAL_OK)
{
    Error_Handler();
}
sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
if (HAL_TIMEx_MasterConfigSynchronization(&htim2, &sMasterConfig) != HAL_OK)
{
    Error_Handler();
}
/* USER CODE BEGIN TIM2_Init 2 */

/* USER CODE END TIM2_Init 2 */

}

/**
 * @brief TIM3 Initialization Function
 * @param None
 * @retval None
 */
static void MX_TIM3_Init(void)
{
    /* USER CODE BEGIN TIM3_Init 0 */

    /* USER CODE END TIM3_Init 0 */

    TIM_ClockConfigTypeDef sClockSourceConfig = {0};
    TIM_MasterConfigTypeDef sMasterConfig = {0};
    TIM_IC_InitTypeDef sConfigIC = {0};

    /* USER CODE BEGIN TIM3_Init 1 */

    /* USER CODE END TIM3_Init 1 */
    htim3.Instance = TIM3;
    htim3.Init.Prescaler = 0;
    htim3.Init.CounterMode = TIM_COUNTERMODE_UP;
    htim3.Init.Period = 65535;
    htim3.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
    htim3.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
    if (HAL_TIM_Base_Init(&htim3) != HAL_OK)
    {
        Error_Handler();
    }
    sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
    if (HAL_TIM_ConfigClockSource(&htim3, &sClockSourceConfig) != HAL_OK)
    {

```

```

    Error_Handler();
}
if (HAL_TIM_IC_Init(&htim3) != HAL_OK)
{
    Error_Handler();
}
sMasterConfig.MasterOutputTrigger = TIM_TRGO_UPDATE;
sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
if (HAL_TIMEx_MasterConfigSynchronization(&htim3, &sMasterConfig) != HAL_OK)
{
    Error_Handler();
}
sConfigIC.ICPolarity = TIM_INPUTCHANNELPOLARITY_RISING;
sConfigIC.ICSelection = TIM_ICSELECTION_DIRECTTI;
sConfigIC.ICPrescaler = TIM_ICPSC_DIV1;
sConfigIC.ICFilter = 8;
if (HAL_TIM_IC_ConfigChannel(&htim3, &sConfigIC, TIM_CHANNEL_2) != HAL_OK)
{
    Error_Handler();
}
/* USER CODE BEGIN TIM3_Init 2 */
HAL_TIM_IC_Start_IT(&htim3, TIM_CHANNEL_2);
/* USER CODE END TIM3_Init 2 */

}

/**
 * @brief TIM4 Initialization Function
 * @param None
 * @retval None
 */
static void MX_TIM4_Init(void)
{

    /* USER CODE BEGIN TIM4_Init 0 */

    /* USER CODE END TIM4_Init 0 */

    TIM_ClockConfigTypeDef sClockSourceConfig = {0};
    TIM_MasterConfigTypeDef sMasterConfig = {0};

    /* USER CODE BEGIN TIM4_Init 1 */

    /* USER CODE END TIM4_Init 1 */
    htim4.Instance = TIM4;
    htim4.Init.Prescaler = 840-1;
    htim4.Init.CounterMode = TIM_COUNTERMODE_UP;
    htim4.Init.Period = 200-1;
    htim4.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
    htim4.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
    if (HAL_TIM_Base_Init(&htim4) != HAL_OK)

```

```

{
    Error_Handler();
}
sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
if (HAL_TIM_ConfigClockSource(&htim4, &sClockSourceConfig) != HAL_OK)
{
    Error_Handler();
}
sMasterConfig.MasterOutputTrigger = TIM_TRGO_UPDATE;
sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
if (HAL_TIMEx_MasterConfigSynchronization(&htim4, &sMasterConfig) != HAL_OK)
{
    Error_Handler();
}
/* USER CODE BEGIN TIM4_Init 2 */

/* USER CODE END TIM4_Init 2 */

}

/**
 * @brief USART2 Initialization Function
 * @param None
 * @retval None
 */
static void MX_USART2_UART_Init(void)
{

/* USER CODE BEGIN USART2_Init 0 */

/* USER CODE END USART2_Init 0 */

/* USER CODE BEGIN USART2_Init 1 */

/* USER CODE END USART2_Init 1 */
huart2.Instance = USART2;
huart2.Init.BaudRate = 115200;
huart2.Init.WordLength = UART_WORDLENGTH_8B;
huart2.Init.StopBits = UART_STOPBITS_1;
huart2.Init.Parity = UART_PARITY_NONE;
huart2.Init.Mode = UART_MODE_TX_RX;
huart2.Init.HwFlowCtl = UART_HWCONTROL_NONE;
huart2.Init.OverSampling = UART_OVERSAMPLING_16;
if (HAL_UART_Init(&huart2) != HAL_OK)
{
    Error_Handler();
}
/* USER CODE BEGIN USART2_Init 2 */

/* USER CODE END USART2_Init 2 */

```

```

}

/**
 * Enable DMA controller clock
 */
static void MX_DMA_Init(void)
{

    /* DMA controller clock enable */
    __HAL_RCC_DMA2_CLK_ENABLE();
    __HAL_RCC_DMA1_CLK_ENABLE();

    /* DMA interrupt init */
    /* DMA1_Stream5_IRQn interrupt configuration */
    HAL_NVIC_SetPriority(DMA1_Stream5_IRQn, 0, 0);
    HAL_NVIC_EnableIRQ(DMA1_Stream5_IRQn);
    /* DMA2_Stream0_IRQn interrupt configuration */
    HAL_NVIC_SetPriority(DMA2_Stream0_IRQn, 0, 0);
    HAL_NVIC_EnableIRQ(DMA2_Stream0_IRQn);

}

/**
 * @brief GPIO Initialization Function
 * @param None
 * @retval None
 */
static void MX_GPIO_Init(void)
{
    GPIO_InitTypeDef GPIO_InitStruct = {0};
    /* USER CODE BEGIN MX_GPIO_Init_1 */
    /* USER CODE END MX_GPIO_Init_1 */

    /* GPIO Ports Clock Enable */
    __HAL_RCC_GPIOC_CLK_ENABLE();
    __HAL_RCC_GPIOH_CLK_ENABLE();
    __HAL_RCC_GPIOA_CLK_ENABLE();
    __HAL_RCC_GPIOB_CLK_ENABLE();

    /*Configure GPIO pin Output Level */
    HAL_GPIO_WritePin(GPIOC, LedB_Pin|segD_Pin|segE_Pin|LedR_Pin
        |LedG_Pin, GPIO_PIN_RESET);

    /*Configure GPIO pin Output Level */
    HAL_GPIO_WritePin(userLed_GPIO_Port, userLed_Pin, GPIO_PIN_SET);

    /*Configure GPIO pin Output Level */
    HAL_GPIO_WritePin(GPIOA, dis1_Pin|dis2_Pin|dis4_Pin|segA_Pin, GPIO_PIN_RESET);

    /*Configure GPIO pin Output Level */
    HAL_GPIO_WritePin(GPIOB, dis3_Pin|segF_Pin|segB_Pin|segC_Pin

```

```

|segG_Pin, GPIO_PIN_RESET);

/*Configure GPIO pins : LedB_Pin segD_Pin segE_Pin LedR_Pin
   LedG_Pin */
GPIO_InitStruct.Pin = LedB_Pin | segD_Pin | segE_Pin | LedR_Pin
   | LedG_Pin;
GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
GPIO_InitStruct.Pull = GPIO_NOPULL;
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);

/*Configure GPIO pin : userLed_Pin */
GPIO_InitStruct.Pin = userLed_Pin;
GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
GPIO_InitStruct.Pull = GPIO_NOPULL;
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
HAL_GPIO_Init(userLed_GPIO_Port, &GPIO_InitStruct);

/*Configure GPIO pins : dis1_Pin dis2_Pin dis4_Pin segA_Pin */
GPIO_InitStruct.Pin = dis1_Pin | dis2_Pin | dis4_Pin | segA_Pin;
GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
GPIO_InitStruct.Pull = GPIO_NOPULL;
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

/*Configure GPIO pins : dis3_Pin segF_Pin segB_Pin segC_Pin
   segG_Pin */
GPIO_InitStruct.Pin = dis3_Pin | segF_Pin | segB_Pin | segC_Pin
   | segG_Pin;
GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
GPIO_InitStruct.Pull = GPIO_NOPULL;
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);

/*Configure GPIO pin : userData_Pin */
GPIO_InitStruct.Pin = userData_Pin;
GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
GPIO_InitStruct.Pull = GPIO_NOPULL;
HAL_GPIO_Init(userData_GPIO_Port, &GPIO_InitStruct);

/*Configure GPIO pin : userSw_Pin */
GPIO_InitStruct.Pin = userSw_Pin;
GPIO_InitStruct.Mode = GPIO_MODE_IT_RISING;
GPIO_InitStruct.Pull = GPIO_NOPULL;
HAL_GPIO_Init(userSw_GPIO_Port, &GPIO_InitStruct);

/*Configure GPIO pin : userClk_Pin */
GPIO_InitStruct.Pin = userClk_Pin;
GPIO_InitStruct.Mode = GPIO_MODE_IT_RISING;
GPIO_InitStruct.Pull = GPIO_NOPULL;
HAL_GPIO_Init(userClk_GPIO_Port, &GPIO_InitStruct);

```

```

/* EXTI interrupt init*/
HAL_NVIC_SetPriority(EXTI9_5_IRQn, 0, 0);
HAL_NVIC_EnableIRQ(EXTI9_5_IRQn);

HAL_NVIC_SetPriority(EXTI15_10_IRQn, 0, 0);
HAL_NVIC_EnableIRQ(EXTI15_10_IRQn);

/* USER CODE BEGIN MX_GPIO_Init_2 */
/* USER CODE END MX_GPIO_Init_2 */
}

/* USER CODE BEGIN 4 */
void medirTiempoEnvioUART(void) {
    const char *msg = "Hola mundo! - Examen Taller V\n";
    uint32_t start = DWT->CYCCNT;

    printf("%s", msg);

    uint32_t elapsed = DWT->CYCCNT - start;
    printf("Ciclos para enviar mensaje UART: %lu\n", (unsigned long)elapsed);
}

int __io_putchar(int ch) {
    HAL_UART_Transmit(&huart2, (uint8_t*)&ch, 1, HAL_MAX_DELAY);
    return ch;
}

void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim){
    if(htim->Instance==TIM2){
        HAL_GPIO_TogglePin(userLed_GPIO_Port,userLed_Pin);
    }else if(htim->Instance==TIM4){
        if (pending_event == IDLE)
            pending_event = EVENT_TIMER_TICK;
    }else if (htim->Instance==TIM3){

        counterOverflow++;
    }
}

void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
{
    if (GPIO_Pin == userClk_Pin) {
        data_snapshot = HAL_GPIO_ReadPin(userData_GPIO_Port, userData_Pin);
        if (pending_event == IDLE)
            pending_event = EVENT_ENCODER;
    } else if (GPIO_Pin == userSw_Pin) {
        if (pending_event == IDLE)
            pending_event = EVENT_SW;
    }
}

void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart){
    if(huart->Instance == USART2){
        if(rx_index<sizeof(rx_buffer)-1){

```

```

        rx_buffer[rx_index++]=rx_char;
    }
    if (rx_char == '@') {
        rx_buffer[rx_index]='\0';
        rx_index=0;
        pending_event = EVENT_USART; //aquí levanta tu bandera
    }
    HAL_UART_Receive_IT(huart, &rx_char,1);
}
}

void HAL_TIM_IC_CaptureCallback(TIM_HandleTypeDef *htim) {
    if (htim->Instance == TIM3 && htim->Channel == HAL_TIM_ACTIVE_CHANNEL_2) {
        static uint8_t ready = 0;

        if (ready == 0) {
            firstCapture = TIM3->CCR2;
            counterOverflow = 0;
            ready = 1;
        } else {
            secondCapture = TIM3->CCR2;

            if (secondCapture >= firstCapture) {
                elapsedTicks = secondCapture - firstCapture + (counterOverflow * 65536);
            } else {
                elapsedTicks = (0xFFFF - firstCapture + secondCapture + 1) + (counterOverflow * 65536);
            }

            if (elapsedTicks > 0 && pending_event == IDLE) {
                pending_event = EVENT_IC_CAPTURE;
            }

            ready = 0; // volver a medir siguiente ciclo
        }
    }
}

/* USER CODE END 4 */

/**
 * @brief This function is executed in case of error occurrence.
 * @retval None
 */
void Error_Handler(void)
{
    /* USER CODE BEGIN Error_Handler_Debug */
    /* User can add his own implementation to report the HAL error return state */
    __disable_irq();
    while (1)
    {
    }
    /* USER CODE END Error_Handler_Debug */
}

```

```
#ifdef USE_FULL_ASSERT
/**
 * @brief Reports the name of the source file and the source line number
 *        where the assert_param error has occurred.
 * @param file: pointer to the source file name
 * @param line: assert_param error line source number
 * @retval None
 */
void assert_failed(uint8_t *file, uint32_t line)
{
    /* USER CODE BEGIN 6 */
    /* User can add his own implementation to report the file name and line number,
       ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */
    /* USER CODE END 6 */
}
#endif /* USE_FULL_ASSERT */
```

Tu código ya contiene una gran parte de lo requerido en el enunciado, pero también tiene funcionalidades extra que no son necesarias para el reto. A continuación, te indico:

✓ FUNCIONALIDADES QUE DEBES CONSERVAR

Estas cumplen con el enunciado del examen:

1. Display de 7 segmentos – ✓

- Código que multiplica y muestra los dígitos (`displayNumber`, `nextDigit_FSM`) ya está bien hecho.
- FSM maneja eventos del encoder y botón (`EVENT_ENCODER`, `EVENT_SW`, etc.).

2. DMA para ADC – ✓

- Ya se hace lectura de `VRx` y `VRy` usando DMA con doble canal (`hadc1.Init.NbrOfConversion = 2`).
- Arreglo `adc_buffer[2*NUM_MUESTRAS]` con separación entre ejes ya está correcto.

3. USART2 a 115200 8N1 – ✓

- Configurado correctamente, con recepción por interrupción y comandos.

4. Control de LED RGB – ⚠

- Actualmente se controla por `HAL_GPIO_WritePin()`. Debes cambiar esto por PWM.

✗ FUNCIONALIDADES QUE DEBES BORRAR O COMENTAR

1. Todo lo relacionado con FFT y espectros:

- Borra o comenta:
 - `HandleFFTSIZECmd`
 - `HandlePrintFFT`
 - `HandleFFTInfo`

- `fft_size`, `input_f32`, `output_fft`, etc.
- `arm_math.h`
- `freq_buffer`, `freq_index`, `freq_full` y el `EVENT_IC_CAPTURE`
- `HandleFreqDisplayCmd()`

2. Comando `print` para imprimir ADC crudo (no es requerido):

- Borra `HandlePrintADC`.

3. Funciones de operaciones lógicas del crucigrama binario:

- Elimina `verificarOperacionesCrucigrama()` y `mostrarOperacion()`.

FUNCIONALIDADES QUE FALTAN POR IMPLEMENTAR

1. Transformación lineal de 0-4095 → 0-99

- En el `main()` o en el `EVENT_TIMER_TICK`, deberás transformar así:

```
c
uint8_t x = (uint8_t)((vr_x_buffer[0] * 99) / 4095);
uint8_t y = (uint8_t)((vr_y_buffer[0] * 99) / 4095);
display_value = (y * 100) + x; // dígitos izquierdos: Y, derechos: X
```

2. Control PWM del LED RGB (3 canales):

- Debes configurar 3 canales PWM (por ejemplo TIMx CH1, CH2, CH3) y usar `__HAL_TIM_SET_COMPARE()` para cambiar el duty en `HandleRGBCmd()`.



```
c
void HandleRGBCmd(const char *arg) {
    int r, g, b;
    sscanf(arg, "%d,%d,%d", &r, &g, &b);
    __HAL_TIM_SET_COMPARE(&htimX, TIM_CHANNEL_1, r);
    __HAL_TIM_SET_COMPARE(&htimX, TIM_CHANNEL_2, g);
    __HAL_TIM_SET_COMPARE(&htimX, TIM_CHANNEL_3, b);
}
```




- Recuerda configurar previamente los pines con `HAL_TIM_PWM_Start()`.

3. Contar ciclos para mostrar el mensaje UART

- Ya lo tienes implementado con `medirTiempoEnvioUART()`.

RESUMEN DE AJUSTES

Elemento	Acción
Display 7 segmentos	 Ya está implementado
DMA para ADC (joystick X/Y)	 Ya implementado
USART2 RX+TX con interrupción	 Ya implementado
LED RGB controlado por PWM	 Cambiar de GPIO a PWM
Transformación lineal ADC 0-4095 a 0-99	 Agregar fórmula y update

Elemento	Acción
Comando para cambiar PWM RGB	 Modificar HandleRGBCmd ()
Comando para ciclos UART	 Ya implementado
Funciones de FFT, espectro, ADC crudo, crucigrama	 Borrar o comentar

¿Deseas que te ayude a reescribir el **HandleRGBCmd ()** con PWM y a configurar los canales PWM correctamente desde cero (por ejemplo con TIM1 CH1/CH2/CH3)?



```

revisa que mas debo borrar quiero quedarme con mostrar operacion y crucigrama : /* USER CODE
BEGIN Header */
/**
*****
* @file      : main.c
* @brief     : Main program body
*****
* @attention
*
* Copyright (c) 2025 STMicroelectronics.
* All rights reserved.
*
* This software is licensed under terms that can be found in the LICENSE file
* in the root directory of this software component.
* If no LICENSE file comes with this software, it is provided AS-IS.
*
*****
*/
/* USER CODE END Header */
/* Includes -----*/
#include "main.h"

/* Private includes -----*/
/* USER CODE BEGIN Includes */
#include "fsm.h"
#include "stdint.h"
#include "stdio.h"
#include "string.h"
#include <stdlib.h>
#include "arm_math.h"
/* USER CODE END Includes */

/* Private typedef -----*/
/* USER CODE BEGIN PTD */

/* USER CODE END PTD */

/* Private define -----*/
/* USER CODE BEGIN PD */

```

```

/* USER CODE END PD */

/* Private macro -----*/
/* USER CODE BEGIN PM */

/* USER CODE END PM */

/* Private variables -----*/
ADC_HandleTypeDef hadc1;
DMA_HandleTypeDef hdma_adc1;

TIM_HandleTypeDef htim2;
TIM_HandleTypeDef htim3;
TIM_HandleTypeDef htim4;
DMA_HandleTypeDef hdma_tim3_ch2;

UART_HandleTypeDef huart2;

/* USER CODE BEGIN PV */
#define SIZE_MAX 2048
const uint32_t timer_clk = 84000000UL;
static uint16_t fft_size = 1024;
static uint16_t display_value = 0;
static uint8_t nextDigit_FSM = 1;
static uint8_t tx_buffer[256] = {0};
#define FREQ_BUFFER_SIZE 16
float freq_buffer[FREQ_BUFFER_SIZE] = {0};
#define NUM_MUESTRAS 1024
uint16_t adc_buffer[2 * NUM_MUESTRAS];
uint16_t vrx_buffer[NUM_MUESTRAS];
uint16_t vry_buffer[NUM_MUESTRAS];
uint8_t freq_index = 0;
uint8_t freq_full = 0;
char rx_buffer[128] = {0};
uint8_t rx_char = 0;
uint8_t rx_index=0;
volatile e_PosiblesEvents pending_event = IDLE;
volatile uint8_t data_snapshot = 0;

volatile uint8_t flagCapture = 0;
volatile uint32_t firstCapture = 0;
volatile uint32_t secondCapture = 0;
volatile uint32_t elapsedTicks = 0;
volatile uint32_t counterOverflow = 0;
/* USER CODE END PV */

/* Private function prototypes -----*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_DMA_Init(void);

```

```

static void MX_USART2_UART_Init(void);
static void MX_TIM2_Init(void);
static void MX_TIM4_Init(void);
static void MX_ADC1_Init(void);
static void MX_TIM3_Init(void);
/* USER CODE BEGIN PFP */
void displayNumber(uint8_t digitValue);
void HandleLEDDelayCmd(const char *arg);
void HandleSampleFreqCmd(const char *arg);
void HandleRGBCmd(const char *arg);
void HandlePWMFreqCmd(const char *arg);
void HandleUnknownCmd(void);
void HandleStatusCmd(void);
void HandlePrintADC(void);
void HandleFreqDisplayCmd(void);
void HandleClearCmd(void);
/* USER CODE END PFP */

/* Private user code -----*/
/* USER CODE BEGIN 0 */

e_PosiblesStates state_machine_action(e_PosiblesEvents event) {
    switch (event) {
        case IDLE:
            break;
        case EVENT_ENCODER:
            if (data_snapshot) // DATA snapshot
                display_value = (display_value == 0) ? 4095 : display_value - 1;
            else
                display_value = (display_value == 4095) ? 0 : display_value + 1;
            break;

        case EVENT_SW:
            display_value = 0;
            nextDigit_FSM = 1;
            break;

        case EVENT_TIMER_TICK:{
            // Apagar todos los dígitos
            HAL_GPIO_WritePin(GPIOA, GPIO_PIN_0, GPIO_PIN_RESET); // userDis1
            HAL_GPIO_WritePin(GPIOA, GPIO_PIN_1, GPIO_PIN_RESET); // userDis2
            HAL_GPIO_WritePin(GPIOB, GPIO_PIN_0, GPIO_PIN_RESET); // userDis3
            HAL_GPIO_WritePin(GPIOA, GPIO_PIN_4, GPIO_PIN_RESET); // userDis4

            switch (nextDigit_FSM) {
                case 1:
                    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_0, GPIO_PIN_SET);
                    displayNumber(display_value % 10);
                    break;
                case 2:
                    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_1, GPIO_PIN_SET);

```

```

        displayNumber((display_value / 10) % 10);
        break;
    case 3:
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_0, GPIO_PIN_SET);
        displayNumber((display_value / 100) % 10);
        break;
    case 4:
        HAL_GPIO_WritePin(GPIOA, GPIO_PIN_4, GPIO_PIN_SET);
        displayNumber((display_value / 1000) % 10);
        break;
    }
    nextDigit_FSM = (nextDigit_FSM < 4) ? nextDigit_FSM + 1 : 1;
    break;}
case EVENT_USART: {
    // eco
    int len = snprintf((char*)tx_buffer, sizeof(tx_buffer),
        "Comando recibido: %s\r\n", rx_buffer);
    HAL_UART_Transmit(&huart2, tx_buffer, len, 1000);

    // dispatch
    if (strncmp((char*)rx_buffer,"led=",4)==0)  HandleLEDDelayCmd(rx_buffer+4);
    else if (strncmp((char*)rx_buffer,"fmuestreo=",10)==0)
        HandleSampleFreqCmd(rx_buffer+10);
    else if (strncmp((char*)rx_buffer,"rgb=",4)==0)  HandleRGBCmd(rx_buffer+4);
    else if (strncmp((char*)rx_buffer,"status",6)==0) HandleStatusCmd();
    else if (strncmp((char*)rx_buffer,"freq",4)==0) HandleFreqDisplayCmd();
    else if (strncmp((char*)rx_buffer,"help",4)==0) HandleHelpCmd();
    else if (strncmp((char*)rx_buffer, "clear", 5) == 0) HandleClearCmd();
    else
        ;

    memset(rx_buffer,0,sizeof(rx_buffer));
    break;
}
case EVENT_IC_CAPTURE:{
    float periodo_ms = elapsedTicks * (1.0f / 84000000.0f) * 1000.0f; // En ms
    float frecuencia_hz = 84000000.0f / elapsedTicks;
    // Ejemplo: enviar por serial
    freq_buffer[freq_index++] = frecuencia_hz;
    if (freq_index >= FREQ_BUFFER_SIZE) {
        freq_index = 0;
        freq_full = 1;
    }break;
}return event;
}}

void displayNumber(uint8_t digitValue) {
    uint32_t start = DWT->CYCCNT;
    static uint8_t contador_muestras = 0;
    // Apagar todos los segmentos
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_10, GPIO_PIN_RESET); // A
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_10, GPIO_PIN_RESET); // B

```

```

HAL_GPIO_WritePin(GPIOB, GPIO_PIN_13, GPIO_PIN_RESET); // C
HAL_GPIO_WritePin(GPIOC, GPIO_PIN_0, GPIO_PIN_RESET); // D
HAL_GPIO_WritePin(GPIOC, GPIO_PIN_1, GPIO_PIN_RESET); // E
HAL_GPIO_WritePin(GPIOB, GPIO_PIN_1, GPIO_PIN_RESET); // F
HAL_GPIO_WritePin(GPIOB, GPIO_PIN_5, GPIO_PIN_RESET); // G

```

```

switch (digitValue) {

```

```

    case 0:

```

```

        HAL_GPIO_WritePin(GPIOA, GPIO_PIN_10, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_10, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_13, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOC, GPIO_PIN_0, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOC, GPIO_PIN_1, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_1, GPIO_PIN_SET);
        break;

```

```

    case 1:

```

```

        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_10, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_13, GPIO_PIN_SET);
        break;

```

```

    case 2:

```

```

        HAL_GPIO_WritePin(GPIOA, GPIO_PIN_10, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_10, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOC, GPIO_PIN_0, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOC, GPIO_PIN_1, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_5, GPIO_PIN_SET);
        break;

```

```

    case 3:

```

```

        HAL_GPIO_WritePin(GPIOA, GPIO_PIN_10, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_10, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_13, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOC, GPIO_PIN_0, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_5, GPIO_PIN_SET);
        break;

```

```

    case 4:

```

```

        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_10, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_13, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_1, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_5, GPIO_PIN_SET);
        break;

```

```

    case 5:

```

```

        HAL_GPIO_WritePin(GPIOA, GPIO_PIN_10, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_13, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOC, GPIO_PIN_0, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_1, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_5, GPIO_PIN_SET);
        break;

```

```

    case 6:

```

```

        HAL_GPIO_WritePin(GPIOA, GPIO_PIN_10, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_13, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOC, GPIO_PIN_0, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOC, GPIO_PIN_1, GPIO_PIN_SET);

```

```

    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_1, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_5, GPIO_PIN_SET);
    break;
case 7:
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_10, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_10, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_13, GPIO_PIN_SET);
    break;
case 8:
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_10, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_10, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_13, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOC, GPIO_PIN_0, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOC, GPIO_PIN_1, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_1, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_5, GPIO_PIN_SET);
    break;
case 9:
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_10, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_10, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_13, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_1, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_5, GPIO_PIN_SET);
    break;
}
uint32_t elapsed = DWT->CYCCNT - start;

if (contador_muestras < 10) {
    printf("Ciclos para mostrar digito %d: %lu\n", digitValue, (unsigned long)elapsed);
    contador_muestras++;
}
}

void HandleLEDDelayCmd(const char *arg) {
    uint32_t nuevo = atoi(arg);
    // Ajusta TIM2 para el blinky
    __HAL_TIM_SET_AUTORELOAD(&htim2, nuevo);
    __HAL_TIM_SET_COUNTER(&htim2, 0);
    // Feedback
    int len = snprintf((char*)tx_buffer, sizeof(tx_buffer),
        "LED delay = %lu ms\r\n", (unsigned long)nuevo);
    HAL_UART_Transmit(&huart2, tx_buffer, len, 1000);
}

void HandleSampleFreqCmd(const char *arg) {
    uint32_t fs;
    char option = arg[0];
    // Verifica la opción seleccionada (1, 2, 3, 4)
    switch(option) {
        case '1': fs = 44100; break;
        case '2': fs = 48000; break;
        case '3': fs = 96000; break;
    }
}

```

```

        case '4': fs = 128000; break;
        default: {
            const char *msg = "Opciones válidas para 'fmuestreo=' son: 1, 2, 3, 4\r\n";
            HAL_UART_Transmit(&huart2, (uint8_t*)msg, strlen(msg), 1000);
            return;
        }
    }
}

// calcular ARR y PSC
uint32_t arr = timer_clk / fs - 1;
uint32_t psc = 0;
if (arr > 0xFFFF) {
    // escalar PSC si supera 16 bits
    psc = (arr / 0x10000) + 1;
    arr = (timer_clk / (psc+1) / fs) - 1;
}

// aplicar a TIM3 y reiniciar
__HAL_TIM_SET_PRESCALER(&htim3, psc);
__HAL_TIM_SET_AUTORELOAD(&htim3, arr);
__HAL_TIM_SET_COUNTER(&htim3, 0);
HAL_TIM_Base_Stop(&htim3);
HAL_TIM_Base_Start(&htim3);
// Feedback
int len = snprintf((char*)tx_buffer, sizeof(tx_buffer),
    "Sample TIM3 @ %lu Hz (PSC=%lu, ARR=%lu)\r\n",
    (unsigned long)fs, (unsigned long)psc, (unsigned long)arr);
HAL_UART_Transmit(&huart2, tx_buffer, len, 1000);
}

void HandleRGBCmd(const char *arg) {
    // R, G, B si aparecen en arg
    GPIO_PinState R = (strchr(arg, 'R') ? GPIO_PIN_SET : GPIO_PIN_RESET);
    GPIO_PinState G = (strchr(arg, 'G') ? GPIO_PIN_SET : GPIO_PIN_RESET);
    GPIO_PinState B = (strchr(arg, 'B') ? GPIO_PIN_SET : GPIO_PIN_RESET);
    HAL_GPIO_WritePin(GPIOC, GPIO_PIN_3, R);
    HAL_GPIO_WritePin(GPIOC, GPIO_PIN_2, G);
    HAL_GPIO_WritePin(GPIOC, GPIO_PIN_13, B);
    // Feedback
    int len = snprintf((char*)tx_buffer, sizeof(tx_buffer),
        "RGB -> R:%c G:%c B:%c\r\n",
        R==GPIO_PIN_SET?'1':'0',
        G==GPIO_PIN_SET?'1':'0',
        B==GPIO_PIN_SET?'1':'0');
    HAL_UART_Transmit(&huart2, tx_buffer, len, 1000);
}

void HandleStatusCmd(void) {
    uint32_t psc = htim3.Init.Prescaler;
    uint32_t arr = htim3.Init.Period;
    float fs = 84000000.0f / ((psc + 1) * (arr + 1));
    float bin_res = fs / fft_size;

```



```
int len = snprintf((char*)tx_buffer, sizeof(tx_buffer),
    "Config:\r\nSample TIM3 @ %.2f Hz (PSC=%lu, ARR=%lu)\r\nFFT size: %u\r\nResolucion
espectral: %.2f Hz/bin\r\nCanal ADC: 6\r\nTrigger ADC: TIM3_TRGO\r\n",
    fs, (unsigned long)psc, (unsigned long)arr, fft_size, bin_res);
HAL_UART_Transmit(&huart2, tx_buffer, len, 1000);
}
```

```
void HandleFreqDisplayCmd(void) {
    char msg[64];
    float suma = 0.0f;
    int count = freq_full ? FREQ_BUFFER_SIZE : freq_index;

    HAL_UART_Transmit(&huart2, (uint8_t*)"Frecuencias IC (Hz):\r\n", 24, 100);

    for (int i = 0; i < count; i++) {
        int idx = (freq_index + i) % FREQ_BUFFER_SIZE;
        suma += freq_buffer[idx];
        int len = snprintf(msg, sizeof(msg), "%.2f\r\n", freq_buffer[idx]);
        HAL_UART_Transmit(&huart2, (uint8_t*)msg, len, 100);
    }

    float promedio = (count > 0) ? suma / count : 0.0f;
    int len = snprintf(msg, sizeof(msg), "Promedio: %.2f Hz\r\n", promedio);
    HAL_UART_Transmit(&huart2, (uint8_t*)msg, len, 100);
}
```

```
void HandleClearCmd(void) {
    const char *clear_screen = "\033[2J\033[H"; // Secuencia ANSI para limpiar la pantalla
    HAL_UART_Transmit(&huart2, (uint8_t*)clear_screen, strlen(clear_screen), 1000);
}
```

```
void HandleHelpCmd(void) {
    const char *help_msg =
        "\r\n===== AYUDA =====\r\n"
        "\r\nComandos disponibles:\r\n"
        " led=<ms>@ - Cambia la frecuencia del LED Blinky\r\n"
        " fmuestreo=<1|2|3|4>@ - Frecuencia de muestreo del ADC:\r\n"
        "     - 1 -> 44100 Hz\r\n"
        "     - 2 -> 48000 Hz\r\n"
        "     - 3 -> 96000 Hz\r\n"
        "     - 4 -> 128000 Hz\r\n"
        " rgb=<RGB>@ - Control de LED RGB, ej: rgb=RG\r\n"
        " status@ - Mostrar configuración actual\r\n"
        " print@ - Imprimir datos crudos del ADC\r\n"
        " help @ - Mostrar esta ayuda\r\n"
        "===== \r\n";
    HAL_UART_Transmit(&huart2, (uint8_t*)help_msg, strlen(help_msg), 1000);
}

void imprimirBinario(uint8_t valor) {
```

```

for (int i = 7; i >= 0; i--) {
    printf("%d", (valor >> i) & 1);
}
}

void mostrarOperacion(const char *desc, uint8_t resultado) {
    printf("%s\n", desc);
    printf("Binario: "); imprimirBinario(resultado);
    printf("\nHex: 0x%02X\n", resultado);
    printf("ASCII: %c\n\n", (resultado >= 32 && resultado <= 126) ? resultado : '.');
}

void verificarOperacionesCrucigrama(void) {
    // 1. Caracter ASCII 'B'
    uint8_t op1 = 'B';
    mostrarOperacion("1. ASCII 'B'", op1);

    // 2. ((!!0xC) << 0b11) | (~0x45)
    uint8_t op2 = ((!!0xC) << 0b11) | (~0x45);
    mostrarOperacion("2. ((!!0xC) << 0b11) | (~0x45)", op2);

    // 3. (0b00111100 & 0xAC) << 0x3
    uint8_t op3 = (0b00111100 & 0xAC) << 0x3;
    mostrarOperacion("3. (0b00111100 & 0xAC) << 0x3)", op3);

    // 4. -121
    uint8_t op4 = (uint8_t)(-121);
    mostrarOperacion("4. -121 (uint8_t)", op4);

    // 5. (0xCAFE & 0xFF0) >> 7
    uint16_t val5 = (0xCAFE & 0xFF0) >> 7;
    uint8_t op5 = (uint8_t)(val5 & 0xFF);
    mostrarOperacion("5. (0xCAFE & 0xFF0) >> 7", op5);

    // 6. (1 << 4) + (0x3 << 4) - 10 + (0b11 << 0x6)
    uint8_t op6 = (1 << 4) + (0x3 << 4) - 10 + (0b11 << 6);
    mostrarOperacion("6. (1<<4)+(0x3<<4)-10+(0b11<<6)", op6);

    // 7. ('T' | '5')
    uint8_t op7 = 'T' | '5';
    mostrarOperacion("7. ('T' | '5')", op7);

    // 8. (2921 & 0xD6)
    uint8_t op8 = 2921 & 0xD6;
    mostrarOperacion("8. (2921 & 0xD6)", op8);

    // 9. 01100110 ^ 00101011
    uint8_t a9 = 0b01100110;
    uint8_t b9 = 0b00101011;
    uint8_t op9 = a9 ^ b9;
    mostrarOperacion("9. 01100110 ^ 00101011", op9);
}

```

```

}
/* USER CODE END 0 */

/**
 * @brief The application entry point.
 * @retval int
 */
int main(void)
{

    /* USER CODE BEGIN 1 */

    /* USER CODE END 1 */

    /* MCU Configuration-----*/

    /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
    HAL_Init();

    /* USER CODE BEGIN Init */
    CoreDebug->DEMCR |= CoreDebug_DEMCR_TRCENA_Msk; // Habilitar DWT
    DWT->CYCCNT = 0; // Reiniciar contador
    DWT->CTRL |= DWT_CTRL_CYCCNTENA_Msk; // Encender contador de ciclos
    /* USER CODE END Init */

    /* Configure the system clock */
    SystemClock_Config();

    /* USER CODE BEGIN SysInit */

    /* USER CODE END SysInit */

    /* Initialize all configured peripherals */
    MX_GPIO_Init();
    MX_DMA_Init();
    MX_USART2_UART_Init();
    MX_TIM2_Init();
    MX_TIM4_Init();
    MX_ADC1_Init();
    MX_TIM3_Init();
    /* USER CODE BEGIN 2 */
    HAL_TIM_Base_Start_IT(&htim2);
    HAL_TIM_Base_Start_IT(&htim3);
    HAL_TIM_Base_Start_IT(&htim4);
    HAL_UART_Receive_IT(&huart2,&rx_char,1);
    HAL_ADC_Start_DMA(&hadc1, (uint32_t*)adc_buffer, fft_size);
    HAL_TIM_IC_Start_IT(&htim3, TIM_CHANNEL_2);
    HandleSampleFreqCmd(3); //96000 hz muestreo

    verificarOperacionesCrucigrama();
    medirTiempoEnvioUART();

```

```

HandleHelpCmd();

/* USER CODE END 2 */

/* Infinite loop */
/* USER CODE BEGIN WHILE */
while (1)
{
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
    if (pending_event != IDLE){
        state_machine_action(pending_event);
        pending_event = IDLE;
    }
}
/* USER CODE END 3 */
}

/**
 * @brief System Clock Configuration
 * @retval None
 */
void SystemClock_Config(void)
{
    RCC_OscInitTypeDef RCC_OscInitStruct = {0};
    RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};

    /** Configure the main internal regulator output voltage
    */
    __HAL_RCC_PWR_CLK_ENABLE();
    __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE1);

    /** Initializes the RCC Oscillators according to the specified parameters
    * in the RCC_OscInitTypeDef structure.
    */
    RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI;
    RCC_OscInitStruct.HSISState = RCC_HSI_ON;
    RCC_OscInitStruct.HSICalibrationValue = RCC_HSICALIBRATION_DEFAULT;
    RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
    RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSI;
    RCC_OscInitStruct.PLL.PLLM = 16;
    RCC_OscInitStruct.PLL.PLLN = 336;
    RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV4;
    RCC_OscInitStruct.PLL.PLLQ = 4;
    if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
    {
        Error_Handler();
    }

    /** Initializes the CPU, AHB and APB buses clocks

```

```

*/
RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
                             |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV2;
RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;

if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_2) != HAL_OK)
{
    Error_Handler();
}
}

/**
 * @brief ADC1 Initialization Function
 * @param None
 * @retval None
 */
static void MX_ADC1_Init(void)
{

    /* USER CODE BEGIN ADC1_Init 0 */

    /* USER CODE END ADC1_Init 0 */

    ADC_ChannelConfTypeDef sConfig = {0};

    /* USER CODE BEGIN ADC1_Init 1 */

    /* USER CODE END ADC1_Init 1 */

    /** Configure the global features of the ADC (Clock, Resolution, Data Alignment and number of
    conversion)
    */
    hadc1.Instance = ADC1;
    hadc1.Init.ClockPrescaler = ADC_CLOCK_SYNC_PCLK_DIV4;
    hadc1.Init.Resolution = ADC_RESOLUTION_12B;
    hadc1.Init.ScanConvMode = ENABLE;
    hadc1.Init.ContinuousConvMode = DISABLE;
    hadc1.Init.DiscontinuousConvMode = DISABLE;
    hadc1.Init.ExternalTrigConvEdge = ADC_EXTERNALTRIGCONVEDGE_RISING;
    hadc1.Init.ExternalTrigConv = ADC_EXTERNALTRIGCONV_T3_TRGO;
    hadc1.Init.DataAlign = ADC_DATAALIGN_RIGHT;
    hadc1.Init.NbrOfConversion = 2;
    hadc1.Init.DMAContinuousRequests = ENABLE;
    hadc1.Init.EOCSelection = ADC_EOC_SINGLE_CONV;
    if (HAL_ADC_Init(&hadc1) != HAL_OK)
    {
        Error_Handler();
    }
}

```

```

/** Configure for the selected ADC regular channel its corresponding rank in the sequencer and its
sample time.
*/
sConfig.Channel = ADC_CHANNEL_6;
sConfig.Rank = 1;
sConfig.SamplingTime = ADC_SAMPLETIME_3CYCLES;
if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
{
    Error_Handler();
}

/** Configure for the selected ADC regular channel its corresponding rank in the sequencer and its
sample time.
*/
sConfig.Rank = 2;
if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
{
    Error_Handler();
}
/* USER CODE BEGIN ADC1_Init 2 */

/* USER CODE END ADC1_Init 2 */

}

/**
 * @brief TIM2 Initialization Function
 * @param None
 * @retval None
 */
static void MX_TIM2_Init(void)
{

/* USER CODE BEGIN TIM2_Init 0 */

/* USER CODE END TIM2_Init 0 */

TIM_ClockConfigTypeDef sClockSourceConfig = {0};
TIM_MasterConfigTypeDef sMasterConfig = {0};

/* USER CODE BEGIN TIM2_Init 1 */

/* USER CODE END TIM2_Init 1 */
htim2.Instance = TIM2;
htim2.Init.Prescaler = 42000-1;
htim2.Init.CounterMode = TIM_COUNTERMODE_UP;
htim2.Init.Period = 500;
htim2.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
htim2.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_ENABLE;
if (HAL_TIM_Base_Init(&htim2) != HAL_OK)

```

```

{
    Error_Handler();
}
sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
if (HAL_TIM_ConfigClockSource(&htim2, &sClockSourceConfig) != HAL_OK)
{
    Error_Handler();
}
sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
if (HAL_TIMEx_MasterConfigSynchronization(&htim2, &sMasterConfig) != HAL_OK)
{
    Error_Handler();
}
/* USER CODE BEGIN TIM2_Init 2 */

/* USER CODE END TIM2_Init 2 */

}

/**
 * @brief TIM3 Initialization Function
 * @param None
 * @retval None
 */
static void MX_TIM3_Init(void)
{
    /* USER CODE BEGIN TIM3_Init 0 */

    /* USER CODE END TIM3_Init 0 */

    TIM_ClockConfigTypeDef sClockSourceConfig = {0};
    TIM_MasterConfigTypeDef sMasterConfig = {0};
    TIM_IC_InitTypeDef sConfigIC = {0};

    /* USER CODE BEGIN TIM3_Init 1 */

    /* USER CODE END TIM3_Init 1 */
    htim3.Instance = TIM3;
    htim3.Init.Prescaler = 0;
    htim3.Init.CounterMode = TIM_COUNTERMODE_UP;
    htim3.Init.Period = 65535;
    htim3.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
    htim3.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
    if (HAL_TIM_Base_Init(&htim3) != HAL_OK)
    {
        Error_Handler();
    }
    sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
    if (HAL_TIM_ConfigClockSource(&htim3, &sClockSourceConfig) != HAL_OK)

```

```

{
    Error_Handler();
}
if (HAL_TIM_IC_Init(&htim3) != HAL_OK)
{
    Error_Handler();
}
sMasterConfig.MasterOutputTrigger = TIM_TRGO_UPDATE;
sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
if (HAL_TIMEx_MasterConfigSynchronization(&htim3, &sMasterConfig) != HAL_OK)
{
    Error_Handler();
}
sConfigIC.ICPolarity = TIM_INPUTCHANNELPOLARITY_RISING;
sConfigIC.ICSelection = TIM_ICSELECTION_DIRECTTI;
sConfigIC.ICPrescaler = TIM_ICPSC_DIV1;
sConfigIC.ICFilter = 8;
if (HAL_TIM_IC_ConfigChannel(&htim3, &sConfigIC, TIM_CHANNEL_2) != HAL_OK)
{
    Error_Handler();
}
/* USER CODE BEGIN TIM3_Init 2 */
HAL_TIM_IC_Start_IT(&htim3, TIM_CHANNEL_2);
/* USER CODE END TIM3_Init 2 */

}

/**
 * @brief TIM4 Initialization Function
 * @param None
 * @retval None
 */
static void MX_TIM4_Init(void)
{
    /* USER CODE BEGIN TIM4_Init 0 */

    /* USER CODE END TIM4_Init 0 */

    TIM_ClockConfigTypeDef sClockSourceConfig = {0};
    TIM_MasterConfigTypeDef sMasterConfig = {0};

    /* USER CODE BEGIN TIM4_Init 1 */

    /* USER CODE END TIM4_Init 1 */
    htim4.Instance = TIM4;
    htim4.Init.Prescaler = 840-1;
    htim4.Init.CounterMode = TIM_COUNTERMODE_UP;
    htim4.Init.Period = 200-1;
    htim4.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
    htim4.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;

```



```

if (HAL_TIM_Base_Init(&htim4) != HAL_OK)
{
    Error_Handler();
}
sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
if (HAL_TIM_ConfigClockSource(&htim4, &sClockSourceConfig) != HAL_OK)
{
    Error_Handler();
}
sMasterConfig.MasterOutputTrigger = TIM_TRGO_UPDATE;
sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
if (HAL_TIMEx_MasterConfigSynchronization(&htim4, &sMasterConfig) != HAL_OK)
{
    Error_Handler();
}
/* USER CODE BEGIN TIM4_Init 2 */

/* USER CODE END TIM4_Init 2 */

}

/**
 * @brief USART2 Initialization Function
 * @param None
 * @retval None
 */
static void MX_USART2_UART_Init(void)
{
    /* USER CODE BEGIN USART2_Init 0 */

    /* USER CODE END USART2_Init 0 */

    /* USER CODE BEGIN USART2_Init 1 */

    /* USER CODE END USART2_Init 1 */
    huart2.Instance = USART2;
    huart2.Init.BaudRate = 115200;
    huart2.Init.WordLength = UART_WORDLENGTH_8B;
    huart2.Init.StopBits = UART_STOPBITS_1;
    huart2.Init.Parity = UART_PARITY_NONE;
    huart2.Init.Mode = UART_MODE_TX_RX;
    huart2.Init.HwFlowCtl = UART_HWCONTROL_NONE;
    huart2.Init.OverSampling = UART_OVERSAMPLING_16;
    if (HAL_UART_Init(&huart2) != HAL_OK)
    {
        Error_Handler();
    }
    /* USER CODE BEGIN USART2_Init 2 */

    /* USER CODE END USART2_Init 2 */

```

```

}

/**
 * Enable DMA controller clock
 */
static void MX_DMA_Init(void)
{
    /* DMA controller clock enable */
    __HAL_RCC_DMA2_CLK_ENABLE();
    __HAL_RCC_DMA1_CLK_ENABLE();

    /* DMA interrupt init */
    /* DMA1_Stream5_IRQn interrupt configuration */
    HAL_NVIC_SetPriority(DMA1_Stream5_IRQn, 0, 0);
    HAL_NVIC_EnableIRQ(DMA1_Stream5_IRQn);
    /* DMA2_Stream0_IRQn interrupt configuration */
    HAL_NVIC_SetPriority(DMA2_Stream0_IRQn, 0, 0);
    HAL_NVIC_EnableIRQ(DMA2_Stream0_IRQn);
}

/**
 * @brief GPIO Initialization Function
 * @param None
 * @retval None
 */
static void MX_GPIO_Init(void)
{
    GPIO_InitTypeDef GPIO_InitStruct = {0};
    /* USER CODE BEGIN MX_GPIO_Init_1 */
    /* USER CODE END MX_GPIO_Init_1 */

    /* GPIO Ports Clock Enable */
    __HAL_RCC_GPIOC_CLK_ENABLE();
    __HAL_RCC_GPIOH_CLK_ENABLE();
    __HAL_RCC_GPIOA_CLK_ENABLE();
    __HAL_RCC_GPIOB_CLK_ENABLE();

    /*Configure GPIO pin Output Level */
    HAL_GPIO_WritePin(GPIOC, LedB_Pin | segD_Pin | segE_Pin | LedR_Pin
        | LedG_Pin, GPIO_PIN_RESET);

    /*Configure GPIO pin Output Level */
    HAL_GPIO_WritePin(userLed_GPIO_Port, userLed_Pin, GPIO_PIN_SET);

    /*Configure GPIO pin Output Level */
    HAL_GPIO_WritePin(GPIOA, dis1_Pin | dis2_Pin | dis4_Pin | segA_Pin, GPIO_PIN_RESET);

    /*Configure GPIO pin Output Level */

```

```

HAL_GPIO_WritePin(GPIOB, dis3_Pin|segF_Pin|segB_Pin|segC_Pin
                  |segG_Pin, GPIO_PIN_RESET);

/*Configure GPIO pins : LedB_Pin segD_Pin segE_Pin LedR_Pin
   LedG_Pin */
GPIO_InitStruct.Pin = LedB_Pin|segD_Pin|segE_Pin|LedR_Pin
                  |LedG_Pin;
GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
GPIO_InitStruct.Pull = GPIO_NOPULL;
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);

/*Configure GPIO pin : userLed_Pin */
GPIO_InitStruct.Pin = userLed_Pin;
GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
GPIO_InitStruct.Pull = GPIO_NOPULL;
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
HAL_GPIO_Init(userLed_GPIO_Port, &GPIO_InitStruct);

/*Configure GPIO pins : dis1_Pin dis2_Pin dis4_Pin segA_Pin */
GPIO_InitStruct.Pin = dis1_Pin|dis2_Pin|dis4_Pin|segA_Pin;
GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
GPIO_InitStruct.Pull = GPIO_NOPULL;
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

/*Configure GPIO pins : dis3_Pin segF_Pin segB_Pin segC_Pin
   segG_Pin */
GPIO_InitStruct.Pin = dis3_Pin|segF_Pin|segB_Pin|segC_Pin
                  |segG_Pin;
GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
GPIO_InitStruct.Pull = GPIO_NOPULL;
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);

/*Configure GPIO pin : userData_Pin */
GPIO_InitStruct.Pin = userData_Pin;
GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
GPIO_InitStruct.Pull = GPIO_NOPULL;
HAL_GPIO_Init(userData_GPIO_Port, &GPIO_InitStruct);

/*Configure GPIO pin : userSw_Pin */
GPIO_InitStruct.Pin = userSw_Pin;
GPIO_InitStruct.Mode = GPIO_MODE_IT_RISING;
GPIO_InitStruct.Pull = GPIO_NOPULL;
HAL_GPIO_Init(userSw_GPIO_Port, &GPIO_InitStruct);

/*Configure GPIO pin : userClk_Pin */
GPIO_InitStruct.Pin = userClk_Pin;
GPIO_InitStruct.Mode = GPIO_MODE_IT_RISING;
GPIO_InitStruct.Pull = GPIO_NOPULL;

```

```

HAL_GPIO_Init(userClk_GPIO_Port, &GPIO_InitStruct);

/* EXTI interrupt init*/
HAL_NVIC_SetPriority(EXTI9_5_IRQn, 0, 0);
HAL_NVIC_EnableIRQ(EXTI9_5_IRQn);

HAL_NVIC_SetPriority(EXTI15_10_IRQn, 0, 0);
HAL_NVIC_EnableIRQ(EXTI15_10_IRQn);

/* USER CODE BEGIN MX_GPIO_Init_2 */
/* USER CODE END MX_GPIO_Init_2 */
}

/* USER CODE BEGIN 4 */
void medirTiempoEnvioUART(void) {
    const char *msg = "Hola mundo! - Examen Taller V\n";
    uint32_t start = DWT->CYCCNT;

    printf("%s", msg);

    uint32_t elapsed = DWT->CYCCNT - start;
    printf("Ciclos para enviar mensaje UART: %lu\n", (unsigned long)elapsed);
}
int __io_putchar(int ch) {
    HAL_UART_Transmit(&huart2, (uint8_t*)&ch, 1, HAL_MAX_DELAY);
    return ch;
}
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim){
    if(htim->Instance==TIM2){
        HAL_GPIO_TogglePin(userLed_GPIO_Port,userLed_Pin);
    }else if(htim->Instance==TIM4){
        if (pending_event == IDLE)
            pending_event = EVENT_TIMER_TICK;
    }else if (htim->Instance==TIM3){

        counterOverflow++;
    }
}
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
{
    if (GPIO_Pin == userClk_Pin) {
        data_snapshot = HAL_GPIO_ReadPin(userData_GPIO_Port, userData_Pin);
        if (pending_event == IDLE)
            pending_event = EVENT_ENCODER;
    } else if (GPIO_Pin == userSw_Pin) {
        if (pending_event == IDLE)
            pending_event = EVENT_SW;
    }
}
void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart){
    if(huart->Instance == USART2){

```

```

        if(rx_index<sizeof(rx_buffer)-1){
            rx_buffer[rx_index++]=rx_char;
        }
    if (rx_char == '@') {
        rx_buffer[rx_index]='\0';
        rx_index=0;
        pending_event = EVENT_USART; //aquí levanta tu bandera
    }
    HAL_UART_Receive_IT(huart, &rx_char,1);
}
}

void HAL_TIM_IC_CaptureCallback(TIM_HandleTypeDef *htim) {
    if (htim->Instance == TIM3 && htim->Channel == HAL_TIM_ACTIVE_CHANNEL_2) {
        static uint8_t ready = 0;

        if (ready == 0) {
            firstCapture = TIM3->CCR2;
            counterOverflow = 0;
            ready = 1;
        } else {
            secondCapture = TIM3->CCR2;

            if (secondCapture >= firstCapture) {
                elapsedTicks = secondCapture - firstCapture + (counterOverflow * 65536);
            } else {
                elapsedTicks = (0xFFFF - firstCapture + secondCapture + 1) + (counterOverflow * 65536);
            }

            if (elapsedTicks > 0 && pending_event == IDLE) {
                pending_event = EVENT_IC_CAPTURE;
            }

            ready = 0; // volver a medir siguiente ciclo
        }
    }
}

/* USER CODE END 4 */

/**
 * @brief This function is executed in case of error occurrence.
 * @retval None
 */
void Error_Handler(void)
{
    /* USER CODE BEGIN Error_Handler_Debug */
    /* User can add his own implementation to report the HAL error return state */
    __disable_irq();
    while (1)
    {
    }
    /* USER CODE END Error_Handler_Debug */
}

```

```

}

#ifdef USE_FULL_ASSERT
/**
 * @brief Reports the name of the source file and the source line number
 *        where the assert_param error has occurred.
 * @param file: pointer to the source file name
 * @param line: assert_param error line source number
 * @retval None
 */
void assert_failed(uint8_t *file, uint32_t line)
{
    /* USER CODE BEGIN 6 */
    /* User can add his own implementation to report the file name and line number,
       ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */
    /* USER CODE END 6 */
}
#endif /* USE_FULL_ASSERT */

```

Perfecto. Como deseas **conservar** el bloque del crucigrama binario (**verificarOperacionesCrucigrama()** y **mostrarOperacion()**), y estás **limpiando** el proyecto para cumplir exactamente con el enunciado del examen, a continuación te indico **todo lo que debes eliminar** o comentar.

✂ COSAS QUE DEBES BORRAR / COMENTAR

1. ✗ Inclusión de librería innecesaria

```

c

#include "arm_math.h"

```

➡ Bórrala. Ya no usas FFT ni funciones DSP.

2. ✗ Variables y buffers que ya no se usan

```

c

#define SIZE_MAX 2048 // <-- si no lo usas, elimínalo
float freq_buffer[FREQ_BUFFER_SIZE] = {0};
uint8_t freq_index = 0;
uint8_t freq_full = 0;
volatile uint8_t flagCapture = 0;
volatile uint32_t firstCapture = 0;
volatile uint32_t secondCapture = 0;
volatile uint32_t elapsedTicks = 0;
volatile uint32_t counterOverflow = 0;

```

➡ Elimina todas estas variables, ya no se usan al eliminar **EVENT_IC_CAPTURE** y **HandleFreqDisplayCmd**.

3. ✗ Función y evento IC (Input Capture)

```
c

void HAL_TIM_IC_CaptureCallback(TIM_HandleTypeDef *htim) { ... }
```

→ Borra toda esta función si ya no usas captura de flanco para medir frecuencia.

Y también este caso en `state_machine_action()`:

```
c

case EVENT_IC_CAPTURE: { ... } break;
```

4. ✗ `HandleFreqDisplayCmd`

```
c

void HandleFreqDisplayCmd(void) { ... }
```

→ Puedes eliminar completamente esta función.

Y también su referencia en el `dispatcher`:

```
c

else if (strncmp((char*)rx_buffer,"freq",4)==0) HandleFreqDisplayCmd();
```

5. ✗ `fft_size` y referencias a FFT

```
c

static uint16_t fft_size = 1024;
```

→ Bórralo. Ya no se necesita.

En `HandleStatusCmd()` también borra esta línea:

```
c

float bin_res = fs / fft_size;
```

Y cambia el mensaje, por ejemplo:

```
c

"Config:\r\nSample TIM3 @ %.2f Hz (PSC=%lu, ARR=%lu)\r\nCanal ADC: 6 y 7\r\n",
fs, (unsigned long)psc, (unsigned long)arr;
```

6. ✗ `print@` del mensaje de ayuda

En `HandleHelpCmd()` elimina:

```
c
" print@          - Imprimir datos crudos del ADC\r\n"
```

✓ COSAS QUE ESTÁN BIEN Y DEBES CONSERVAR

- `verificarOperacionesCrucigrama()` ✓
- `mostrarOperacion()` ✓
- `imprimirBinario()` ✓
- `displayNumber()` y el manejo del display ✓
- DMA ADC para joystick ✓
- `medirTiempoEnvioUART()` ✓
- FSM de `EVENT_ENCODER`, `EVENT_SW`, `EVENT_TIMER_TICK`, `EVENT_USART` ✓
- USART + comandos + `HAL_UART_RxCpltCallback()` ✓
- Timer del LED de estado ✓
- Transformación de joystick a display (debes implementarla aún, pero la base está ✓)

SUGERENCIA EXTRA

Agrega dentro de `EVENT_TIMER_TICK` el código para transformar `vr_x_buffer[0]` y `vry_buffer[0]` en el valor a mostrar:

```
c
uint8_t x_val = (vr_x_buffer[0] * 99) / 4095;
uint8_t y_val = (vry_buffer[0] * 99) / 4095;
display_value = (y_val * 100) + x_val;
```

¿Te gustaría que te entregue un diff o parche con los cambios limpios listos para copiar y pegar?



propone pines para cambiar los led RGB que funcione con pwm y que pueda por serial cambiar los valores así RGB = 255 255 255

Perfecto. A continuación te propongo una configuración **funcional** para controlar los tres canales del LED RGB mediante PWM usando el **STM32F4xx**, en tu caso probablemente un STM32F401/STM32F411 o similar.

✓ Objetivo

- Controlar los tres LEDs (Rojo, Verde, Azul) con PWM usando un temporizador.

- Modificar los valores por **UART** usando un comando tipo:

```
ini
rgb=255 128 0@
```

🔧 Propuesta de Pines PWM para el LED RGB

Usaremos **TIM1** (porque tiene 3 canales completos y está disponible en la mayoría de placas STM32).

Canal PWM	Color LED	Pin sugerido	Alternativa TIM1
CH1	Rojo	PA8	TIM1_CH1
CH2	Verde	PA9	TIM1_CH2
CH3	Azul	PA10	TIM1_CH3

Asegúrate de que estos pines están disponibles en tu diseño de hardware y no estén usados por el display.

⚙️ Configuración en STM32CubeMX / Código

1. **TIM1** → **PWM Generation CH1, CH2, CH3**
2. Frecuencia: 1 kHz (o la que desees)
3. Resolución: 8 bits (ARR = 255)
4. Inicia los canales PWM con `HAL_TIM_PWM_Start()` para cada canal.

🔧 Código de inicialización de PWM

```
c
// En MX_TIM1_Init (agregar si no existe)
htim1.Instance = TIM1;
htim1.Init.Prescaler = 83;
htim1.Init.CounterMode = TIM_COUNTERMODE_UP;
htim1.Init.Period = 255; // 8 bits
htim1.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
htim1.Init.RepetitionCounter = 0;
htim1.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
HAL_TIM_PWM_Init(&htim1);

// Configurar cada canal
TIM_OC_InitTypeDef sConfigOC = {0};
sConfigOC.OCMode = TIM_OCMODE_PWM1;
sConfigOC.Pulse = 0;
sConfigOC.OCpolarity = TIM_OCPOLARITY_HIGH;
sConfigOC.OCFastMode = TIM_OCFAST_DISABLE;

HAL_TIM_PWM_ConfigChannel(&htim1, &sConfigOC, TIM_CHANNEL_1);
HAL_TIM_PWM_ConfigChannel(&htim1, &sConfigOC, TIM_CHANNEL_2);
HAL_TIM_PWM_ConfigChannel(&htim1, &sConfigOC, TIM_CHANNEL_3);
```

Y en `main()` después de la inicialización:

c

```
HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_1); // Rojo
HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_2); // Verde
HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_3); // Azul
```

Comando por UART

Reemplaza tu función `HandleRGBCmd()` por esta nueva versión que admite valores de 0–255:

c

```
void HandleRGBCmd(const char *arg) {
    int r = 0, g = 0, b = 0;
    if (sscanf(arg, "%d %d %d", &r, &g, &b) == 3) {
        if (r > 255) r = 255;
        if (g > 255) g = 255;
        if (b > 255) b = 255;

        __HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_1, r); // Rojo
        __HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_2, g); // Verde
        __HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_3, b); // Azul

        int len = snprintf((char*)tx_buffer, sizeof(tx_buffer),
            "RGB PWM -> R:%3d G:%3d B:%3d\r\n", r, g, b);
        HAL_UART_Transmit(&huart2, tx_buffer, len, 1000);
    } else {
        const char *msg = "Formato inválido. Usa: rgb=R G B\r\n";
        HAL_UART_Transmit(&huart2, (uint8_t*)msg, strlen(msg), 1000);
    }
}
```

Resumen

- ✓ Usa TIM1_CH1/2/3 para PA8/9/10.
- ✓ ARR = 255, para controlar intensidad de color con 8 bits.
- ✓ Comando por UART: `rgb=R G B`
- 🔧 Recuerda agregar resistencias limitadoras para los LEDs.

¿Deseas que genere un bloque de código completo listo para pegar en tu proyecto (incluyendo `MX_TIM1_Init` y `HandleRGBCmd()` ya integrados)?



verifica mi código y dime si ya lo puedo probar el pwm para el led e intentar hacer diferentes colores mira mi código y también que me falta de la imagen de arriba para cumplir lo que me piden

```
/* USER CODE BEGIN Header */
```

```
/**
```

```
*****
```

```
* @file      : main.c
```

```
* @brief     : Main program body
```

```
*****
```

```
* @attention
```

```

*
* Copyright (c) 2025 STMicroelectronics.
* All rights reserved.
*
* This software is licensed under terms that can be found in the LICENSE file
* in the root directory of this software component.
* If no LICENSE file comes with this software, it is provided AS-IS.
*
*****
*/
/* USER CODE END Header */
/* Includes -----*/
#include "main.h"

/* Private includes -----*/
/* USER CODE BEGIN Includes */
#include "fsm.h"
#include "stdint.h"
#include "stdio.h"
#include "string.h"
#include <stdlib.h>
/* USER CODE END Includes */

/* Private typedef -----*/
/* USER CODE BEGIN PTD */

/* USER CODE END PTD */

/* Private define -----*/
/* USER CODE BEGIN PD */

/* USER CODE END PD */

/* Private macro -----*/
/* USER CODE BEGIN PM */

/* USER CODE END PM */

/* Private variables -----*/
ADC_HandleTypeDef hadc1;
DMA_HandleTypeDef hdma_adc1;

TIM_HandleTypeDef htim1;
TIM_HandleTypeDef htim2;
TIM_HandleTypeDef htim3;
TIM_HandleTypeDef htim4;
DMA_HandleTypeDef hdma_tim3_ch2;

UART_HandleTypeDef huart2;

/* USER CODE BEGIN PV */

```

```

const uint32_t timer_clk = 84000000UL;
static uint16_t fft_size = 1024;
static uint16_t display_value = 0;
static uint8_t nextDigit_FSM = 1;
static uint8_t tx_buffer[256] = {0};
#define FREQ_BUFFER_SIZE 16
float freq_buffer[FREQ_BUFFER_SIZE] = {0};
#define NUM_MUESTRAS 1024
uint16_t adc_buffer[2 * NUM_MUESTRAS];
uint16_t vrx_buffer[NUM_MUESTRAS];
uint16_t vry_buffer[NUM_MUESTRAS];

char rx_buffer[128] = {0};
uint8_t rx_char = 0;
uint8_t rx_index=0;
volatile e_PosiblesEvents pending_event = IDLE;
volatile uint8_t data_snapshot = 0;

/* USER CODE END PV */

/* Private function prototypes -----*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_DMA_Init(void);
static void MX_USART2_UART_Init(void);
static void MX_TIM2_Init(void);
static void MX_TIM4_Init(void);
static void MX_ADC1_Init(void);
static void MX_TIM3_Init(void);
static void MX_TIM1_Init(void);
/* USER CODE BEGIN PFP */
void displayNumber(uint8_t digitValue);
void HandleLEDDelayCmd(const char *arg);
void HandleSampleFreqCmd(const char *arg);
void HandleRGBCmd(const char *arg);
void HandlePWMFreqCmd(const char *arg);
void HandleUnknownCmd(void);
void HandleStatusCmd(void);
void HandlePrintADC(void);
void HandleClearCmd(void) ;
/* USER CODE END PFP */

/* Private user code -----*/
/* USER CODE BEGIN 0 */

e_PosiblesStates state_machine_action(e_PosiblesEvents event) {
    switch (event) {
        case IDLE:
            break;
        case EVENT_ENCODER:
            if (data_snapshot) // DATA snapshot

```

```

        display_value = (display_value == 0) ? 4095 : display_value - 1;
    else
        display_value = (display_value == 4095) ? 0 : display_value + 1;
    break;

case EVENT_SW:
    display_value = 0;
    nextDigit_FSM = 1;
    break;

case EVENT_TIMER_TICK:{
    uint8_t x_val = (vr_x_buffer[0] * 99) / 4095;
    uint8_t y_val = (vry_buffer[0] * 99) / 4095;
    display_value = (y_val * 100) + x_val;
    // Apagar todos los dígitos
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_0, GPIO_PIN_RESET); // userDis1
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_1, GPIO_PIN_RESET); // userDis2
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_0, GPIO_PIN_RESET); // userDis3
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_4, GPIO_PIN_RESET); // userDis4

    switch (nextDigit_FSM) {
        case 1:
            HAL_GPIO_WritePin(GPIOA, GPIO_PIN_0, GPIO_PIN_SET);
            displayNumber(display_value % 10);
            break;
        case 2:
            HAL_GPIO_WritePin(GPIOA, GPIO_PIN_1, GPIO_PIN_SET);
            displayNumber((display_value / 10) % 10);
            break;
        case 3:
            HAL_GPIO_WritePin(GPIOB, GPIO_PIN_0, GPIO_PIN_SET);
            displayNumber((display_value / 100) % 10);
            break;
        case 4:
            HAL_GPIO_WritePin(GPIOA, GPIO_PIN_4, GPIO_PIN_SET);
            displayNumber((display_value / 1000) % 10);
            break;
    }
    nextDigit_FSM = (nextDigit_FSM < 4) ? nextDigit_FSM + 1 : 1;
    break;}

case EVENT_USART: {
    // eco
    int len = snprintf((char*)tx_buffer, sizeof(tx_buffer),
        "Comando recibido: %s\r\n", rx_buffer);
    HAL_UART_Transmit(&huart2, tx_buffer, len, 1000);

    // dispatch
    if (strncmp((char*)rx_buffer,"led=",4)==0)    HandleLEDDelayCmd(rx_buffer+4);
    else if (strncmp((char*)rx_buffer,"fmuestreo=",10)==0)
        HandleSampleFreqCmd(rx_buffer+10);
    else if (strncmp((char*)rx_buffer,"rgb=",4)==0)    HandleRGBCmd(rx_buffer+4);
}

```

```

else if (strncmp((char*)rx_buffer,"status",6)==0) HandleStatusCmd();
else if (strncmp((char*)rx_buffer,"help",4)==0) HandleHelpCmd();
else if (strncmp((char*)rx_buffer, "clear", 5) == 0) HandleClearCmd();
else
    ;

memset(rx_buffer,0,sizeof(rx_buffer));
break;
}return event;
}
}

```

```

void displayNumber(uint8_t digitValue) {
    uint32_t start = DWT->CYCCNT;
    static uint8_t contador_muestras = 0;
    // Apagar todos los segmentos
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_10, GPIO_PIN_RESET); // A
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_10, GPIO_PIN_RESET); // B
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_13, GPIO_PIN_RESET); // C
    HAL_GPIO_WritePin(GPIOC, GPIO_PIN_0, GPIO_PIN_RESET); // D
    HAL_GPIO_WritePin(GPIOC, GPIO_PIN_1, GPIO_PIN_RESET); // E
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_1, GPIO_PIN_RESET); // F
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_5, GPIO_PIN_RESET); // G

```

```

switch (digitValue) {
    case 0:
        HAL_GPIO_WritePin(GPIOA, GPIO_PIN_10, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_10, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_13, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOC, GPIO_PIN_0, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOC, GPIO_PIN_1, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_1, GPIO_PIN_SET);
        break;
    case 1:
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_10, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_13, GPIO_PIN_SET);
        break;
    case 2:
        HAL_GPIO_WritePin(GPIOA, GPIO_PIN_10, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_10, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOC, GPIO_PIN_0, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOC, GPIO_PIN_1, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_5, GPIO_PIN_SET);
        break;
    case 3:
        HAL_GPIO_WritePin(GPIOA, GPIO_PIN_10, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_10, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_13, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOC, GPIO_PIN_0, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_5, GPIO_PIN_SET);
        break;
    case 4:

```

```

    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_10, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_13, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_1, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_5, GPIO_PIN_SET);
    break;
case 5:
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_10, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_13, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOC, GPIO_PIN_0, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_1, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_5, GPIO_PIN_SET);
    break;
case 6:
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_10, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_13, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOC, GPIO_PIN_0, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOC, GPIO_PIN_1, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_1, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_5, GPIO_PIN_SET);
    break;
case 7:
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_10, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_10, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_13, GPIO_PIN_SET);
    break;
case 8:
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_10, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_10, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_13, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOC, GPIO_PIN_0, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOC, GPIO_PIN_1, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_1, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_5, GPIO_PIN_SET);
    break;
case 9:
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_10, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_10, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_13, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_1, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_5, GPIO_PIN_SET);
    break;
}
uint32_t elapsed = DWT->CYCCNT - start;

if (contador_muestras < 10) {
    printf("Ciclos para mostrar digito %d: %lu\n", digitValue, (unsigned long)elapsed);
    contador_muestras++;
}
}
}

void HandleLEDDelayCmd(const char *arg) {
    uint32_t nuevo = atoi(arg);

```

```
// Ajusta TIM2 para el blinky
__HAL_TIM_SET_AUTORELOAD(&htim2, nuevo);
__HAL_TIM_SET_COUNTER(&htim2, 0);
// Feedback
int len = snprintf((char*)tx_buffer, sizeof(tx_buffer),
    "LED delay = %lu ms\r\n", (unsigned long)nuevo);
HAL_UART_Transmit(&huart2, tx_buffer, len, 1000);
}

void HandleSampleFreqCmd(const char *arg) {
    uint32_t fs;
    char option = arg[0];
    // Verifica la opción seleccionada (1, 2, 3, 4)
    switch(option) {
        case '1': fs = 44100; break;
        case '2': fs = 48000; break;
        case '3': fs = 96000; break;
        case '4': fs = 128000; break;
        default: {
            const char *msg = "Opciones válidas para 'fmuestreo=' son: 1, 2, 3, 4\r\n";
            HAL_UART_Transmit(&huart2, (uint8_t*)msg, strlen(msg), 1000);
            return;
        }
    }
    // calcular ARR y PSC
    uint32_t arr = timer_clk / fs - 1;
    uint32_t psc = 0;
    if (arr > 0xFFFF) {
        // escalar PSC si supera 16 bits
        psc = (arr / 0x10000) + 1;
        arr = (timer_clk / (psc+1) / fs) - 1;
    }
    // aplicar a TIM3 y reiniciar
    __HAL_TIM_SET_PRESCALER(&htim3, psc);
    __HAL_TIM_SET_AUTORELOAD(&htim3, arr);
    __HAL_TIM_SET_COUNTER(&htim3, 0);
    HAL_TIM_Base_Stop(&htim3);
    HAL_TIM_Base_Start(&htim3);
    // Feedback
    int len = snprintf((char*)tx_buffer, sizeof(tx_buffer),
        "Sample TIM3 @ %lu Hz (PSC=%lu, ARR=%lu)\r\n",
        (unsigned long)fs, (unsigned long)psc, (unsigned long)arr);
    HAL_UART_Transmit(&huart2, tx_buffer, len, 1000);
}

void HandleRGBCmd(const char *arg) {
    // R, G, B si aparecen en arg
    GPIO_PinState R = (strchr(arg, 'R') ? GPIO_PIN_SET : GPIO_PIN_RESET);
    GPIO_PinState G = (strchr(arg, 'G') ? GPIO_PIN_SET : GPIO_PIN_RESET);
    GPIO_PinState B = (strchr(arg, 'B') ? GPIO_PIN_SET : GPIO_PIN_RESET);
    HAL_GPIO_WritePin(GPIOC, GPIO_PIN_3, R);
}
```



```

HAL_GPIO_WritePin(GPIOC, GPIO_PIN_2, G);
HAL_GPIO_WritePin(GPIOC, GPIO_PIN_13, B);
// Feedback
int len = snprintf((char*)tx_buffer, sizeof(tx_buffer),
    "RGB -> R:%c G:%c B:%c\r\n",
    R==GPIO_PIN_SET?'1':'0',
    G==GPIO_PIN_SET?'1':'0',
    B==GPIO_PIN_SET?'1':'0');
HAL_UART_Transmit(&huart2, tx_buffer, len, 1000);
}

void HandleStatusCmd(void) {
    uint32_t psc = htim3.Init.Prescaler;
    uint32_t arr = htim3.Init.Period;
    float fs = 84000000.0f / ((psc + 1) * (arr + 1));

    int len = snprintf((char*)tx_buffer, sizeof(tx_buffer),
        "Config:\r\nSample TIM3 @ %.2f Hz (PSC=%lu, ARR=%lu)\r\nCanal ADC: 6 y 7\r\n",
        fs, (unsigned long)psc, (unsigned long)arr);
    HAL_UART_Transmit(&huart2, tx_buffer, len, 1000);
}

void HandleClearCmd(void) {
    const char *clear_screen = "\033[2J\033[H"; // Secuencia ANSI para limpiar la pantalla
    HAL_UART_Transmit(&huart2, (uint8_t*)clear_screen, strlen(clear_screen), 1000);
}

void HandleHelpCmd(void) {
    const char *help_msg =
        "\r\n===== AYUDA =====\r\n"
        "\r\nComandos disponibles:\r\n"
        " led=<ms>@ - Cambia la frecuencia del LED Blinky\r\n"
        " fmuestreo=<1|2|3|4>@ - Frecuencia de muestreo del ADC:\r\n"
        "     - 1 -> 44100 Hz\r\n"
        "     - 2 -> 48000 Hz\r\n"
        "     - 3 -> 96000 Hz\r\n"
        "     - 4 -> 128000 Hz\r\n"
        " rgb=<RGB>@ - Control de LED RGB, ej: rgb=RG\r\n"
        " status@ - Mostrar configuración actual\r\n"
        " print@ - Imprimir datos crudos del ADC\r\n"
        " help @ - Mostrar esta ayuda\r\n"
        "===== \r\n";
    HAL_UART_Transmit(&huart2, (uint8_t*)help_msg, strlen(help_msg), 1000);
}

void imprimirBinario(uint8_t valor) {
    for (int i = 7; i >= 0; i--) {
        printf("%d", (valor >> i) & 1);
    }
}

```

```

    }
}

void mostrarOperacion(const char *desc, uint8_t resultado) {
    printf("%s\n", desc);
    printf("Binario: "); imprimirBinario(resultado);
    printf("\nHex: 0x%02X\n", resultado);
    printf("ASCII: %c\n\n", (resultado >= 32 && resultado <= 126) ? resultado : '.');
}

void verificarOperacionesCrucigrama(void) {
    // 1. Caracter ASCII 'B'
    uint8_t op1 = 'B';
    mostrarOperacion("1. ASCII 'B'", op1);

    // 2. ((!!0xC) << 0b11) | (~0x45)
    uint8_t op2 = ((!!0xC) << 0b11) | (~0x45);
    mostrarOperacion("2. ((!!0xC) << 0b11) | (~0x45)", op2);

    // 3. (0b00111100 & 0xAC) << 0x3
    uint8_t op3 = (0b00111100 & 0xAC) << 0x3;
    mostrarOperacion("3. (0b00111100 & 0xAC) << 0x3)", op3);

    // 4. -121
    uint8_t op4 = (uint8_t)(-121);
    mostrarOperacion("4. -121 (uint8_t)", op4);

    // 5. (0xCAFE & 0xFF0) >> 7
    uint16_t val5 = (0xCAFE & 0xFF0) >> 7;
    uint8_t op5 = (uint8_t)(val5 & 0xFF);
    mostrarOperacion("5. (0xCAFE & 0xFF0) >> 7", op5);

    // 6. (1 << 4) + (0x3 << 4) - 10 + (0b11 << 0x6)
    uint8_t op6 = (1 << 4) + (0x3 << 4) - 10 + (0b11 << 6);
    mostrarOperacion("6. (1<<4)+(0x3<<4)-10+(0b11<<6)", op6);

    // 7. ('T' | '5')
    uint8_t op7 = 'T' | '5';
    mostrarOperacion("7. ('T' | '5')", op7);

    // 8. (2921 & 0xD6)
    uint8_t op8 = 2921 & 0xD6;
    mostrarOperacion("8. (2921 & 0xD6)", op8);

    // 9. 01100110 ^ 00101011
    uint8_t a9 = 0b01100110;
    uint8_t b9 = 0b00101011;
    uint8_t op9 = a9 ^ b9;
    mostrarOperacion("9. 01100110 ^ 00101011", op9);
}

/* USER CODE END 0 */

```

```

/**
 * @brief The application entry point.
 * @retval int
 */
int main(void)
{

    /* USER CODE BEGIN 1 */

    /* USER CODE END 1 */

    /* MCU Configuration-----*/

    /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
    HAL_Init();

    /* USER CODE BEGIN Init */
    CoreDebug->DEMCR |= CoreDebug_DEMCR_TRCENA_Msk; // Habilitar DWT
    DWT->CYCCNT = 0; // Reiniciar contador
    DWT->CTRL |= DWT_CTRL_CYCCNTENA_Msk; // Encender contador de ciclos
    /* USER CODE END Init */

    /* Configure the system clock */
    SystemClock_Config();

    /* USER CODE BEGIN SysInit */

    /* USER CODE END SysInit */

    /* Initialize all configured peripherals */
    MX_GPIO_Init();
    MX_DMA_Init();
    MX_USART2_UART_Init();
    MX_TIM2_Init();
    MX_TIM4_Init();
    MX_ADC1_Init();
    MX_TIM3_Init();
    MX_TIM1_Init();
    /* USER CODE BEGIN 2 */
    HAL_TIM_Base_Start_IT(&htim2);
    HAL_TIM_Base_Start_IT(&htim3);
    HAL_TIM_Base_Start_IT(&htim4);
    HAL_UART_Receive_IT(&huart2,&rx_char,1);
    HAL_ADC_Start_DMA(&hadc1, (uint32_t*)adc_buffer, fft_size);
    HAL_TIM_IC_Start_IT(&htim3, TIM_CHANNEL_2);
    HandleSampleFreqCmd(3); //96000 hz muestreo

    verificarOperacionesCrucigrama();
    medirTiempoEnvioUART();
    HandleHelpCmd();

```

```

/* USER CODE END 2 */

/* Infinite loop */
/* USER CODE BEGIN WHILE */
while (1)
{
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
    if (pending_event != IDLE){
        state_machine_action(pending_event);
        pending_event = IDLE;
    }
}
/* USER CODE END 3 */
}

/**
 * @brief System Clock Configuration
 * @retval None
 */
void SystemClock_Config(void)
{
    RCC_OscInitTypeDef RCC_OscInitStruct = {0};
    RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};

    /** Configure the main internal regulator output voltage
    */
    __HAL_RCC_PWR_CLK_ENABLE();
    __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE1);

    /** Initializes the RCC Oscillators according to the specified parameters
    * in the RCC_OscInitTypeDef structure.
    */
    RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI;
    RCC_OscInitStruct.HSISState = RCC_HSI_ON;
    RCC_OscInitStruct.HSICalibrationValue = RCC_HSICALIBRATION_DEFAULT;
    RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
    RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSI;
    RCC_OscInitStruct.PLL.PLLM = 16;
    RCC_OscInitStruct.PLL.PLLN = 336;
    RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV4;
    RCC_OscInitStruct.PLL.PLLQ = 4;
    if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
    {
        Error_Handler();
    }

    /** Initializes the CPU, AHB and APB buses clocks
    */

```

```

RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
                             |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV2;
RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;

if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_2) != HAL_OK)
{
    Error_Handler();
}
}

/**
 * @brief ADC1 Initialization Function
 * @param None
 * @retval None
 */
static void MX_ADC1_Init(void)
{

    /* USER CODE BEGIN ADC1_Init 0 */

    /* USER CODE END ADC1_Init 0 */

    ADC_ChannelConfTypeDef sConfig = {0};

    /* USER CODE BEGIN ADC1_Init 1 */

    /* USER CODE END ADC1_Init 1 */

    /** Configure the global features of the ADC (Clock, Resolution, Data Alignment and number of
    conversion)
    */
    hadc1.Instance = ADC1;
    hadc1.Init.ClockPrescaler = ADC_CLOCK_SYNC_PCLK_DIV4;
    hadc1.Init.Resolution = ADC_RESOLUTION_12B;
    hadc1.Init.ScanConvMode = ENABLE;
    hadc1.Init.ContinuousConvMode = DISABLE;
    hadc1.Init.DiscontinuousConvMode = DISABLE;
    hadc1.Init.ExternalTrigConvEdge = ADC_EXTERNALTRIGCONVEDGE_RISING;
    hadc1.Init.ExternalTrigConv = ADC_EXTERNALTRIGCONV_T3_TRGO;
    hadc1.Init.DataAlign = ADC_DATAALIGN_RIGHT;
    hadc1.Init.NbrOfConversion = 2;
    hadc1.Init.DMAContinuousRequests = ENABLE;
    hadc1.Init.EOCSelection = ADC_EOC_SINGLE_CONV;
    if (HAL_ADC_Init(&hadc1) != HAL_OK)
    {
        Error_Handler();
    }
}

```

```

/** Configure for the selected ADC regular channel its corresponding rank in the sequencer and its
sample time.
*/
sConfig.Channel = ADC_CHANNEL_6;
sConfig.Rank = 1;
sConfig.SamplingTime = ADC_SAMPLETIME_3CYCLES;
if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
{
    Error_Handler();
}

/** Configure for the selected ADC regular channel its corresponding rank in the sequencer and its
sample time.
*/
sConfig.Rank = 2;
if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
{
    Error_Handler();
}
/* USER CODE BEGIN ADC1_Init 2 */

/* USER CODE END ADC1_Init 2 */

}

/**
 * @brief TIM1 Initialization Function
 * @param None
 * @retval None
 */
static void MX_TIM1_Init(void)
{
    /* USER CODE BEGIN TIM1_Init 0 */

    /* USER CODE END TIM1_Init 0 */

    TIM_MasterConfigTypeDef sMasterConfig = {0};
    TIM_OC_InitTypeDef sConfigOC = {0};
    TIM_BreakDeadTimeConfigTypeDef sBreakDeadTimeConfig = {0};

    /* USER CODE BEGIN TIM1_Init 1 */

    /* USER CODE END TIM1_Init 1 */
    htim1.Instance = TIM1;
    htim1.Init.Prescaler = 83;
    htim1.Init.CounterMode = TIM_COUNTERMODE_UP;
    htim1.Init.Period = 255;
    htim1.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
    htim1.Init.RepetitionCounter = 0;
    htim1.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;

```

```

if (HAL_TIM_PWM_Init(&htim1) != HAL_OK)
{
    Error_Handler();
}
sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
if (HAL_TIMEx_MasterConfigSynchronization(&htim1, &sMasterConfig) != HAL_OK)
{
    Error_Handler();
}
sConfigOC.OCMode = TIM_OCMODE_PWM1;
sConfigOC.Pulse = 0;
sConfigOC.OCpolarity = TIM_OCPOLARITY_HIGH;
sConfigOC.OCNPolarity = TIM_OCNPOLARITY_HIGH;
sConfigOC.OCFastMode = TIM_OCFAST_DISABLE;
sConfigOC.OCIdleState = TIM_OCIDLESTATE_RESET;
sConfigOC.OCNIdleState = TIM_OCNIDLESTATE_RESET;
if (HAL_TIM_PWM_ConfigChannel(&htim1, &sConfigOC, TIM_CHANNEL_1) != HAL_OK)
{
    Error_Handler();
}
if (HAL_TIM_PWM_ConfigChannel(&htim1, &sConfigOC, TIM_CHANNEL_2) != HAL_OK)
{
    Error_Handler();
}
if (HAL_TIM_PWM_ConfigChannel(&htim1, &sConfigOC, TIM_CHANNEL_3) != HAL_OK)
{
    Error_Handler();
}
sBreakDeadTimeConfig.OffStateRunMode = TIM_OSSR_DISABLE;
sBreakDeadTimeConfig.OffStateIDLEMode = TIM_OSSI_DISABLE;
sBreakDeadTimeConfig.LockLevel = TIM_LOCKLEVEL_OFF;
sBreakDeadTimeConfig.DeadTime = 0;
sBreakDeadTimeConfig.BreakState = TIM_BREAK_DISABLE;
sBreakDeadTimeConfig.BreakPolarity = TIM_BREAKPOLARITY_HIGH;
sBreakDeadTimeConfig.AutomaticOutput = TIM_AUTOMATICOUTPUT_DISABLE;
if (HAL_TIMEx_ConfigBreakDeadTime(&htim1, &sBreakDeadTimeConfig) != HAL_OK)
{
    Error_Handler();
}
/* USER CODE BEGIN TIM1_Init 2 */

/* USER CODE END TIM1_Init 2 */
HAL_TIM_MspPostInit(&htim1);

}

/**
 * @brief TIM2 Initialization Function
 * @param None
 * @retval None

```

```

*/
static void MX_TIM2_Init(void)
{

/* USER CODE BEGIN TIM2_Init 0 */

/* USER CODE END TIM2_Init 0 */

TIM_ClockConfigTypeDef sClockSourceConfig = {0};
TIM_MasterConfigTypeDef sMasterConfig = {0};

/* USER CODE BEGIN TIM2_Init 1 */

/* USER CODE END TIM2_Init 1 */
htim2.Instance = TIM2;
htim2.Init.Prescaler = 42000-1;
htim2.Init.CounterMode = TIM_COUNTERMODE_UP;
htim2.Init.Period = 500;
htim2.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
htim2.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_ENABLE;
if (HAL_TIM_Base_Init(&htim2) != HAL_OK)
{
    Error_Handler();
}
sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
if (HAL_TIM_ConfigClockSource(&htim2, &sClockSourceConfig) != HAL_OK)
{
    Error_Handler();
}
sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
if (HAL_TIMEx_MasterConfigSynchronization(&htim2, &sMasterConfig) != HAL_OK)
{
    Error_Handler();
}
/* USER CODE BEGIN TIM2_Init 2 */

/* USER CODE END TIM2_Init 2 */

}

/**
 * @brief TIM3 Initialization Function
 * @param None
 * @retval None
 */
static void MX_TIM3_Init(void)
{

/* USER CODE BEGIN TIM3_Init 0 */

```



```

/* USER CODE END TIM3_Init 0 */

TIM_ClockConfigTypeDef sClockSourceConfig = {0};
TIM_MasterConfigTypeDef sMasterConfig = {0};
TIM_IC_InitTypeDef sConfigIC = {0};

/* USER CODE BEGIN TIM3_Init 1 */

/* USER CODE END TIM3_Init 1 */
htim3.Instance = TIM3;
htim3.Init.Prescaler = 0;
htim3.Init.CounterMode = TIM_COUNTERMODE_UP;
htim3.Init.Period = 65535;
htim3.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
htim3.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
if (HAL_TIM_Base_Init(&htim3) != HAL_OK)
{
    Error_Handler();
}
sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
if (HAL_TIM_ConfigClockSource(&htim3, &sClockSourceConfig) != HAL_OK)
{
    Error_Handler();
}
if (HAL_TIM_IC_Init(&htim3) != HAL_OK)
{
    Error_Handler();
}
sMasterConfig.MasterOutputTrigger = TIM_TRGO_UPDATE;
sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
if (HAL_TIMEx_MasterConfigSynchronization(&htim3, &sMasterConfig) != HAL_OK)
{
    Error_Handler();
}
sConfigIC.ICPolarity = TIM_INPUTCHANNELPOLARITY_RISING;
sConfigIC.ICSelection = TIM_ICSELECTION_DIRECTTI;
sConfigIC.ICPrescaler = TIM_ICPSC_DIV1;
sConfigIC.ICFilter = 8;
if (HAL_TIM_IC_ConfigChannel(&htim3, &sConfigIC, TIM_CHANNEL_2) != HAL_OK)
{
    Error_Handler();
}
/* USER CODE BEGIN TIM3_Init 2 */
HAL_TIM_IC_Start_IT(&htim3, TIM_CHANNEL_2);
/* USER CODE END TIM3_Init 2 */

}

/**
 * @brief TIM4 Initialization Function
 * @param None

```

```

* @retval None
*/
static void MX_TIM4_Init(void)
{

    /* USER CODE BEGIN TIM4_Init 0 */

    /* USER CODE END TIM4_Init 0 */

    TIM_ClockConfigTypeDef sClockSourceConfig = {0};
    TIM_MasterConfigTypeDef sMasterConfig = {0};

    /* USER CODE BEGIN TIM4_Init 1 */

    /* USER CODE END TIM4_Init 1 */
    htim4.Instance = TIM4;
    htim4.Init.Prescaler = 840-1;
    htim4.Init.CounterMode = TIM_COUNTERMODE_UP;
    htim4.Init.Period = 200-1;
    htim4.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
    htim4.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
    if (HAL_TIM_Base_Init(&htim4) != HAL_OK)
    {
        Error_Handler();
    }
    sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
    if (HAL_TIM_ConfigClockSource(&htim4, &sClockSourceConfig) != HAL_OK)
    {
        Error_Handler();
    }
    sMasterConfig.MasterOutputTrigger = TIM_TRGO_UPDATE;
    sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
    if (HAL_TIMEx_MasterConfigSynchronization(&htim4, &sMasterConfig) != HAL_OK)
    {
        Error_Handler();
    }
    /* USER CODE BEGIN TIM4_Init 2 */

    /* USER CODE END TIM4_Init 2 */

}

/**
 * @brief USART2 Initialization Function
 * @param None
 * @retval None
 */
static void MX_USART2_UART_Init(void)
{

    /* USER CODE BEGIN USART2_Init 0 */

```

```

/* USER CODE END USART2_Init 0 */

/* USER CODE BEGIN USART2_Init 1 */

/* USER CODE END USART2_Init 1 */
huart2.Instance = USART2;
huart2.Init.BaudRate = 115200;
huart2.Init.WordLength = UART_WORDLENGTH_8B;
huart2.Init.StopBits = UART_STOPBITS_1;
huart2.Init.Parity = UART_PARITY_NONE;
huart2.Init.Mode = UART_MODE_TX_RX;
huart2.Init.HwFlowCtl = UART_HWCONTROL_NONE;
huart2.Init.OverSampling = UART_OVERSAMPLING_16;
if (HAL_UART_Init(&huart2) != HAL_OK)
{
    Error_Handler();
}
/* USER CODE BEGIN USART2_Init 2 */

/* USER CODE END USART2_Init 2 */

}

/**
 * Enable DMA controller clock
 */
static void MX_DMA_Init(void)
{

    /* DMA controller clock enable */
    __HAL_RCC_DMA2_CLK_ENABLE();
    __HAL_RCC_DMA1_CLK_ENABLE();

    /* DMA interrupt init */
    /* DMA1_Stream5_IRQn interrupt configuration */
    HAL_NVIC_SetPriority(DMA1_Stream5_IRQn, 0, 0);
    HAL_NVIC_EnableIRQ(DMA1_Stream5_IRQn);
    /* DMA2_Stream0_IRQn interrupt configuration */
    HAL_NVIC_SetPriority(DMA2_Stream0_IRQn, 0, 0);
    HAL_NVIC_EnableIRQ(DMA2_Stream0_IRQn);

}

/**
 * @brief GPIO Initialization Function
 * @param None
 * @retval None
 */
static void MX_GPIO_Init(void)
{

```

```

GPIO_InitTypeDef GPIO_InitStructure = {0};
/* USER CODE BEGIN MX_GPIO_Init_1 */
/* USER CODE END MX_GPIO_Init_1 */

/* GPIO Ports Clock Enable */
__HAL_RCC_GPIOC_CLK_ENABLE();
__HAL_RCC_GPIOH_CLK_ENABLE();
__HAL_RCC_GPIOA_CLK_ENABLE();
__HAL_RCC_GPIOB_CLK_ENABLE();

/*Configure GPIO pin Output Level */
HAL_GPIO_WritePin(GPIOC, LedB_Pin|segD_Pin|segE_Pin|LedR_Pin
                  |LedG_Pin|segA_Pin, GPIO_PIN_RESET);

/*Configure GPIO pin Output Level */
HAL_GPIO_WritePin(userLed_GPIO_Port, userLed_Pin, GPIO_PIN_SET);

/*Configure GPIO pin Output Level */
HAL_GPIO_WritePin(GPIOA, dis1_Pin|dis2_Pin|dis4_Pin, GPIO_PIN_RESET);

/*Configure GPIO pin Output Level */
HAL_GPIO_WritePin(GPIOB, dis3_Pin|segF_Pin|segB_Pin|segC_Pin
                  |segG_Pin, GPIO_PIN_RESET);

/*Configure GPIO pins : LedB_Pin segD_Pin segE_Pin LedR_Pin
                  LedG_Pin segA_Pin */
GPIO_InitStructure.Pin = LedB_Pin|segD_Pin|segE_Pin|LedR_Pin
                  |LedG_Pin|segA_Pin;
GPIO_InitStructure.Mode = GPIO_MODE_OUTPUT_PP;
GPIO_InitStructure.Pull = GPIO_NOPULL;
GPIO_InitStructure.Speed = GPIO_SPEED_FREQ_LOW;
HAL_GPIO_Init(GPIOC, &GPIO_InitStructure);

/*Configure GPIO pin : userLed_Pin */
GPIO_InitStructure.Pin = userLed_Pin;
GPIO_InitStructure.Mode = GPIO_MODE_OUTPUT_PP;
GPIO_InitStructure.Pull = GPIO_NOPULL;
GPIO_InitStructure.Speed = GPIO_SPEED_FREQ_LOW;
HAL_GPIO_Init(userLed_GPIO_Port, &GPIO_InitStructure);

/*Configure GPIO pins : dis1_Pin dis2_Pin dis4_Pin */
GPIO_InitStructure.Pin = dis1_Pin|dis2_Pin|dis4_Pin;
GPIO_InitStructure.Mode = GPIO_MODE_OUTPUT_PP;
GPIO_InitStructure.Pull = GPIO_NOPULL;
GPIO_InitStructure.Speed = GPIO_SPEED_FREQ_LOW;
HAL_GPIO_Init(GPIOA, &GPIO_InitStructure);

/*Configure GPIO pins : dis3_Pin segF_Pin segB_Pin segC_Pin
                  segG_Pin */
GPIO_InitStructure.Pin = dis3_Pin|segF_Pin|segB_Pin|segC_Pin
                  |segG_Pin;

```

```

GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
GPIO_InitStruct.Pull = GPIO_NOPULL;
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);

/*Configure GPIO pin : userData_Pin */
GPIO_InitStruct.Pin = userData_Pin;
GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
GPIO_InitStruct.Pull = GPIO_NOPULL;
HAL_GPIO_Init(userData_GPIO_Port, &GPIO_InitStruct);

/*Configure GPIO pin : userSw_Pin */
GPIO_InitStruct.Pin = userSw_Pin;
GPIO_InitStruct.Mode = GPIO_MODE_IT_RISING;
GPIO_InitStruct.Pull = GPIO_NOPULL;
HAL_GPIO_Init(userSw_GPIO_Port, &GPIO_InitStruct);

/*Configure GPIO pin : userClk_Pin */
GPIO_InitStruct.Pin = userClk_Pin;
GPIO_InitStruct.Mode = GPIO_MODE_IT_RISING;
GPIO_InitStruct.Pull = GPIO_NOPULL;
HAL_GPIO_Init(userClk_GPIO_Port, &GPIO_InitStruct);

/* EXTI interrupt init*/
HAL_NVIC_SetPriority(EXTI9_5_IRQn, 0, 0);
HAL_NVIC_EnableIRQ(EXTI9_5_IRQn);

HAL_NVIC_SetPriority(EXTI15_10_IRQn, 0, 0);
HAL_NVIC_EnableIRQ(EXTI15_10_IRQn);

/* USER CODE BEGIN MX_GPIO_Init_2 */
/* USER CODE END MX_GPIO_Init_2 */
}

/* USER CODE BEGIN 4 */
void medirTiempoEnvioUART(void) {
    const char *msg = "Hola mundo! - Examen Taller V\n";
    uint32_t start = DWT->CYCCNT;

    printf("%s", msg);

    uint32_t elapsed = DWT->CYCCNT - start;
    printf("Ciclos para enviar mensaje UART: %lu\n", (unsigned long)elapsed);
}
int __io_putchar(int ch) {
    HAL_UART_Transmit(&huart2, (uint8_t*)&ch, 1, HAL_MAX_DELAY);
    return ch;
}
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim){
    if(htim->Instance==TIM2){
        HAL_GPIO_TogglePin(userLed_GPIO_Port,userLed_Pin);
    }
}

```

```

    }else if(htim->Instance==TIM4){
        if (pending_event == IDLE)
            pending_event = EVENT_TIMER_TICK;
    }else if (htim->Instance==TIM3){

    }
}
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
{
    if (GPIO_Pin == userClk_Pin) {
        data_snapshot = HAL_GPIO_ReadPin(userData_GPIO_Port, userData_Pin);
        if (pending_event == IDLE)
            pending_event = EVENT_ENCODER;
    } else if (GPIO_Pin == userSw_Pin) {
        if (pending_event == IDLE)
            pending_event = EVENT_SW;
    }
}
void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart){
    if(huart->Instance == USART2){
        if(rx_index<sizeof(rx_buffer)-1){
            rx_buffer[rx_index++]=rx_char;
        }
        if (rx_char == '@') {
            rx_buffer[rx_index]='\0';
            rx_index=0;
            pending_event = EVENT_USART; //aquí levanta tu bandera
        }
        HAL_UART_Receive_IT(huart, &rx_char,1);
    }
}

/* USER CODE END 4 */

/**
 * @brief This function is executed in case of error occurrence.
 * @retval None
 */
void Error_Handler(void)
{
    /* USER CODE BEGIN Error_Handler_Debug */
    /* User can add his own implementation to report the HAL error return state */
    __disable_irq();
    while (1)
    {
    }
    /* USER CODE END Error_Handler_Debug */
}

#ifdef USE_FULL_ASSERT

```

```
/**
 * @brief Reports the name of the source file and the source line number
 *        where the assert_param error has occurred.
 * @param file: pointer to the source file name
 * @param line: assert_param error line source number
 * @retval None
 */
void assert_failed(uint8_t *file, uint32_t line)
{
    /* USER CODE BEGIN 6 */
    /* User can add his own implementation to report the file name and line number,
       ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */
    /* USER CODE END 6 */
}
#endif /* USE_FULL_ASSERT */
```

Gracias por compartir el código completo. A continuación, se presenta un análisis y recomendaciones para que puedas **probar el control PWM del LED RGB mediante UART** y cumplir lo que se solicita en la imagen compartida:

✓ Estado actual de tu código respecto al LED RGB con PWM

1. PWM configurado correctamente:

- TIM1 se encuentra configurado con `Prescaler = 83` y `ARR = 255`, ideal para generar un ciclo PWM de 8 bits (0-255).
- Los tres canales (CH1, CH2, CH3) están configurados para PWM.

2. Faltante: asignación de canales a pines físicos y activación de PWM:

- Tu código actual **no inicia** los canales PWM de TIM1.
- Tampoco se usan las funciones `__HAL_TIM_SET_COMPARE()` para cambiar el duty cycle.

3. Faltante: actualización de PWM en `HandleRGBCmd()`:

- Actualmente, esta función usa `HAL_GPIO_WritePin()` para encender/apagar pines como si fueran GPIO, no PWM.

4. Pines correctos para salida PWM del RGB:

Según la hoja del STM32F4 (asumiendo una placa tipo Nucleo o STM32F411/STM32F407):

- TIM1_CH1 → PA8 → LED ROJO
 - TIM1_CH2 → PA9 → LED VERDE
 - TIM1_CH3 → PA10 → LED AZUL
- (Verifica si en tu proyecto esos pines están disponibles para conectar tu LED RGB con resistencias limitadoras)*

✓ Cambios sugeridos (resumen técnico)

1. Activar PWM en los tres canales de TIM1:

En `main.c`, dentro de `MX_TIM1_Init()` ya configuraste los canales.

Falta esto en `main()`:

```
c
HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_1); // PA8 - R
HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_2); // PA9 - G
HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_3); // PA10 - B
```

2. Actualizar la función `HandleRGBCmd()` para manejar valores PWM tipo `rgb=120 240 60@`:

Sustituye la función actual por:

```
c
void HandleRGBCmd(const char *arg) {
    uint32_t r, g, b;
    if (sscanf(arg, "%lu %lu %lu", &r, &g, &b) == 3) {
        if (r > 255) r = 255;
        if (g > 255) g = 255;
        if (b > 255) b = 255;

        __HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_1, r); // R
        __HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_2, g); // G
        __HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_3, b); // B

        int len = snprintf((char*)tx_buffer, sizeof(tx_buffer),
                           "RGB PWM -> R=%lu, G=%lu, B=%lu\r\n", r, g, b);
        HAL_UART_Transmit(&huart2, tx_buffer, len, 1000);
    } else {
        const char *msg = "Formato inválido. Usa: rgb=R G B@\r\n";
        HAL_UART_Transmit(&huart2, (uint8_t*)msg, strlen(msg), 1000);
    }
}
```

3. Verifica conexiones físicas del LED RGB:

- R (ánodo/cátodo) conectado a PA8 con resistencia de $\sim 220\Omega$
- G \rightarrow PA9
- B \rightarrow PA10
- Si tu LED es cátodo común, conecta el pin común a GND.

Comando UART de prueba

Envía desde tu terminal UART:

```
text

rgb=255 0 0@
```

\rightarrow LED rojo al máximo

\rightarrow Luego prueba:

text

```
rgb=0 255 0@
rgb=0 0 255@
rgb=128 128 0@
```

Conclusión

Ya puedes **probar tu código** con PWM en los pines PA8, PA9, y PA10 si conectas el LED RGB a esos pines y haces los ajustes anteriores:

- ☒ PWM está inicializado
- ☐ Faltaba iniciar `HAL_TIM_PWM_Start()`
- ☐ Faltaba usar `__HAL_TIM_SET_COMPARE()` para cambiar el duty
- ☒ Puedes enviar comandos como `rgb=R G B@` desde UART

Si deseas, te puedo ayudar a hacer un pequeño menú gráfico por UART o permitir gradientes automáticos de colores. ¿Te gustaría eso?



```
el rgb funciona pero el joystick lo muevo y no salen los valores en el display que me falta : /* USER
CODE BEGIN Header */
/**
*****
* @file      : main.c
* @brief     : Main program body
*****
* @attention
*
* Copyright (c) 2025 STMicroelectronics.
* All rights reserved.
*
* This software is licensed under terms that can be found in the LICENSE file
* in the root directory of this software component.
* If no LICENSE file comes with this software, it is provided AS-IS.
*
*****
*/
/* USER CODE END Header */
/* Includes -----*/
#include "main.h"

/* Private includes -----*/
/* USER CODE BEGIN Includes */
#include "fsm.h"
#include "stdint.h"
#include "stdio.h"
#include "string.h"
```

```
#include <stdlib.h>
/* USER CODE END Includes */

/* Private typedef -----*/
/* USER CODE BEGIN PTD */

/* USER CODE END PTD */

/* Private define -----*/
/* USER CODE BEGIN PD */

/* USER CODE END PD */

/* Private macro -----*/
/* USER CODE BEGIN PM */

/* USER CODE END PM */

/* Private variables -----*/
ADC_HandleTypeDef hadc1;
DMA_HandleTypeDef hdma_adc1;

TIM_HandleTypeDef htim1;
TIM_HandleTypeDef htim2;
TIM_HandleTypeDef htim3;
TIM_HandleTypeDef htim4;
DMA_HandleTypeDef hdma_tim3_ch2;

UART_HandleTypeDef huart2;

/* USER CODE BEGIN PV */
const uint32_t timer_clk = 84000000UL;
static uint16_t fft_size = 1024;
static uint16_t display_value = 0;
static uint8_t nextDigit_FSM = 1;
static uint8_t tx_buffer[256] = {0};
#define FREQ_BUFFER_SIZE 16
float freq_buffer[FREQ_BUFFER_SIZE] = {0};
#define NUM_MUESTRAS 1024
uint16_t adc_buffer[2 * NUM_MUESTRAS];
uint16_t vrx_buffer[NUM_MUESTRAS];
uint16_t vry_buffer[NUM_MUESTRAS];

char rx_buffer[128] = {0};
uint8_t rx_char = 0;
uint8_t rx_index=0;
volatile e_PosiblesEvents pending_event = IDLE;
volatile uint8_t data_snapshot = 0;

/* USER CODE END PV */
```

```

/* Private function prototypes -----*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_DMA_Init(void);
static void MX_USART2_UART_Init(void);
static void MX_TIM2_Init(void);
static void MX_TIM4_Init(void);
static void MX_ADC1_Init(void);
static void MX_TIM3_Init(void);
static void MX_TIM1_Init(void);
/* USER CODE BEGIN PFP */
void displayNumber(uint8_t digitValue);
void HandleLEDDelayCmd(const char *arg);
void HandleSampleFreqCmd(const char *arg);
void HandleRGBCmd(const char *arg);
void HandlePWMFreqCmd(const char *arg);
void HandleUnknownCmd(void);
void HandleStatusCmd(void);
void HandlePrintADC(void);
void HandleClearCmd(void);
/* USER CODE END PFP */

/* Private user code -----*/
/* USER CODE BEGIN 0 */

e_PosiblesStates state_machine_action(e_PosiblesEvents event) {
    switch (event) {
        case IDLE:
            break;
        case EVENT_ENCODER:
            if (data_snapshot) // DATA snapshot
                display_value = (display_value == 0) ? 4095 : display_value - 1;
            else
                display_value = (display_value == 4095) ? 0 : display_value + 1;
            break;

        case EVENT_SW:
            display_value = 0;
            nextDigit_FSM = 1;
            break;

        case EVENT_TIMER_TICK:{
            uint8_t x_val = (vr_x_buffer[0] * 99) / 4095;
            uint8_t y_val = (vry_buffer[0] * 99) / 4095;
            display_value = (y_val * 100) + x_val;
            // Apagar todos los dígitos
            HAL_GPIO_WritePin(GPIOA, GPIO_PIN_0, GPIO_PIN_RESET); // userDis1
            HAL_GPIO_WritePin(GPIOA, GPIO_PIN_1, GPIO_PIN_RESET); // userDis2
            HAL_GPIO_WritePin(GPIOB, GPIO_PIN_0, GPIO_PIN_RESET); // userDis3
            HAL_GPIO_WritePin(GPIOA, GPIO_PIN_4, GPIO_PIN_RESET); // userDis4
        }
    }
}

```

```

switch (nextDigit_FSM) {
    case 1:
        HAL_GPIO_WritePin(GPIOA, GPIO_PIN_0, GPIO_PIN_SET);
        displayNumber(display_value % 10);
        break;
    case 2:
        HAL_GPIO_WritePin(GPIOA, GPIO_PIN_1, GPIO_PIN_SET);
        displayNumber((display_value / 10) % 10);
        break;
    case 3:
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_0, GPIO_PIN_SET);
        displayNumber((display_value / 100) % 10);
        break;
    case 4:
        HAL_GPIO_WritePin(GPIOA, GPIO_PIN_4, GPIO_PIN_SET);
        displayNumber((display_value / 1000) % 10);
        break;
}
nextDigit_FSM = (nextDigit_FSM < 4) ? nextDigit_FSM + 1 : 1;
break;}
case EVENT_USART: {
    // eco
    int len = snprintf((char*)tx_buffer, sizeof(tx_buffer),
        "Comando recibido: %s\r\n", rx_buffer);
    HAL_UART_Transmit(&huart2, tx_buffer, len, 1000);

    // dispatch
    if (strncmp((char*)rx_buffer,"led=",4)==0) HandleLEDDelayCmd(rx_buffer+4);
    else if (strncmp((char*)rx_buffer,"fmuestreo=",10)==0)
        HandleSampleFreqCmd(rx_buffer+10);
    else if (strncmp((char*)rx_buffer,"rgb=",4)==0) HandleRGBCmd(rx_buffer+4);
    else if (strncmp((char*)rx_buffer,"status",6)==0) HandleStatusCmd();
    else if (strncmp((char*)rx_buffer,"help",4)==0) HandleHelpCmd();
    else if (strncmp((char*)rx_buffer, "clear", 5) == 0) HandleClearCmd();
    else
        ;

    memset(rx_buffer,0,sizeof(rx_buffer));
    break;
}return event;
}
}

```

```

void displayNumber(uint8_t digitValue) {
    uint32_t start = DWT->CYCCNT;
    static uint8_t contador_muestras = 0;
    // Apagar todos los segmentos
    HAL_GPIO_WritePin(GPIOC, GPIO_PIN_4, GPIO_PIN_RESET); // A
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_10, GPIO_PIN_RESET); // B
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_13, GPIO_PIN_RESET); // C
    HAL_GPIO_WritePin(GPIOC, GPIO_PIN_0, GPIO_PIN_RESET); // D
    HAL_GPIO_WritePin(GPIOC, GPIO_PIN_1, GPIO_PIN_RESET); // E

```

```

HAL_GPIO_WritePin(GPIOB, GPIO_PIN_1, GPIO_PIN_RESET); // F
HAL_GPIO_WritePin(GPIOB, GPIO_PIN_5, GPIO_PIN_RESET); // G

switch (digitValue) {
    case 0:
        HAL_GPIO_WritePin(GPIOC, GPIO_PIN_4, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_10, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_13, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOC, GPIO_PIN_0, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOC, GPIO_PIN_1, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_1, GPIO_PIN_SET);
        break;
    case 1:
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_10, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_13, GPIO_PIN_SET);
        break;
    case 2:
        HAL_GPIO_WritePin(GPIOC, GPIO_PIN_4, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_10, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOC, GPIO_PIN_0, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOC, GPIO_PIN_1, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_5, GPIO_PIN_SET);
        break;
    case 3:
        HAL_GPIO_WritePin(GPIOC, GPIO_PIN_4, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_10, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_13, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOC, GPIO_PIN_0, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_5, GPIO_PIN_SET);
        break;
    case 4:
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_10, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_13, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_1, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_5, GPIO_PIN_SET);
        break;
    case 5:
        HAL_GPIO_WritePin(GPIOC, GPIO_PIN_4, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_13, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOC, GPIO_PIN_0, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_1, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_5, GPIO_PIN_SET);
        break;
    case 6:
        HAL_GPIO_WritePin(GPIOC, GPIO_PIN_4, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_13, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOC, GPIO_PIN_0, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOC, GPIO_PIN_1, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_1, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_5, GPIO_PIN_SET);
        break;
}

```

```

case 7:
    HAL_GPIO_WritePin(GPIOC, GPIO_PIN_4, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_10, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_13, GPIO_PIN_SET);
    break;
case 8:
    HAL_GPIO_WritePin(GPIOC, GPIO_PIN_4, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_10, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_13, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOC, GPIO_PIN_0, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOC, GPIO_PIN_1, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_1, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_5, GPIO_PIN_SET);
    break;
case 9:
    HAL_GPIO_WritePin(GPIOC, GPIO_PIN_4, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_10, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_13, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_1, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_5, GPIO_PIN_SET);
    break;
}
uint32_t elapsed = DWT->CYCCNT - start;

if (contador_muestras < 10) {
    printf("Ciclos para mostrar digito %d: %lu\n", digitValue, (unsigned long)elapsed);
    contador_muestras++;
}
}

void HandleLEDDelayCmd(const char *arg) {
    uint32_t nuevo = atoi(arg);
    // Ajusta TIM2 para el blinky
    __HAL_TIM_SET_AUTORELOAD(&htim2, nuevo);
    __HAL_TIM_SET_COUNTER(&htim2, 0);
    // Feedback
    int len = snprintf((char*)tx_buffer, sizeof(tx_buffer),
        "LED delay = %lu ms\r\n", (unsigned long)nuevo);
    HAL_UART_Transmit(&huart2, tx_buffer, len, 1000);
}

void HandleSampleFreqCmd(const char *arg) {
    uint32_t fs;
    char option = arg[0];
    // Verifica la opción seleccionada (1, 2, 3, 4)
    switch(option) {
        case '1': fs = 44100; break;
        case '2': fs = 48000; break;
        case '3': fs = 96000; break;
        case '4': fs = 128000; break;
        default: {
            const char *msg = "Opciones válidas para 'fmuestreo=' son: 1, 2, 3, 4\r\n";

```

```

        HAL_UART_Transmit(&huart2, (uint8_t*)msg, strlen(msg), 1000);
        return;
    }
}

// calcular ARR y PSC
uint32_t arr = timer_clk / fs - 1;
uint32_t psc = 0;
if (arr > 0xFFFF) {
    // escalar PSC si supera 16 bits
    psc = (arr / 0x10000) + 1;
    arr = (timer_clk / (psc+1) / fs) - 1;
}
// aplicar a TIM3 y reiniciar
__HAL_TIM_SET_PRESCALER(&htim3, psc);
__HAL_TIM_SET_AUTORELOAD(&htim3, arr);
__HAL_TIM_SET_COUNTER(&htim3, 0);
HAL_TIM_Base_Stop(&htim3);
HAL_TIM_Base_Start(&htim3);
// Feedback
int len = snprintf((char*)tx_buffer, sizeof(tx_buffer),
    "Sample TIM3 @ %lu Hz (PSC=%lu, ARR=%lu)\r\n",
    (unsigned long)fs, (unsigned long)psc, (unsigned long)arr);
HAL_UART_Transmit(&huart2, tx_buffer, len, 1000);
}

void HandleRGBCmd(const char *arg) {
    uint32_t r, g, b;
    if (sscanf(arg, "%lu %lu %lu", &r, &g, &b) == 3) {
        if (r > 255) r = 255;
        if (g > 255) g = 255;
        if (b > 255) b = 255;

        __HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_1, g); // G
        __HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_2, b); // B
        __HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_3, r); // R

        int len = snprintf((char*)tx_buffer, sizeof(tx_buffer),
            "RGB PWM -> R=%lu, G=%lu, B=%lu\r\n", r, g, b);
        HAL_UART_Transmit(&huart2, tx_buffer, len, 1000);
    } else {
        const char *msg = "Formato inválido. Usa: rgb=R G B@\r\n";
        HAL_UART_Transmit(&huart2, (uint8_t*)msg, strlen(msg), 1000);
    }
}

void HandleStatusCmd(void) {
    uint32_t psc = htim3.Init.Prescaler;
    uint32_t arr = htim3.Init.Period;
    float fs = 84000000.0f / ((psc + 1) * (arr + 1));

    int len = snprintf((char*)tx_buffer, sizeof(tx_buffer),

```

```

        "Config:\r\nSample TIM3 @ %.2f Hz (PSC=%lu, ARR=%lu)\r\nCanal ADC: 6 y 7\r\n",
        fs, (unsigned long)psc, (unsigned long)arr);
    HAL_UART_Transmit(&huart2, tx_buffer, len, 1000);
}

```

```

void HandleClearCmd(void) {
    const char *clear_screen = "\033[2J\033[H"; // Secuencia ANSI para limpiar la pantalla
    HAL_UART_Transmit(&huart2, (uint8_t*)clear_screen, strlen(clear_screen), 1000);
}

```

```

void HandleHelpCmd(void) {
    const char *help_msg =
        "\r\n===== AYUDA =====\r\n"
        "\r\nComandos disponibles:\r\n"
        " led=<ms>@    - Cambia la frecuencia del LED Blinky\r\n"
        " fmuestreo=<1|2|3|4>@ - Frecuencia de muestreo del ADC:\r\n"
        "                - 1 -> 44100 Hz\r\n"
        "                - 2 -> 48000 Hz\r\n"
        "                - 3 -> 96000 Hz\r\n"
        "                - 4 -> 128000 Hz\r\n"
        " rgb=<RGB>@    - Control de LED RGB, ej: rgb=RG\r\n"
        "  status@     - Mostrar configuración actual\r\n"
        " print@       - Imprimir datos crudos del ADC\r\n"
        " help @       - Mostrar esta ayuda\r\n"
        "===== \r\n";
    HAL_UART_Transmit(&huart2, (uint8_t*)help_msg, strlen(help_msg), 1000);
}

void imprimirBinario(uint8_t valor) {
    for (int i = 7; i >= 0; i--) {
        printf("%d", (valor >> i) & 1);
    }
}

```

```

void mostrarOperacion(const char *desc, uint8_t resultado) {
    printf("%s\n", desc);
    printf("Binario: "); imprimirBinario(resultado);
    printf("\nHex: 0x%02X\n", resultado);
    printf("ASCII: %c\n\n", (resultado >= 32 && resultado <= 126) ? resultado : '.');
}

```

```

void verificarOperacionesCrucigrama(void) {
    // 1. Caracter ASCII 'B'
    uint8_t op1 = 'B';
    mostrarOperacion("1. ASCII 'B'", op1);

    // 2. ((!!0xC) << 0b11) | (~0x45)
    uint8_t op2 = ((!!0xC) << 0b11) | (~0x45);
}

```



```

mostrarOperacion("2. ((!!0xC) << 0b11) | (~0x45)", op2);

// 3. ((0b00111100 & 0xAC) << 0x3)
uint8_t op3 = (0b00111100 & 0xAC) << 0x3;
mostrarOperacion("3. ((0b00111100 & 0xAC) << 0x3)", op3);

// 4. -121
uint8_t op4 = (uint8_t)(-121);
mostrarOperacion("4. -121 (uint8_t)", op4);

// 5. (0xCAFE & 0xFF0) >> 7
uint16_t val5 = (0xCAFE & 0xFF0) >> 7;
uint8_t op5 = (uint8_t)(val5 & 0xFF);
mostrarOperacion("5. (0xCAFE & 0xFF0) >> 7", op5);

// 6. (1 << 4) + (0x3 << 4) - 10 + (0b11 << 0x6)
uint8_t op6 = (1 << 4) + (0x3 << 4) - 10 + (0b11 << 6);
mostrarOperacion("6. (1<<4)+(0x3<<4)-10+(0b11<<6)", op6);

// 7. ('T' | '5')
uint8_t op7 = 'T' | '5';
mostrarOperacion("7. ('T' | '5')", op7);

// 8. (2921 & 0xD6)
uint8_t op8 = 2921 & 0xD6;
mostrarOperacion("8. (2921 & 0xD6)", op8);

// 9. 01100110 ^ 00101011
uint8_t a9 = 0b01100110;
uint8_t b9 = 0b00101011;
uint8_t op9 = a9 ^ b9;
mostrarOperacion("9. 01100110 ^ 00101011", op9);
}
/* USER CODE END 0 */

/**
 * @brief The application entry point.
 * @retval int
 */
int main(void)
{

/* USER CODE BEGIN 1 */

/* USER CODE END 1 */

/* MCU Configuration-----*/

/* Reset of all peripherals, Initializes the Flash interface and the Systick. */
HAL_Init();

```

```

/* USER CODE BEGIN Init */
    CoreDebug->DEMCR |= CoreDebug_DEMCR_TRCENA_Msk; // Habilitar DWT
    DWT->CYCCNT = 0;                                // Reiniciar contador
    DWT->CTRL |= DWT_CTRL_CYCCNTENA_Msk;           // Encender contador de ciclos
/* USER CODE END Init */

/* Configure the system clock */
SystemClock_Config();

/* USER CODE BEGIN SysInit */

/* USER CODE END SysInit */

/* Initialize all configured peripherals */
MX_GPIO_Init();
MX_DMA_Init();
MX_USART2_UART_Init();
MX_TIM2_Init();
MX_TIM4_Init();
MX_ADC1_Init();
MX_TIM3_Init();
MX_TIM1_Init();
/* USER CODE BEGIN 2 */
HAL_TIM_Base_Start_IT(&htim2);
HAL_TIM_Base_Start_IT(&htim3);
HAL_TIM_Base_Start_IT(&htim4);
HAL_UART_Receive_IT(&huart2,&rx_char,1);
HAL_ADC_Start_DMA(&hadc1, (uint32_t*)adc_buffer, fft_size);
HAL_TIM_IC_Start_IT(&htim3, TIM_CHANNEL_2);
HandleSampleFreqCmd(3); //96000 hz muestreo

verificarOperacionesCrucigrama();
medirTiempoEnvioUART();
HandleHelpCmd();

/* USER CODE END 2 */

/* Infinite loop */
/* USER CODE BEGIN WHILE */
while (1)
{
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
    if (pending_event != IDLE){
        state_machine_action(pending_event);
        pending_event = IDLE;
    }
}
/* USER CODE END 3 */
}

```

```

/**
 * @brief System Clock Configuration
 * @retval None
 */
void SystemClock_Config(void)
{
    RCC_OscInitTypeDef RCC_OscInitStruct = {0};
    RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};

    /** Configure the main internal regulator output voltage
    */
    __HAL_RCC_PWR_CLK_ENABLE();
    __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE1);

    /** Initializes the RCC Oscillators according to the specified parameters
    * in the RCC_OscInitTypeDef structure.
    */
    RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI;
    RCC_OscInitStruct.HSIState = RCC_HSI_ON;
    RCC_OscInitStruct.HSICalibrationValue = RCC_HSICALIBRATION_DEFAULT;
    RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
    RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSI;
    RCC_OscInitStruct.PLL.PLLM = 16;
    RCC_OscInitStruct.PLL.PLLN = 336;
    RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV4;
    RCC_OscInitStruct.PLL.PLLQ = 4;
    if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
    {
        Error_Handler();
    }

    /** Initializes the CPU, AHB and APB buses clocks
    */
    RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
                                |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
    RCC_ClkInitStruct.SYSClkSource = RCC_SYSClkSOURCE_PLLCLK;
    RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSClk_DIV1;
    RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV2;
    RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;

    if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_2) != HAL_OK)
    {
        Error_Handler();
    }
}

/**
 * @brief ADC1 Initialization Function
 * @param None
 * @retval None

```

```

*/
static void MX_ADC1_Init(void)
{

/* USER CODE BEGIN ADC1_Init 0 */

/* USER CODE END ADC1_Init 0 */

ADC_ChannelConfTypeDef sConfig = {0};

/* USER CODE BEGIN ADC1_Init 1 */

/* USER CODE END ADC1_Init 1 */

/** Configure the global features of the ADC (Clock, Resolution, Data Alignment and number of
conversion)
*/
hadc1.Instance = ADC1;
hadc1.Init.ClockPrescaler = ADC_CLOCK_SYNC_PCLK_DIV4;
hadc1.Init.Resolution = ADC_RESOLUTION_12B;
hadc1.Init.ScanConvMode = ENABLE;
hadc1.Init.ContinuousConvMode = DISABLE;
hadc1.Init.DiscontinuousConvMode = DISABLE;
hadc1.Init.ExternalTrigConvEdge = ADC_EXTERNALTRIGCONVEDGE_RISING;
hadc1.Init.ExternalTrigConv = ADC_EXTERNALTRIGCONV_T3_TRGO;
hadc1.Init.DataAlign = ADC_DATAALIGN_RIGHT;
hadc1.Init.NbrOfConversion = 2;
hadc1.Init.DMAContinuousRequests = ENABLE;
hadc1.Init.EOCSelection = ADC_EOC_SINGLE_CONV;
if (HAL_ADC_Init(&hadc1) != HAL_OK)
{
Error_Handler();
}

/** Configure for the selected ADC regular channel its corresponding rank in the sequencer and its
sample time.
*/
sConfig.Channel = ADC_CHANNEL_6;
sConfig.Rank = 1;
sConfig.SamplingTime = ADC_SAMPLETIME_3CYCLES;
if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
{
Error_Handler();
}

/** Configure for the selected ADC regular channel its corresponding rank in the sequencer and its
sample time.
*/
sConfig.Rank = 2;
if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
{

```

```

    Error_Handler();
}
/* USER CODE BEGIN ADC1_Init 2 */

/* USER CODE END ADC1_Init 2 */

}

/**
 * @brief TIM1 Initialization Function
 * @param None
 * @retval None
 */
static void MX_TIM1_Init(void)
{

/* USER CODE BEGIN TIM1_Init 0 */

/* USER CODE END TIM1_Init 0 */

TIM_MasterConfigTypeDef sMasterConfig = {0};
TIM_OC_InitTypeDef sConfigOC = {0};
TIM_BreakDeadTimeConfigTypeDef sBreakDeadTimeConfig = {0};

/* USER CODE BEGIN TIM1_Init 1 */

/* USER CODE END TIM1_Init 1 */
htim1.Instance = TIM1;
htim1.Init.Prescaler = 83;
htim1.Init.CounterMode = TIM_COUNTERMODE_UP;
htim1.Init.Period = 255;
htim1.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
htim1.Init.RepetitionCounter = 0;
htim1.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
if (HAL_TIM_PWM_Init(&htim1) != HAL_OK)
{
    Error_Handler();
}
sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
if (HAL_TIMEx_MasterConfigSynchronization(&htim1, &sMasterConfig) != HAL_OK)
{
    Error_Handler();
}
sConfigOC.OCMode = TIM_OCMODE_PWM1;
sConfigOC.Pulse = 0;
sConfigOC.OCpolarity = TIM_OCPOLARITY_HIGH;
sConfigOC.OCNPolarity = TIM_OCNPOLARITY_HIGH;
sConfigOC.OCFastMode = TIM_OCFAST_DISABLE;
sConfigOC.OCIdleState = TIM_OCIDLESTATE_RESET;
sConfigOC.OCNIdleState = TIM_OCNIDLESTATE_RESET;

```

```

if (HAL_TIM_PWM_ConfigChannel(&htim1, &sConfigOC, TIM_CHANNEL_1) != HAL_OK)
{
    Error_Handler();
}
if (HAL_TIM_PWM_ConfigChannel(&htim1, &sConfigOC, TIM_CHANNEL_2) != HAL_OK)
{
    Error_Handler();
}
if (HAL_TIM_PWM_ConfigChannel(&htim1, &sConfigOC, TIM_CHANNEL_3) != HAL_OK)
{
    Error_Handler();
}
sBreakDeadTimeConfig.OffStateRunMode = TIM_OSSR_DISABLE;
sBreakDeadTimeConfig.OffStateIDLEMode = TIM_OSSI_DISABLE;
sBreakDeadTimeConfig.LockLevel = TIM_LOCKLEVEL_OFF;
sBreakDeadTimeConfig.DeadTime = 0;
sBreakDeadTimeConfig.BreakState = TIM_BREAK_DISABLE;
sBreakDeadTimeConfig.BreakPolarity = TIM_BREAKPOLARITY_HIGH;
sBreakDeadTimeConfig.AutomaticOutput = TIM_AUTOMATICOUTPUT_DISABLE;
if (HAL_TIMEx_ConfigBreakDeadTime(&htim1, &sBreakDeadTimeConfig) != HAL_OK)
{
    Error_Handler();
}
/* USER CODE BEGIN TIM1_Init 2 */
HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_1); // PA8 - g
HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_2); // PA9 - B
HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_3); // PA10 - R
/* USER CODE END TIM1_Init 2 */
HAL_TIM_MspPostInit(&htim1);

}

/**
 * @brief TIM2 Initialization Function
 * @param None
 * @retval None
 */
static void MX_TIM2_Init(void)
{
    /* USER CODE BEGIN TIM2_Init 0 */

    /* USER CODE END TIM2_Init 0 */

    TIM_ClockConfigTypeDef sClockSourceConfig = {0};
    TIM_MasterConfigTypeDef sMasterConfig = {0};

    /* USER CODE BEGIN TIM2_Init 1 */

    /* USER CODE END TIM2_Init 1 */
    htim2.Instance = TIM2;

```

```

htim2.Init.Prescaler = 42000-1;
htim2.Init.CounterMode = TIM_COUNTERMODE_UP;
htim2.Init.Period = 500;
htim2.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
htim2.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_ENABLE;
if (HAL_TIM_Base_Init(&htim2) != HAL_OK)
{
    Error_Handler();
}
sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
if (HAL_TIM_ConfigClockSource(&htim2, &sClockSourceConfig) != HAL_OK)
{
    Error_Handler();
}
sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
if (HAL_TIMEx_MasterConfigSynchronization(&htim2, &sMasterConfig) != HAL_OK)
{
    Error_Handler();
}
/* USER CODE BEGIN TIM2_Init 2 */

/* USER CODE END TIM2_Init 2 */

}

/**
 * @brief TIM3 Initialization Function
 * @param None
 * @retval None
 */
static void MX_TIM3_Init(void)
{
    /* USER CODE BEGIN TIM3_Init 0 */

    /* USER CODE END TIM3_Init 0 */

    TIM_ClockConfigTypeDef sClockSourceConfig = {0};
    TIM_MasterConfigTypeDef sMasterConfig = {0};
    TIM_IC_InitTypeDef sConfigIC = {0};

    /* USER CODE BEGIN TIM3_Init 1 */

    /* USER CODE END TIM3_Init 1 */
    htim3.Instance = TIM3;
    htim3.Init.Prescaler = 0;
    htim3.Init.CounterMode = TIM_COUNTERMODE_UP;
    htim3.Init.Period = 65535;
    htim3.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
    htim3.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;

```

```

if (HAL_TIM_Base_Init(&htim3) != HAL_OK)
{
    Error_Handler();
}
sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
if (HAL_TIM_ConfigClockSource(&htim3, &sClockSourceConfig) != HAL_OK)
{
    Error_Handler();
}
if (HAL_TIM_IC_Init(&htim3) != HAL_OK)
{
    Error_Handler();
}
sMasterConfig.MasterOutputTrigger = TIM_TRGO_UPDATE;
sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
if (HAL_TIMEx_MasterConfigSynchronization(&htim3, &sMasterConfig) != HAL_OK)
{
    Error_Handler();
}
sConfigIC.ICPolarity = TIM_INPUTCHANNELPOLARITY_RISING;
sConfigIC.ICSelection = TIM_ICSELECTION_DIRECTTI;
sConfigIC.ICPrescaler = TIM_ICPSC_DIV1;
sConfigIC.ICFilter = 8;
if (HAL_TIM_IC_ConfigChannel(&htim3, &sConfigIC, TIM_CHANNEL_2) != HAL_OK)
{
    Error_Handler();
}
/* USER CODE BEGIN TIM3_Init 2 */
HAL_TIM_IC_Start_IT(&htim3, TIM_CHANNEL_2);
/* USER CODE END TIM3_Init 2 */

}

/**
 * @brief TIM4 Initialization Function
 * @param None
 * @retval None
 */
static void MX_TIM4_Init(void)
{

    /* USER CODE BEGIN TIM4_Init 0 */

    /* USER CODE END TIM4_Init 0 */

    TIM_ClockConfigTypeDef sClockSourceConfig = {0};
    TIM_MasterConfigTypeDef sMasterConfig = {0};

    /* USER CODE BEGIN TIM4_Init 1 */

    /* USER CODE END TIM4_Init 1 */

```



```

htim4.Instance = TIM4;
htim4.Init.Prescaler = 840-1;
htim4.Init.CounterMode = TIM_COUNTERMODE_UP;
htim4.Init.Period = 200-1;
htim4.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
htim4.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
if (HAL_TIM_Base_Init(&htim4) != HAL_OK)
{
    Error_Handler();
}
sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
if (HAL_TIM_ConfigClockSource(&htim4, &sClockSourceConfig) != HAL_OK)
{
    Error_Handler();
}
sMasterConfig.MasterOutputTrigger = TIM_TRGO_UPDATE;
sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
if (HAL_TIMEx_MasterConfigSynchronization(&htim4, &sMasterConfig) != HAL_OK)
{
    Error_Handler();
}
/* USER CODE BEGIN TIM4_Init 2 */

/* USER CODE END TIM4_Init 2 */

}

/**
 * @brief USART2 Initialization Function
 * @param None
 * @retval None
 */
static void MX_USART2_UART_Init(void)
{
    /* USER CODE BEGIN USART2_Init 0 */

    /* USER CODE END USART2_Init 0 */

    /* USER CODE BEGIN USART2_Init 1 */

    /* USER CODE END USART2_Init 1 */
    huart2.Instance = USART2;
    huart2.Init.BaudRate = 115200;
    huart2.Init.WordLength = UART_WORDLENGTH_8B;
    huart2.Init.StopBits = UART_STOPBITS_1;
    huart2.Init.Parity = UART_PARITY_NONE;
    huart2.Init.Mode = UART_MODE_TX_RX;
    huart2.Init.HwFlowCtl = UART_HWCONTROL_NONE;
    huart2.Init.OverSampling = UART_OVERSAMPLING_16;
    if (HAL_UART_Init(&huart2) != HAL_OK)

```

```

{
    Error_Handler();
}
/* USER CODE BEGIN USART2_Init 2 */

/* USER CODE END USART2_Init 2 */

}

/**
 * Enable DMA controller clock
 */
static void MX_DMA_Init(void)
{

    /* DMA controller clock enable */
    __HAL_RCC_DMA2_CLK_ENABLE();
    __HAL_RCC_DMA1_CLK_ENABLE();

    /* DMA interrupt init */
    /* DMA1_Stream5_IRQn interrupt configuration */
    HAL_NVIC_SetPriority(DMA1_Stream5_IRQn, 0, 0);
    HAL_NVIC_EnableIRQ(DMA1_Stream5_IRQn);
    /* DMA2_Stream0_IRQn interrupt configuration */
    HAL_NVIC_SetPriority(DMA2_Stream0_IRQn, 0, 0);
    HAL_NVIC_EnableIRQ(DMA2_Stream0_IRQn);

}

/**
 * @brief GPIO Initialization Function
 * @param None
 * @retval None
 */
static void MX_GPIO_Init(void)
{
    GPIO_InitTypeDef GPIO_InitStruct = {0};
    /* USER CODE BEGIN MX_GPIO_Init_1 */
    /* USER CODE END MX_GPIO_Init_1 */

    /* GPIO Ports Clock Enable */
    __HAL_RCC_GPIOC_CLK_ENABLE();
    __HAL_RCC_GPIOH_CLK_ENABLE();
    __HAL_RCC_GPIOA_CLK_ENABLE();
    __HAL_RCC_GPIOB_CLK_ENABLE();

    /*Configure GPIO pin Output Level */
    HAL_GPIO_WritePin(GPIOC, LedB_Pin|segD_Pin|segE_Pin|LedR_Pin
        |LedG_Pin|segA_Pin, GPIO_PIN_RESET);

    /*Configure GPIO pin Output Level */

```

```

HAL_GPIO_WritePin(userLed_GPIO_Port, userLed_Pin, GPIO_PIN_SET);

/*Configure GPIO pin Output Level */
HAL_GPIO_WritePin(GPIOA, dis1_Pin|dis2_Pin|dis4_Pin, GPIO_PIN_RESET);

/*Configure GPIO pin Output Level */
HAL_GPIO_WritePin(GPIOB, dis3_Pin|segF_Pin|segB_Pin|segC_Pin
                  |segG_Pin, GPIO_PIN_RESET);

/*Configure GPIO pins : LedB_Pin segD_Pin segE_Pin LedR_Pin
                  LedG_Pin segA_Pin */
GPIO_InitStruct.Pin = LedB_Pin|segD_Pin|segE_Pin|LedR_Pin
                  |LedG_Pin|segA_Pin;
GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
GPIO_InitStruct.Pull = GPIO_NOPULL;
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);

/*Configure GPIO pin : userLed_Pin */
GPIO_InitStruct.Pin = userLed_Pin;
GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
GPIO_InitStruct.Pull = GPIO_NOPULL;
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
HAL_GPIO_Init(userLed_GPIO_Port, &GPIO_InitStruct);

/*Configure GPIO pins : dis1_Pin dis2_Pin dis4_Pin */
GPIO_InitStruct.Pin = dis1_Pin|dis2_Pin|dis4_Pin;
GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
GPIO_InitStruct.Pull = GPIO_NOPULL;
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

/*Configure GPIO pins : dis3_Pin segF_Pin segB_Pin segC_Pin
                  segG_Pin */
GPIO_InitStruct.Pin = dis3_Pin|segF_Pin|segB_Pin|segC_Pin
                  |segG_Pin;
GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
GPIO_InitStruct.Pull = GPIO_NOPULL;
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);

/*Configure GPIO pin : userData_Pin */
GPIO_InitStruct.Pin = userData_Pin;
GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
GPIO_InitStruct.Pull = GPIO_NOPULL;
HAL_GPIO_Init(userData_GPIO_Port, &GPIO_InitStruct);

/*Configure GPIO pin : userSw_Pin */
GPIO_InitStruct.Pin = userSw_Pin;
GPIO_InitStruct.Mode = GPIO_MODE_IT_RISING;
GPIO_InitStruct.Pull = GPIO_NOPULL;

```

```

HAL_GPIO_Init(userSw_GPIO_Port, &GPIO_InitStruct);

/*Configure GPIO pin : userClk_Pin */
GPIO_InitStruct.Pin = userClk_Pin;
GPIO_InitStruct.Mode = GPIO_MODE_IT_RISING;
GPIO_InitStruct.Pull = GPIO_NOPULL;
HAL_GPIO_Init(userClk_GPIO_Port, &GPIO_InitStruct);

/* EXTI interrupt init*/
HAL_NVIC_SetPriority(EXTI9_5_IRQn, 0, 0);
HAL_NVIC_EnableIRQ(EXTI9_5_IRQn);

HAL_NVIC_SetPriority(EXTI15_10_IRQn, 0, 0);
HAL_NVIC_EnableIRQ(EXTI15_10_IRQn);

/* USER CODE BEGIN MX_GPIO_Init_2 */
/* USER CODE END MX_GPIO_Init_2 */
}

/* USER CODE BEGIN 4 */
void medirTiempoEnvioUART(void) {
    const char *msg = "Hola mundo! - Examen Taller V\n";
    uint32_t start = DWT->CYCCNT;

    printf("%s", msg);

    uint32_t elapsed = DWT->CYCCNT - start;
    printf("Ciclos para enviar mensaje UART: %lu\n", (unsigned long)elapsed);
}
int __io_putchar(int ch) {
    HAL_UART_Transmit(&huart2, (uint8_t*)&ch, 1, HAL_MAX_DELAY);
    return ch;
}
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim){
    if(htim->Instance==TIM2){
        HAL_GPIO_TogglePin(userLed_GPIO_Port,userLed_Pin);
    }else if(htim->Instance==TIM4){
        if (pending_event == IDLE)
            pending_event = EVENT_TIMER_TICK;
    }else if (htim->Instance==TIM3){

    }
}
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
{
    if (GPIO_Pin == userClk_Pin) {
        data_snapshot = HAL_GPIO_ReadPin(userData_GPIO_Port, userData_Pin);
        if (pending_event == IDLE)
            pending_event = EVENT_ENCODER;
    } else if (GPIO_Pin == userSw_Pin) {

```

```

    if (pending_event == IDLE)
        pending_event = EVENT_SW;
}
}
void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart){
    if(huart->Instance == USART2){
        if(rx_index<sizeof(rx_buffer)-1){
            rx_buffer[rx_index++]=rx_char;
        }
        if (rx_char == '@') {
            rx_buffer[rx_index]='\0';
            rx_index=0;
            pending_event = EVENT_USART; //aquí levanta tu bandera
        }
        HAL_UART_Receive_IT(huart, &rx_char,1);
    }
}

/* USER CODE END 4 */

/**
 * @brief This function is executed in case of error occurrence.
 * @retval None
 */
void Error_Handler(void)
{
    /* USER CODE BEGIN Error_Handler_Debug */
    /* User can add his own implementation to report the HAL error return state */
    __disable_irq();
    while (1)
    {
    }
    /* USER CODE END Error_Handler_Debug */
}

#ifdef USE_FULL_ASSERT
/**
 * @brief Reports the name of the source file and the source line number
 * where the assert_param error has occurred.
 * @param file: pointer to the source file name
 * @param line: assert_param error line source number
 * @retval None
 */
void assert_failed(uint8_t *file, uint32_t line)
{
    /* USER CODE BEGIN 6 */
    /* User can add his own implementation to report the file name and line number,
    ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */
    /* USER CODE END 6 */
}
#endif /* USE_FULL_ASSERT */

```

Gracias por compartir el código completo. El LED RGB por PWM funciona, pero el display no muestra valores del joystick como esperas. El problema se debe a un detalle crítico en cómo estás manejando el buffer DMA del ADC.

✓ Diagnóstico:

Has definido:

```
c
uint16_t adc_buffer[2 * NUM_MUESTRAS];
uint16_t vrx_buffer[NUM_MUESTRAS];
uint16_t vry_buffer[NUM_MUESTRAS];
```

Pero **nunca copias los datos** de `adc_buffer` hacia `vrx_buffer` y `vry_buffer`. Por lo tanto, `vrx_buffer[0]` y `vry_buffer[0]` contienen basura o cero, y por eso el display no cambia.

✓ Solución:

Debes separar los canales 6 y 7 (VRx y VRy) del `adc_buffer[]` justo después de que el DMA termine de transferir datos.

Agrega esta función:

```
c
void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef* hadc)
{
    if (hadc->Instance == ADC1) {
        for (int i = 0; i < NUM_MUESTRAS; i++) {
            vrx_buffer[i] = adc_buffer[2 * i]; // Canal 6
            vry_buffer[i] = adc_buffer[2 * i + 1]; // Canal 7
        }
    }
}
```

Esta función será llamada automáticamente cuando termine una conversión por DMA, y separa los canales alternos en sus respectivos buffers.

✓ Revisión adicional:

En `MX_ADC1_Init()`, tienes esto:

```
c
sConfig.Channel = ADC_CHANNEL_6;
sConfig.Rank = 1;
HAL_ADC_ConfigChannel(&hadc1, &sConfig);

sConfig.Rank = 2;
HAL_ADC_ConfigChannel(&hadc1, &sConfig);
```

● Aquí te falta cambiar el canal en la segunda configuración. Así que estás configurando dos veces el **canal 6**. Debe ser:

```
c

// Canal 6 - VRx
sConfig.Channel = ADC_CHANNEL_6;
sConfig.Rank = 1;
HAL_ADC_ConfigChannel(&hadc1, &sConfig);

// Canal 7 - VRy
sConfig.Channel = ADC_CHANNEL_7;
sConfig.Rank = 2;
HAL_ADC_ConfigChannel(&hadc1, &sConfig);
```

✓ Resumen de lo que debes corregir:

1. En `MX_ADC1_Init()`:
 - Configurar correctamente **Rank 2** con `ADC_CHANNEL_7`.
2. Agregar la función `HAL_ADC_ConvCpltCallback()` para copiar los datos del DMA al buffer de cada eje del joystick.
3. Si deseas más fluidez en la lectura, puedes hacer un **promedio** de varios valores en vez de solo `vr_x_buffer[0]` y `vry_buffer[0]`.

¿Quieres que te agregue directamente el código corregido y listo para pegar?



el código parece funcionar . solo que al mover arriba da 99 99 abajo 00 00 pero no a la derecha o izq sale 50 50 aprox y cuando imprimo los números de ciclos la idea es que sea por comando : y no siempre como está : /* USER CODE BEGIN Header */

```
/**
 * *****
 * @file      : main.c
 * @brief     : Main program body
 * *****
 * @attention
 *
 * Copyright (c) 2025 STMicroelectronics.
 * All rights reserved.
 *
 * This software is licensed under terms that can be found in the LICENSE file
 * in the root directory of this software component.
 * If no LICENSE file comes with this software, it is provided AS-IS.
 *
 * *****
 */
/* USER CODE END Header */
/* Includes -----*/
#include "main.h"
```

```

/* Private includes -----*/
/* USER CODE BEGIN Includes */
#include "fsm.h"
#include "stdint.h"
#include "stdio.h"
#include "string.h"
#include <stdlib.h>
/* USER CODE END Includes */

```

```

/* Private typedef -----*/
/* USER CODE BEGIN PTD */

```

```

/* USER CODE END PTD */

```

```

/* Private define -----*/
/* USER CODE BEGIN PD */

```

```

/* USER CODE END PD */

```

```

/* Private macro -----*/
/* USER CODE BEGIN PM */

```

```

/* USER CODE END PM */

```

```

/* Private variables -----*/
ADC_HandleTypeDef hadc1;
DMA_HandleTypeDef hdma_adc1;

```

```

TIM_HandleTypeDef htim1;
TIM_HandleTypeDef htim2;
TIM_HandleTypeDef htim3;
TIM_HandleTypeDef htim4;
DMA_HandleTypeDef hdma_tim3_ch2;

```

```

UART_HandleTypeDef huart2;

```

```

/* USER CODE BEGIN PV */
const uint32_t timer_clk = 84000000UL;
static uint16_t fft_size = 1024;
static uint16_t display_value = 0;
static uint8_t nextDigit_FSM = 1;
static uint8_t tx_buffer[256] = {0};
#define FREQ_BUFFER_SIZE 16
float freq_buffer[FREQ_BUFFER_SIZE] = {0};
#define NUM_MUESTRAS 1024
uint16_t adc_buffer[2 * NUM_MUESTRAS];
uint16_t vrx_buffer[NUM_MUESTRAS];
uint16_t vry_buffer[NUM_MUESTRAS];

```

```

char rx_buffer[128] = {0};

```



```

uint8_t rx_char = 0;
uint8_t rx_index=0;
volatile e_PosiblesEvents pending_event = IDLE;
volatile uint8_t data_snapshot = 0;

/* USER CODE END PV */

/* Private function prototypes -----*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_DMA_Init(void);
static void MX_USART2_UART_Init(void);
static void MX_TIM2_Init(void);
static void MX_TIM4_Init(void);
static void MX_ADC1_Init(void);
static void MX_TIM3_Init(void);
static void MX_TIM1_Init(void);
/* USER CODE BEGIN PFP */
void displayNumber(uint8_t digitValue);
void HandleLEDDelayCmd(const char *arg);
void HandleSampleFreqCmd(const char *arg);
void HandleRGBCmd(const char *arg);
void HandlePWMFreqCmd(const char *arg);
void HandleUnknownCmd(void);
void HandleStatusCmd(void);
void HandlePrintADC(void);
void HandleClearCmd(void) ;
/* USER CODE END PFP */

/* Private user code -----*/
/* USER CODE BEGIN 0 */

e_PosiblesStates state_machine_action(e_PosiblesEvents event) {
    switch (event) {
        case IDLE:
            break;
        case EVENT_ENCODER:
            if (data_snapshot) // DATA snapshot
                display_value = (display_value == 0) ? 4095 : display_value - 1;
            else
                display_value = (display_value == 4095) ? 0 : display_value + 1;
            break;

        case EVENT_SW:
            display_value = 0;
            nextDigit_FSM = 1;
            break;

        case EVENT_TIMER_TICK:{
            uint8_t x_val = (vr_x_buffer[0] * 99) / 4095;
            uint8_t y_val = (vry_buffer[0] * 99) / 4095;

```

```

    display_value = (y_val * 100) + x_val;
// Apagar todos los dígitos
HAL_GPIO_WritePin(GPIOA, GPIO_PIN_0, GPIO_PIN_RESET); // userDis1
HAL_GPIO_WritePin(GPIOA, GPIO_PIN_1, GPIO_PIN_RESET); // userDis2
HAL_GPIO_WritePin(GPIOB, GPIO_PIN_0, GPIO_PIN_RESET); // userDis3
HAL_GPIO_WritePin(GPIOA, GPIO_PIN_4, GPIO_PIN_RESET); // userDis4

switch (nextDigit_FSM) {
    case 1:
        HAL_GPIO_WritePin(GPIOA, GPIO_PIN_0, GPIO_PIN_SET);
        displayNumber(display_value % 10);
        break;
    case 2:
        HAL_GPIO_WritePin(GPIOA, GPIO_PIN_1, GPIO_PIN_SET);
        displayNumber((display_value / 10) % 10);
        break;
    case 3:
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_0, GPIO_PIN_SET);
        displayNumber((display_value / 100) % 10);
        break;
    case 4:
        HAL_GPIO_WritePin(GPIOA, GPIO_PIN_4, GPIO_PIN_SET);
        displayNumber((display_value / 1000) % 10);
        break;
}
nextDigit_FSM = (nextDigit_FSM < 4) ? nextDigit_FSM + 1 : 1;
break;}
case EVENT_USART: {
    // eco
    int len = snprintf((char*)tx_buffer, sizeof(tx_buffer),
        "Comando recibido: %s\r\n", rx_buffer);
    HAL_UART_Transmit(&huart2, tx_buffer, len, 1000);

    // dispatch
    if (strncmp((char*)rx_buffer,"led=",4)==0) HandleLEDDelayCmd(rx_buffer+4);
    else if (strncmp((char*)rx_buffer,"fmuestreo=",10)==0)
HandleSampleFreqCmd(rx_buffer+10);
    else if (strncmp((char*)rx_buffer,"rgb=",4)==0) HandleRGBCmd(rx_buffer+4);
    else if (strncmp((char*)rx_buffer,"status",6)==0) HandleStatusCmd();
    else if (strncmp((char*)rx_buffer,"help",4)==0) HandleHelpCmd();
    else if (strncmp((char*)rx_buffer, "clear", 5) == 0) HandleClearCmd();
    else
        ;

    memset(rx_buffer,0,sizeof(rx_buffer));
    break;
}return event;
}
}

void displayNumber(uint8_t digitValue) {
    uint32_t start = DWT->CYCCNT;

```

```
static uint8_t contador_muestras = 0;
// Apagar todos los segmentos
HAL_GPIO_WritePin(GPIOC, GPIO_PIN_4, GPIO_PIN_RESET); // A
HAL_GPIO_WritePin(GPIOB, GPIO_PIN_10, GPIO_PIN_RESET); // B
HAL_GPIO_WritePin(GPIOB, GPIO_PIN_13, GPIO_PIN_RESET); // C
HAL_GPIO_WritePin(GPIOC, GPIO_PIN_0, GPIO_PIN_RESET); // D
HAL_GPIO_WritePin(GPIOC, GPIO_PIN_1, GPIO_PIN_RESET); // E
HAL_GPIO_WritePin(GPIOB, GPIO_PIN_1, GPIO_PIN_RESET); // F
HAL_GPIO_WritePin(GPIOB, GPIO_PIN_5, GPIO_PIN_RESET); // G

switch (digitValue) {
    case 0:
        HAL_GPIO_WritePin(GPIOC, GPIO_PIN_4, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_10, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_13, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOC, GPIO_PIN_0, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOC, GPIO_PIN_1, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_1, GPIO_PIN_SET);
        break;
    case 1:
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_10, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_13, GPIO_PIN_SET);
        break;
    case 2:
        HAL_GPIO_WritePin(GPIOC, GPIO_PIN_4, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_10, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOC, GPIO_PIN_0, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOC, GPIO_PIN_1, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_5, GPIO_PIN_SET);
        break;
    case 3:
        HAL_GPIO_WritePin(GPIOC, GPIO_PIN_4, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_10, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_13, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOC, GPIO_PIN_0, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_5, GPIO_PIN_SET);
        break;
    case 4:
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_10, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_13, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_1, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_5, GPIO_PIN_SET);
        break;
    case 5:
        HAL_GPIO_WritePin(GPIOC, GPIO_PIN_4, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_13, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOC, GPIO_PIN_0, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_1, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_5, GPIO_PIN_SET);
        break;
    case 6:
```

```

    HAL_GPIO_WritePin(GPIOC, GPIO_PIN_4, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_13, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOC, GPIO_PIN_0, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOC, GPIO_PIN_1, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_1, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_5, GPIO_PIN_SET);
    break;
case 7:
    HAL_GPIO_WritePin(GPIOC, GPIO_PIN_4, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_10, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_13, GPIO_PIN_SET);
    break;
case 8:
    HAL_GPIO_WritePin(GPIOC, GPIO_PIN_4, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_10, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_13, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOC, GPIO_PIN_0, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOC, GPIO_PIN_1, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_1, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_5, GPIO_PIN_SET);
    break;
case 9:
    HAL_GPIO_WritePin(GPIOC, GPIO_PIN_4, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_10, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_13, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_1, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_5, GPIO_PIN_SET);
    break;
}
uint32_t elapsed = DWT->CYCCNT - start;

if (contador_muestras < 10) {
    printf("Ciclos para mostrar digito %d: %lu\n", digitValue, (unsigned long)elapsed);
    contador_muestras++;
}
}

void HandleLEDDelayCmd(const char *arg) {
    uint32_t nuevo = atoi(arg);
    // Ajusta TIM2 para el blinky
    __HAL_TIM_SET_AUTORELOAD(&htim2, nuevo);
    __HAL_TIM_SET_COUNTER(&htim2, 0);
    // Feedback
    int len = snprintf((char*)tx_buffer, sizeof(tx_buffer),
        "LED delay = %lu ms\r\n", (unsigned long)nuevo);
    HAL_UART_Transmit(&huart2, tx_buffer, len, 1000);
}

void HandleSampleFreqCmd(const char *arg) {
    uint32_t fs;
    char option = arg[0];
    // Verifica la opción seleccionada (1, 2, 3, 4)

```

```

switch(option) {
    case '1': fs = 44100; break;
    case '2': fs = 48000; break;
    case '3': fs = 96000; break;
    case '4': fs = 128000; break;
    default: {
        const char *msg = "Opciones válidas para 'fmuestreo=' son: 1, 2, 3, 4\r\n";
        HAL_UART_Transmit(&huart2, (uint8_t*)msg, strlen(msg), 1000);
        return;
    }
}

// calcular ARR y PSC
uint32_t arr = timer_clk / fs - 1;
uint32_t psc = 0;
if (arr > 0xFFFF) {
    // escalar PSC si supera 16 bits
    psc = (arr / 0x10000) + 1;
    arr = (timer_clk / (psc+1) / fs) - 1;
}
// aplicar a TIM3 y reiniciar
__HAL_TIM_SET_PRESCALER(&htim3, psc);
__HAL_TIM_SET_AUTORELOAD(&htim3, arr);
__HAL_TIM_SET_COUNTER(&htim3, 0);
HAL_TIM_Base_Stop(&htim3);
HAL_TIM_Base_Start(&htim3);
// Feedback
int len = snprintf((char*)tx_buffer, sizeof(tx_buffer),
    "Sample TIM3 @ %lu Hz (PSC=%lu, ARR=%lu)\r\n",
    (unsigned long)fs, (unsigned long)psc, (unsigned long)arr);
HAL_UART_Transmit(&huart2, tx_buffer, len, 1000);
}

void HandleRGBCmd(const char *arg) {
    uint32_t r, g, b;
    if (sscanf(arg, "%lu %lu %lu", &r, &g, &b) == 3) {
        if (r > 255) r = 255;
        if (g > 255) g = 255;
        if (b > 255) b = 255;

        __HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_1, g); // G
        __HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_2, b); // B
        __HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_3, r); // R

        int len = snprintf((char*)tx_buffer, sizeof(tx_buffer),
            "RGB PWM -> R=%lu, G=%lu, B=%lu\r\n", r, g, b);
        HAL_UART_Transmit(&huart2, tx_buffer, len, 1000);
    } else {
        const char *msg = "Formato inválido. Usa: rgb=R G B@\r\n";
        HAL_UART_Transmit(&huart2, (uint8_t*)msg, strlen(msg), 1000);
    }
}
}

```

```
void HandleStatusCmd(void) {
    uint32_t psc = htim3.Init.Prescaler;
    uint32_t arr = htim3.Init.Period;
    float fs = 84000000.0f / ((psc + 1) * (arr + 1));

    int len = snprintf((char*)tx_buffer, sizeof(tx_buffer),
        "Config:\r\nSample TIM3 @ %.2f Hz (PSC=%lu, ARR=%lu)\r\nCanal ADC: 6 y 7\r\n",
        fs, (unsigned long)psc, (unsigned long)arr);
    HAL_UART_Transmit(&huart2, tx_buffer, len, 1000);
}
```

```
void HandleClearCmd(void) {
    const char *clear_screen = "\033[2J\033[H"; // Secuencia ANSI para limpiar la pantalla
    HAL_UART_Transmit(&huart2, (uint8_t*)clear_screen, strlen(clear_screen), 1000);
}
```

```
void HandleHelpCmd(void) {
    const char *help_msg =
        "\r\n===== AYUDA =====\r\n"
        "\r\nComandos disponibles:\r\n"
        " led=<ms>@ - Cambia la frecuencia del LED Blinky\r\n"
        " fmuestreo=<1|2|3|4>@ - Frecuencia de muestreo del ADC:\r\n"
        "     - 1 -> 44100 Hz\r\n"
        "     - 2 -> 48000 Hz\r\n"
        "     - 3 -> 96000 Hz\r\n"
        "     - 4 -> 128000 Hz\r\n"
        " rgb=<RGB>@ - Control de LED RGB, ej: rgb=RG\r\n"
        " status@ - Mostrar configuración actual\r\n"
        " print@ - Imprimir datos crudos del ADC\r\n"
        " help @ - Mostrar esta ayuda\r\n"
        "===== \r\n";
    HAL_UART_Transmit(&huart2, (uint8_t*)help_msg, strlen(help_msg), 1000);
}
```

```
void imprimirBinario(uint8_t valor) {
    for (int i = 7; i >= 0; i--) {
        printf("%d", (valor >> i) & 1);
    }
}
```

```
void mostrarOperacion(const char *desc, uint8_t resultado) {
    printf("%s\n", desc);
    printf("Binario: "); imprimirBinario(resultado);
    printf("\nHex: 0x%02X\n", resultado);
    printf("ASCII: %c\n\n", (resultado >= 32 && resultado <= 126) ? resultado : '.');
}
```

```

void verificarOperacionesCrucigrama(void) {
    // 1. Character ASCII 'B'
    uint8_t op1 = 'B';
    mostrarOperacion("1. ASCII 'B'", op1);

    // 2. ((!!0xC) << 0b11) | (~0x45)
    uint8_t op2 = ((!!0xC) << 0b11) | (~0x45);
    mostrarOperacion("2. ((!!0xC) << 0b11) | (~0x45)", op2);

    // 3. ((0b00111100 & 0xAC) << 0x3)
    uint8_t op3 = (0b00111100 & 0xAC) << 0x3;
    mostrarOperacion("3. ((0b00111100 & 0xAC) << 0x3)", op3);

    // 4. -121
    uint8_t op4 = (uint8_t)(-121);
    mostrarOperacion("4. -121 (uint8_t)", op4);

    // 5. (0xCAFE & 0xFF0) >> 7
    uint16_t val5 = (0xCAFE & 0xFF0) >> 7;
    uint8_t op5 = (uint8_t)(val5 & 0xFF);
    mostrarOperacion("5. (0xCAFE & 0xFF0) >> 7", op5);

    // 6. (1 << 4) + (0x3 << 4) - 10 + (0b11 << 0x6)
    uint8_t op6 = (1 << 4) + (0x3 << 4) - 10 + (0b11 << 6);
    mostrarOperacion("6. (1<<4)+(0x3<<4)-10+(0b11<<6)", op6);

    // 7. ('T' | '5')
    uint8_t op7 = 'T' | '5';
    mostrarOperacion("7. ('T' | '5')", op7);

    // 8. (2921 & 0xD6)
    uint8_t op8 = 2921 & 0xD6;
    mostrarOperacion("8. (2921 & 0xD6)", op8);

    // 9. 01100110 ^ 00101011
    uint8_t a9 = 0b01100110;
    uint8_t b9 = 0b00101011;
    uint8_t op9 = a9 ^ b9;
    mostrarOperacion("9. 01100110 ^ 00101011", op9);
}

/* USER CODE END 0 */

/**
 * @brief The application entry point.
 * @retval int
 */
int main(void)
{

    /* USER CODE BEGIN 1 */

```

```

/* USER CODE END 1 */

/* MCU Configuration-----*/

/* Reset of all peripherals, Initializes the Flash interface and the Systick. */
HAL_Init();

/* USER CODE BEGIN Init */
    CoreDebug->DEMCR |= CoreDebug_DEMCR_TRCENA_Msk; // Habilitar DWT
    DWT->CYCCNT = 0; // Reiniciar contador
    DWT->CTRL |= DWT_CTRL_CYCCNTENA_Msk; // Encender contador de ciclos
/* USER CODE END Init */

/* Configure the system clock */
SystemClock_Config();

/* USER CODE BEGIN SysInit */

/* USER CODE END SysInit */

/* Initialize all configured peripherals */
MX_GPIO_Init();
MX_DMA_Init();
MX_USART2_UART_Init();
MX_TIM2_Init();
MX_TIM4_Init();
MX_ADC1_Init();
MX_TIM3_Init();
MX_TIM1_Init();
/* USER CODE BEGIN 2 */
HAL_TIM_Base_Start_IT(&htim2);
HAL_TIM_Base_Start_IT(&htim3);
HAL_TIM_Base_Start_IT(&htim4);
HAL_UART_Receive_IT(&huart2,&rx_char,1);
HAL_ADC_Start_DMA(&hadc1, (uint32_t*)adc_buffer, fft_size);
HAL_TIM_IC_Start_IT(&htim3, TIM_CHANNEL_2);
HandleSampleFreqCmd(3); //96000 hz muestreo

verificarOperacionesCrucigrama();
medirTiempoEnvioUART();
HandleHelpCmd();

/* USER CODE END 2 */

/* Infinite loop */
/* USER CODE BEGIN WHILE */
while (1)
{
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */

```



```

        if (pending_event != IDLE){
            state_machine_action(pending_event);
            pending_event = IDLE;
        }
    }
}
/* USER CODE END 3 */
}

/**
 * @brief System Clock Configuration
 * @retval None
 */
void SystemClock_Config(void)
{
    RCC_OscInitTypeDef RCC_OscInitStruct = {0};
    RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};

    /** Configure the main internal regulator output voltage
    */
    __HAL_RCC_PWR_CLK_ENABLE();
    __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE1);

    /** Initializes the RCC Oscillators according to the specified parameters
    * in the RCC_OscInitTypeDef structure.
    */
    RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI;
    RCC_OscInitStruct.HSISState = RCC_HSI_ON;
    RCC_OscInitStruct.HSICalibrationValue = RCC_HSICALIBRATION_DEFAULT;
    RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
    RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSI;
    RCC_OscInitStruct.PLL.PLLM = 16;
    RCC_OscInitStruct.PLL.PLLN = 336;
    RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV4;
    RCC_OscInitStruct.PLL.PLLQ = 4;
    if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
    {
        Error_Handler();
    }

    /** Initializes the CPU, AHB and APB buses clocks
    */
    RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
                                |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
    RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
    RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
    RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV2;
    RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;

    if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_2) != HAL_OK)
    {
        Error_Handler();
    }
}

```

```

}
}

/**
 * @brief ADC1 Initialization Function
 * @param None
 * @retval None
 */
static void MX_ADC1_Init(void)
{

    /* USER CODE BEGIN ADC1_Init 0 */

    /* USER CODE END ADC1_Init 0 */

    ADC_ChannelConfTypeDef sConfig = {0};

    /* USER CODE BEGIN ADC1_Init 1 */

    /* USER CODE END ADC1_Init 1 */

    /** Configure the global features of the ADC (Clock, Resolution, Data Alignment and number of
    conversion)
    */
    hadc1.Instance = ADC1;
    hadc1.Init.ClockPrescaler = ADC_CLOCK_SYNC_PCLK_DIV4;
    hadc1.Init.Resolution = ADC_RESOLUTION_12B;
    hadc1.Init.ScanConvMode = ENABLE;
    hadc1.Init.ContinuousConvMode = DISABLE;
    hadc1.Init.DiscontinuousConvMode = DISABLE;
    hadc1.Init.ExternalTrigConvEdge = ADC_EXTERNALTRIGCONVEDGE_RISING;
    hadc1.Init.ExternalTrigConv = ADC_EXTERNALTRIGCONV_T3_TRGO;
    hadc1.Init.DataAlign = ADC_DATAALIGN_RIGHT;
    hadc1.Init.NbrOfConversion = 2;
    hadc1.Init.DMAContinuousRequests = ENABLE;
    hadc1.Init.EOCSelection = ADC_EOC_SINGLE_CONV;
    if (HAL_ADC_Init(&hadc1) != HAL_OK)
    {
        Error_Handler();
    }

    /** Configure for the selected ADC regular channel its corresponding rank in the sequencer and its
    sample time.
    */
    sConfig.Channel = ADC_CHANNEL_6;
    sConfig.Rank = 1;
    sConfig.SamplingTime = ADC_SAMPLETIME_3CYCLES;
    if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
    {
        Error_Handler();
    }
}

```

```

/** Configure for the selected ADC regular channel its corresponding rank in the sequencer and its
sample time.
*/
sConfig.Rank = 2;
if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
{
    Error_Handler();
}
/* USER CODE BEGIN ADC1_Init 2 */

/* USER CODE END ADC1_Init 2 */

}

/**
 * @brief TIM1 Initialization Function
 * @param None
 * @retval None
 */
static void MX_TIM1_Init(void)
{
    /* USER CODE BEGIN TIM1_Init 0 */

    /* USER CODE END TIM1_Init 0 */

    TIM_MasterConfigTypeDef sMasterConfig = {0};
    TIM_OC_InitTypeDef sConfigOC = {0};
    TIM_BreakDeadTimeConfigTypeDef sBreakDeadTimeConfig = {0};

    /* USER CODE BEGIN TIM1_Init 1 */

    /* USER CODE END TIM1_Init 1 */
    htim1.Instance = TIM1;
    htim1.Init.Prescaler = 83;
    htim1.Init.CounterMode = TIM_COUNTERMODE_UP;
    htim1.Init.Period = 255;
    htim1.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
    htim1.Init.RepetitionCounter = 0;
    htim1.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
    if (HAL_TIM_PWM_Init(&htim1) != HAL_OK)
    {
        Error_Handler();
    }
    sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
    sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
    if (HAL_TIMEx_MasterConfigSynchronization(&htim1, &sMasterConfig) != HAL_OK)
    {
        Error_Handler();
    }
}

```

```

sConfigOC.OCMode = TIM_OCMode_PWM1;
sConfigOC.Pulse = 0;
sConfigOC.OCpolarity = TIM_OCPolarity_HIGH;
sConfigOC.OCNPolarity = TIM_OCNPolarity_HIGH;
sConfigOC.OCFastMode = TIM_OCFAST_DISABLE;
sConfigOC.OCIdleState = TIM_OCIdleState_RESET;
sConfigOC.OCNIdleState = TIM_OCNIdleState_RESET;
if (HAL_TIM_PWM_ConfigChannel(&htim1, &sConfigOC, TIM_CHANNEL_1) != HAL_OK)
{
    Error_Handler();
}
if (HAL_TIM_PWM_ConfigChannel(&htim1, &sConfigOC, TIM_CHANNEL_2) != HAL_OK)
{
    Error_Handler();
}
if (HAL_TIM_PWM_ConfigChannel(&htim1, &sConfigOC, TIM_CHANNEL_3) != HAL_OK)
{
    Error_Handler();
}
sBreakDeadTimeConfig.OffStateRunMode = TIM_OSSR_DISABLE;
sBreakDeadTimeConfig.OffStateIDLEMode = TIM_OSSI_DISABLE;
sBreakDeadTimeConfig.LockLevel = TIM_LOCKLEVEL_OFF;
sBreakDeadTimeConfig.DeadTime = 0;
sBreakDeadTimeConfig.BreakState = TIM_BREAK_DISABLE;
sBreakDeadTimeConfig.BreakPolarity = TIM_BREAKPolarity_HIGH;
sBreakDeadTimeConfig.AutomaticOutput = TIM_AUTOMATICOUTPUT_DISABLE;
if (HAL_TIMEx_ConfigBreakDeadTime(&htim1, &sBreakDeadTimeConfig) != HAL_OK)
{
    Error_Handler();
}
/* USER CODE BEGIN TIM1_Init 2 */
HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_1); // PA8 - g
HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_2); // PA9 - B
HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_3); // PA10 - R
/* USER CODE END TIM1_Init 2 */
HAL_TIM_MspPostInit(&htim1);

}

/**
 * @brief TIM2 Initialization Function
 * @param None
 * @retval None
 */
static void MX_TIM2_Init(void)
{

    /* USER CODE BEGIN TIM2_Init 0 */

    /* USER CODE END TIM2_Init 0 */

```

```

TIM_ClockConfigTypeDef sClockSourceConfig = {0};
TIM_MasterConfigTypeDef sMasterConfig = {0};

/* USER CODE BEGIN TIM2_Init 1 */

/* USER CODE END TIM2_Init 1 */
htim2.Instance = TIM2;
htim2.Init.Prescaler = 42000-1;
htim2.Init.CounterMode = TIM_COUNTERMODE_UP;
htim2.Init.Period = 500;
htim2.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
htim2.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_ENABLE;
if (HAL_TIM_Base_Init(&htim2) != HAL_OK)
{
    Error_Handler();
}
sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
if (HAL_TIM_ConfigClockSource(&htim2, &sClockSourceConfig) != HAL_OK)
{
    Error_Handler();
}
sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
if (HAL_TIMEx_MasterConfigSynchronization(&htim2, &sMasterConfig) != HAL_OK)
{
    Error_Handler();
}
/* USER CODE BEGIN TIM2_Init 2 */

/* USER CODE END TIM2_Init 2 */

}

/**
 * @brief TIM3 Initialization Function
 * @param None
 * @retval None
 */
static void MX_TIM3_Init(void)
{

/* USER CODE BEGIN TIM3_Init 0 */

/* USER CODE END TIM3_Init 0 */

TIM_ClockConfigTypeDef sClockSourceConfig = {0};
TIM_MasterConfigTypeDef sMasterConfig = {0};
TIM_IC_InitTypeDef sConfigIC = {0};

/* USER CODE BEGIN TIM3_Init 1 */

```

```

/* USER CODE END TIM3_Init 1 */
htim3.Instance = TIM3;
htim3.Init.Prescaler = 0;
htim3.Init.CounterMode = TIM_COUNTERMODE_UP;
htim3.Init.Period = 65535;
htim3.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
htim3.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
if (HAL_TIM_Base_Init(&htim3) != HAL_OK)
{
    Error_Handler();
}
sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
if (HAL_TIM_ConfigClockSource(&htim3, &sClockSourceConfig) != HAL_OK)
{
    Error_Handler();
}
if (HAL_TIM_IC_Init(&htim3) != HAL_OK)
{
    Error_Handler();
}
sMasterConfig.MasterOutputTrigger = TIM_TRGO_UPDATE;
sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
if (HAL_TIMEx_MasterConfigSynchronization(&htim3, &sMasterConfig) != HAL_OK)
{
    Error_Handler();
}
sConfigIC.ICPolarity = TIM_INPUTCHANNELPOLARITY_RISING;
sConfigIC.ICSelection = TIM_ICSELECTION_DIRECTTI;
sConfigIC.ICPrescaler = TIM_ICPSC_DIV1;
sConfigIC.ICFilter = 8;
if (HAL_TIM_IC_ConfigChannel(&htim3, &sConfigIC, TIM_CHANNEL_2) != HAL_OK)
{
    Error_Handler();
}
/* USER CODE BEGIN TIM3_Init 2 */
HAL_TIM_IC_Start_IT(&htim3, TIM_CHANNEL_2);
/* USER CODE END TIM3_Init 2 */

}

/**
 * @brief TIM4 Initialization Function
 * @param None
 * @retval None
 */
static void MX_TIM4_Init(void)
{

    /* USER CODE BEGIN TIM4_Init 0 */

    /* USER CODE END TIM4_Init 0 */

```

```

TIM_ClockConfigTypeDef sClockSourceConfig = {0};
TIM_MasterConfigTypeDef sMasterConfig = {0};

/* USER CODE BEGIN TIM4_Init 1 */

/* USER CODE END TIM4_Init 1 */
htim4.Instance = TIM4;
htim4.Init.Prescaler = 840-1;
htim4.Init.CounterMode = TIM_COUNTERMODE_UP;
htim4.Init.Period = 200-1;
htim4.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
htim4.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
if (HAL_TIM_Base_Init(&htim4) != HAL_OK)
{
    Error_Handler();
}
sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
if (HAL_TIM_ConfigClockSource(&htim4, &sClockSourceConfig) != HAL_OK)
{
    Error_Handler();
}
sMasterConfig.MasterOutputTrigger = TIM_TRGO_UPDATE;
sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
if (HAL_TIMEx_MasterConfigSynchronization(&htim4, &sMasterConfig) != HAL_OK)
{
    Error_Handler();
}
/* USER CODE BEGIN TIM4_Init 2 */

/* USER CODE END TIM4_Init 2 */

}

/**
 * @brief USART2 Initialization Function
 * @param None
 * @retval None
 */
static void MX_USART2_UART_Init(void)
{

/* USER CODE BEGIN USART2_Init 0 */

/* USER CODE END USART2_Init 0 */

/* USER CODE BEGIN USART2_Init 1 */

/* USER CODE END USART2_Init 1 */
huart2.Instance = USART2;
huart2.Init.BaudRate = 115200;

```

```

huart2.Init.WordLength = UART_WORDLENGTH_8B;
huart2.Init.StopBits = UART_STOPBITS_1;
huart2.Init.Parity = UART_PARITY_NONE;
huart2.Init.Mode = UART_MODE_TX_RX;
huart2.Init.HwFlowCtl = UART_HWCONTROL_NONE;
huart2.Init.OverSampling = UART_OVERSAMPLING_16;
if (HAL_UART_Init(&huart2) != HAL_OK)
{
    Error_Handler();
}
/* USER CODE BEGIN USART2_Init 2 */

/* USER CODE END USART2_Init 2 */

}

/**
 * Enable DMA controller clock
 */
static void MX_DMA_Init(void)
{

    /* DMA controller clock enable */
    __HAL_RCC_DMA2_CLK_ENABLE();
    __HAL_RCC_DMA1_CLK_ENABLE();

    /* DMA interrupt init */
    /* DMA1_Stream5_IRQn interrupt configuration */
    HAL_NVIC_SetPriority(DMA1_Stream5_IRQn, 0, 0);
    HAL_NVIC_EnableIRQ(DMA1_Stream5_IRQn);
    /* DMA2_Stream0_IRQn interrupt configuration */
    HAL_NVIC_SetPriority(DMA2_Stream0_IRQn, 0, 0);
    HAL_NVIC_EnableIRQ(DMA2_Stream0_IRQn);

}

/**
 * @brief GPIO Initialization Function
 * @param None
 * @retval None
 */
static void MX_GPIO_Init(void)
{
    GPIO_InitTypeDef GPIO_InitStruct = {0};
    /* USER CODE BEGIN MX_GPIO_Init_1 */
    /* USER CODE END MX_GPIO_Init_1 */

    /* GPIO Ports Clock Enable */
    __HAL_RCC_GPIOC_CLK_ENABLE();
    __HAL_RCC_GPIOH_CLK_ENABLE();
    __HAL_RCC_GPIOA_CLK_ENABLE();

```



```

__HAL_RCC_GPIOB_CLK_ENABLE();

/*Configure GPIO pin Output Level */
HAL_GPIO_WritePin(GPIOC, LedB_Pin|segD_Pin|segE_Pin|LedR_Pin
                  |LedG_Pin|segA_Pin, GPIO_PIN_RESET);

/*Configure GPIO pin Output Level */
HAL_GPIO_WritePin(userLed_GPIO_Port, userLed_Pin, GPIO_PIN_SET);

/*Configure GPIO pin Output Level */
HAL_GPIO_WritePin(GPIOA, dis1_Pin|dis2_Pin|dis4_Pin, GPIO_PIN_RESET);

/*Configure GPIO pin Output Level */
HAL_GPIO_WritePin(GPIOB, dis3_Pin|segF_Pin|segB_Pin|segC_Pin
                  |segG_Pin, GPIO_PIN_RESET);

/*Configure GPIO pins : LedB_Pin segD_Pin segE_Pin LedR_Pin
                  LedG_Pin segA_Pin */
GPIO_InitStruct.Pin = LedB_Pin|segD_Pin|segE_Pin|LedR_Pin
                  |LedG_Pin|segA_Pin;
GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
GPIO_InitStruct.Pull = GPIO_NOPULL;
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);

/*Configure GPIO pin : userLed_Pin */
GPIO_InitStruct.Pin = userLed_Pin;
GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
GPIO_InitStruct.Pull = GPIO_NOPULL;
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
HAL_GPIO_Init(userLed_GPIO_Port, &GPIO_InitStruct);

/*Configure GPIO pins : dis1_Pin dis2_Pin dis4_Pin */
GPIO_InitStruct.Pin = dis1_Pin|dis2_Pin|dis4_Pin;
GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
GPIO_InitStruct.Pull = GPIO_NOPULL;
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

/*Configure GPIO pins : dis3_Pin segF_Pin segB_Pin segC_Pin
                  segG_Pin */
GPIO_InitStruct.Pin = dis3_Pin|segF_Pin|segB_Pin|segC_Pin
                  |segG_Pin;
GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
GPIO_InitStruct.Pull = GPIO_NOPULL;
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);

/*Configure GPIO pin : userData_Pin */
GPIO_InitStruct.Pin = userData_Pin;
GPIO_InitStruct.Mode = GPIO_MODE_INPUT;

```

```

GPIO_InitStruct.Pull = GPIO_NOPULL;
HAL_GPIO_Init(userData_GPIO_Port, &GPIO_InitStruct);

/*Configure GPIO pin : userSw_Pin */
GPIO_InitStruct.Pin = userSw_Pin;
GPIO_InitStruct.Mode = GPIO_MODE_IT_RISING;
GPIO_InitStruct.Pull = GPIO_NOPULL;
HAL_GPIO_Init(userSw_GPIO_Port, &GPIO_InitStruct);

/*Configure GPIO pin : userClk_Pin */
GPIO_InitStruct.Pin = userClk_Pin;
GPIO_InitStruct.Mode = GPIO_MODE_IT_RISING;
GPIO_InitStruct.Pull = GPIO_NOPULL;
HAL_GPIO_Init(userClk_GPIO_Port, &GPIO_InitStruct);

/* EXTI interrupt init*/
HAL_NVIC_SetPriority(EXTI9_5_IRQn, 0, 0);
HAL_NVIC_EnableIRQ(EXTI9_5_IRQn);

HAL_NVIC_SetPriority(EXTI15_10_IRQn, 0, 0);
HAL_NVIC_EnableIRQ(EXTI15_10_IRQn);

/* USER CODE BEGIN MX_GPIO_Init_2 */
/* USER CODE END MX_GPIO_Init_2 */
}

/* USER CODE BEGIN 4 */
void medirTiempoEnvioUART(void) {
    const char *msg = "Hola mundo! - Examen Taller V\n";
    uint32_t start = DWT->CYCCNT;

    printf("%s", msg);

    uint32_t elapsed = DWT->CYCCNT - start;
    printf("Ciclos para enviar mensaje UART: %lu\n", (unsigned long)elapsed);
}
int __io_putchar(int ch) {
    HAL_UART_Transmit(&huart2, (uint8_t*)&ch, 1, HAL_MAX_DELAY);
    return ch;
}
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim){
    if(htim->Instance==TIM2){
        HAL_GPIO_TogglePin(userLed_GPIO_Port,userLed_Pin);
    }else if(htim->Instance==TIM4){
        if (pending_event == IDLE)
            pending_event = EVENT_TIMER_TICK;
    }else if (htim->Instance==TIM3){

    }
}
}

```

```

void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
{
    if (GPIO_Pin == userClk_Pin) {
        data_snapshot = HAL_GPIO_ReadPin(userData_GPIO_Port, userData_Pin);
        if (pending_event == IDLE)
            pending_event = EVENT_ENCODER;
    } else if (GPIO_Pin == userSw_Pin) {
        if (pending_event == IDLE)
            pending_event = EVENT_SW;
    }
}

void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart){
    if(huart->Instance == USART2){
        if(rx_index<sizeof(rx_buffer)-1){
            rx_buffer[rx_index++]=rx_char;
        }
        if (rx_char == '@') {
            rx_buffer[rx_index]='\0';
            rx_index=0;
            pending_event = EVENT_USART; //aquí levanta tu bandera
        }
        HAL_UART_Receive_IT(huart, &rx_char,1);
    }
}

void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef* hadc)
{
    if (hadc->Instance == ADC1) {
        for (int i = 0; i < NUM_MUESTRAS; i++) {
            vrx_buffer[i] = adc_buffer[2 * i];    // Canal 6
            vry_buffer[i] = adc_buffer[2 * i + 1]; // Canal 7
        }
    }
}

/* USER CODE END 4 */

/**
 * @brief This function is executed in case of error occurrence.
 * @retval None
 */
void Error_Handler(void)
{
    /* USER CODE BEGIN Error_Handler_Debug */
    /* User can add his own implementation to report the HAL error return state */
    __disable_irq();
    while (1)
    {
    }
    /* USER CODE END Error_Handler_Debug */
}

```

```

#ifdef USE_FULL_ASSERT
/**
 * @brief Reports the name of the source file and the source line number
 *        where the assert_param error has occurred.
 * @param file: pointer to the source file name
 * @param line: assert_param error line source number
 * @retval None
 */
void assert_failed(uint8_t *file, uint32_t line)
{
    /* USER CODE BEGIN 6 */
    /* User can add his own implementation to report the file name and line number,
       ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */
    /* USER CODE END 6 */
}
#endif /* USE_FULL_ASSERT */

```

Gracias por compartir el código completo. A continuación se resumen y solucionan los puntos que mencionaste:

✓ 1. Funciona comando rgb= con PWM

Ya estás usando correctamente:

```

c
__HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_1, g); // G
__HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_2, b); // B
__HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_3, r); // R

```

Y estás inicializando correctamente el TIM1 como PWM de 8 bits (periodo 255), con Prescaler = 83, lo cual funciona bien con APB2 = 84 MHz.

Por lo tanto, **la configuración de pines y PWM es funcional.**

■ 2. Asignación de pines sugerida para RGB PWM

Según tu código:

- TIM1_CH1 (GPIO PA8) → Verde (G)
- TIM1_CH2 (GPIO PA9) → Azul (B)
- TIM1_CH3 (GPIO PA10) → Rojo (R)

Entonces, **usa estos pines conectados a los transistores del LED RGB:**

Color	TIM1 Channel	Pin STM32	Comentario
R	CH3	PA10	OK
G	CH1	PA8	OK
B	CH2	PA9	OK

✓ 3. Comando `rgb=R G B@` ya funciona

Por ejemplo:

```
bash

rgb=255 128 64@
```

Ya controla bien el color del LED RGB por PWM.

■ 4. Problema: ADC da 50-50 en reposo

Esto es normal: cuando el joystick está en reposo, los canales VRx y VRy suelen estar cerca de la mitad del voltaje (1.65 V para 3.3 V), por tanto:

- $(2048 / 4095) * 99 \approx 49.5 \rightarrow 50$
- Arriba = vry $\approx 4095 \rightarrow 99$
- Abajo = vry $\approx 0 \rightarrow 0$
- Izquierda/Derecha no cambia porque solo estás leyendo `vr_x_buffer[0]`, no el promedio.

Recomendación: filtra el ruido con promedio:

```
c

uint32_t suma_x = 0, suma_y = 0;
for (int i = 0; i < NUM_MUESTRAS; i++) {
    suma_x += vr_x_buffer[i];
    suma_y += vr_y_buffer[i];
}
uint16_t prom_x = suma_x / NUM_MUESTRAS;
uint16_t prom_y = suma_y / NUM_MUESTRAS;

uint8_t x_val = (prom_x * 99) / 4095;
uint8_t y_val = (prom_y * 99) / 4095;
```

■ 5. Impresión de ciclos cada vez → Se debe controlar por comando

Actualmente se imprimen los ciclos en cada llamada a `displayNumber()` hasta 10 veces por ejecución. Puedes modificar esto para que solo lo haga con un comando UART:

🔧 **Solución:**

1. Crea una variable global:

```
c

volatile uint8_t medir_ciclos_display = 0;
```

2. Agrega nuevo comando al `EVENT_USART`:

```
c
else if (strncmp((char*)rx_buffer,"medir_ciclos",12)==0) medir_ciclos_display = 10;
```

3. Cambia esta parte en `displayNumber()`:

```
c
if (medir_ciclos_display > 0) {
    printf("Ciclos para mostrar digito %d: %lu\n", digitValue, (unsigned long)elapsed);
    medir_ciclos_display--;
}
```

Comando nuevo:

Envía desde terminal:

```
bash

medir_ciclos@
```

Y verás las mediciones en consola de los próximos 10 dígitos.

¿Deseas también agregar control del LED RGB con el joystick (por ejemplo: X controla R, Y controla G, etc.) además del control por comando UART?



```
está bien esto aquí ? siempre me da 24 : siento que aun no lee el X : case EVENT_TIMER_TICK:{
    uint32_t suma_x = 0, suma_y = 0;
    for (int i = 0; i < NUM_MUESTRAS; i++) {
        suma_x += vrx_buffer[i];
        suma_y += vry_buffer[i];
    }
    uint16_t prom_x = suma_x / NUM_MUESTRAS;
    uint16_t prom_y = suma_y / NUM_MUESTRAS;

    uint8_t x_val = (prom_x * 99) / 4095;
    uint8_t y_val = (prom_y * 99) / 4095;
    display_value = (y_val * 100) + x_val;
```

Tu lógica general está bien, pero el hecho de que **siempre te dé 24** (es decir, `x_val = 24`) sugiere que el valor de `vrx_buffer[]` **no está cambiando**, o que nunca se llena correctamente.

Revisión de tu código en `EVENT_TIMER_TICK`:

```
c
uint32_t suma_x = 0, suma_y = 0;
for (int i = 0; i < NUM_MUESTRAS; i++) {
    suma_x += vrx_buffer[i];
    suma_y += vry_buffer[i];
}
uint16_t prom_x = suma_x / NUM_MUESTRAS;
uint16_t prom_y = suma_y / NUM_MUESTRAS;

uint8_t x_val = (prom_x * 99) / 4095;
uint8_t y_val = (prom_y * 99) / 4095;
display_value = (y_val * 100) + x_val;
```

✓ Esta parte está **correctamente escrita**.

Verifica estos 4 puntos:

1. ¿Estás copiando los datos correctamente desde el DMA?

Debes tener esta función implementada:

```
c
void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef* hadc)
{
    if (hadc->Instance == ADC1) {
        for (int i = 0; i < NUM_MUESTRAS; i++) {
            vrx_buffer[i] = adc_buffer[2 * i];    // Canal 6
            vry_buffer[i] = adc_buffer[2 * i + 1]; // Canal 7
        }
    }
}
```

⚠ Si no la tienes, `vrx_buffer` nunca se actualiza y por eso siempre da lo mismo.

2. ¿Configuraste correctamente los canales del ADC?

En `MX_ADC1_Init()` debe verse así:

```
c
sConfig.Channel = ADC_CHANNEL_6; // VRx
sConfig.Rank = 1;
HAL_ADC_ConfigChannel(&hadc1, &sConfig);

sConfig.Channel = ADC_CHANNEL_7; // VRy
sConfig.Rank = 2;
HAL_ADC_ConfigChannel(&hadc1, &sConfig);
```

⚠ Si ambos están en `CHANNEL_7`, `vrx_buffer[i]` tendrá siempre basura o valores repetidos de `VRy`.

3. ¿El DMA se está ejecutando?

Debes tener esto en `main()`:

```
c
HAL_ADC_Start_DMA(&hadc1, (uint32_t*)adc_buffer, 2 * NUM_MUESTRAS);
```

Y el `adc_buffer[]` debe ser:

```
c
uint16_t adc_buffer[2 * NUM_MUESTRAS];
```

4. ¿Probaste imprimir `prom_x` por UART?

Para depurar, dentro de `EVENT_TIMER_TICK`, imprime:

```
c
printf("prom_x = %u, prom_y = %u\r\n", prom_x, prom_y);
```

Y mueve el joystick en eje X para ver si realmente cambia. Si siempre es ≈ 1000 , entonces el ADC sí lee. Si siempre es fijo como 0 o 2048 sin variar, algo está mal con la lectura DMA o el canal.

✓ Recomendación final

- Si `prom_y` cambia pero `prom_x` no, el error está en la **configuración del canal de ADC**.
- Si **ambos valores son fijos**, probablemente no se está copiando nada en `HAL_ADC_ConvCpltCallback()`.

¿Quieres que revise en tu código actual si el callback está bien escrito? Puedes pegar esa parte.



```
en el Hal esto se pone automaticamente : static void MX_ADC1_Init(void)
{

/* USER CODE BEGIN ADC1_Init 0 */

/* USER CODE END ADC1_Init 0 */

ADC_ChannelConfTypeDef sConfig = {0};

/* USER CODE BEGIN ADC1_Init 1 */

/* USER CODE END ADC1_Init 1 */

/** Configure the global features of the ADC (Clock, Resolution, Data Alignment and number of
conversion)
*/
hadc1.Instance = ADC1;
```



```

hadc1.Init.ClockPrescaler = ADC_CLOCK_SYNC_PCLK_DIV4;
hadc1.Init.Resolution = ADC_RESOLUTION_12B;
hadc1.Init.ScanConvMode = ENABLE;
hadc1.Init.ContinuousConvMode = DISABLE;
hadc1.Init.DiscontinuousConvMode = DISABLE;
hadc1.Init.ExternalTrigConvEdge = ADC_EXTERNALTRIGCONVEDGE_RISING;
hadc1.Init.ExternalTrigConv = ADC_EXTERNALTRIGCONV_T3_TRGO;
hadc1.Init.DataAlign = ADC_DATAALIGN_RIGHT;
hadc1.Init.NbrOfConversion = 2;
hadc1.Init.DMAContinuousRequests = ENABLE;
hadc1.Init.EOCSelection = ADC_EOC_SINGLE_CONV;
if (HAL_ADC_Init(&hadc1) != HAL_OK)
{
    Error_Handler();
}

/** Configure for the selected ADC regular channel its corresponding rank in the sequencer and its
sample time.
*/
sConfig.Channel = ADC_CHANNEL_6;
sConfig.Rank = 1;
sConfig.SamplingTime = ADC_SAMPLETIME_3CYCLES;
if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
{
    Error_Handler();
}

/** Configure for the selected ADC regular channel its corresponding rank in the sequencer and its
sample time.
*/
sConfig.Rank = 2;
if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
{
    Error_Handler();
} si pongo esto manualmente luego abra problema ? sConfig.Channel = ADC_CHANNEL_7; o no
entiendo por que no activa el canal 7

```

Gracias por compartir el fragmento completo. El problema es claro: **tu segundo canal sigue siendo el mismo ADC_CHANNEL_6**, porque no cambiaste `sConfig.Channel` antes de asignar `Rank = 2`.

Error actual (lo que tienes):

```

c

sConfig.Channel = ADC_CHANNEL_6;
sConfig.Rank = 1;
HAL_ADC_ConfigChannel(&hadc1, &sConfig);

sConfig.Rank = 2; // ❌ Aquí no cambiaste el canal, sigue siendo 6
HAL_ADC_ConfigChannel(&hadc1, &sConfig);

```

En este caso, lo que sucede es que configuras **dos veces el canal 6** (VRx), y el canal 7 (VRy) **nunca se configura**, por lo tanto `vry_buffer[]` tiene basura o el mismo valor duplicado.

✓ Solución correcta (lo que debes poner):

```
c
// Canal 6 - VRx
sConfig.Channel = ADC_CHANNEL_6;
sConfig.Rank = 1;
sConfig.SamplingTime = ADC_SAMPLETIME_3CYCLES;
if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK) {
    Error_Handler();
}

// Canal 7 - VRy
sConfig.Channel = ADC_CHANNEL_7; // ✓ CAMBIA AQUÍ el canal
sConfig.Rank = 2;
sConfig.SamplingTime = ADC_SAMPLETIME_3CYCLES;
if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK) {
    Error_Handler();
}
```

✓ ¿Puedes editar esto manualmente?

Sí, puedes y debes editarlo manualmente si STM32CubeMX no lo hace correctamente. Sólo asegúrate de:

- Mantener bien los canales (`ADC_CHANNEL_6` y `ADC_CHANNEL_7`)
- Mantener `NbrOfConversion = 2`
- Mantener `ScanConvMode = ENABLE`
- Usar `DMAContinuousRequests = ENABLE` (para DMA)

STM32CubeMX a veces **no actualiza automáticamente el segundo canal** si agregas ambos canales desde el configurador y luego exportas el código. Por eso es válido editarlo directamente en `main.c`.

✓ Verifica también el pinout:

- **Canal 6** → `ADC1_IN6` → Pin **PA6**
- **Canal 7** → `ADC1_IN7` → Pin **PA7**

Asegúrate de que tu joystick esté conectado a esos pines físicamente.

¿Deseas que revise también cómo se hace la llamada a `HAL_ADC_Start_DMA()` en tu `main()` para asegurar que todo funcione bien con el buffer?



genial ahora funciona quitemos lo del promedio e invirtamos como se grafica por que X está en los dos primeros dígitos del display de la derecha XXYY y necesito YYXX :/* USER CODE BEGIN Header

```

*/
/**
*****
* @file      : main.c
* @brief     : Main program body
*****
* @attention
*
* Copyright (c) 2025 STMicroelectronics.
* All rights reserved.
*
* This software is licensed under terms that can be found in the LICENSE file
* in the root directory of this software component.
* If no LICENSE file comes with this software, it is provided AS-IS.
*
*****
*/
/* USER CODE END Header */
/* Includes -----*/
#include "main.h"

/* Private includes -----*/
/* USER CODE BEGIN Includes */
#include "fsm.h"
#include "stdint.h"
#include "stdio.h"
#include "string.h"
#include <stdlib.h>
/* USER CODE END Includes */

/* Private typedef -----*/
/* USER CODE BEGIN PTD */

/* USER CODE END PTD */

/* Private define -----*/
/* USER CODE BEGIN PD */

/* USER CODE END PD */

/* Private macro -----*/
/* USER CODE BEGIN PM */

/* USER CODE END PM */

/* Private variables -----*/
ADC_HandleTypeDef hadc1;
DMA_HandleTypeDef hdma_adc1;

TIM_HandleTypeDef htim1;
TIM_HandleTypeDef htim2;

```

```

TIM_HandleTypeDef htim3;
TIM_HandleTypeDef htim4;
DMA_HandleTypeDef hdma_tim3_ch2;

UART_HandleTypeDef huart2;

/* USER CODE BEGIN PV */
const uint32_t timer_clk = 84000000UL;
static uint16_t buffer_size = 1024;
static uint16_t display_value = 0;
static uint8_t nextDigit_FSM = 1;
static uint8_t tx_buffer[256] = {0};
#define FREQ_BUFFER_SIZE 16
float freq_buffer[FREQ_BUFFER_SIZE] = {0};
#define NUM_MUESTRAS 1024
uint16_t adc_buffer[2 * NUM_MUESTRAS];
uint16_t vrx_buffer[NUM_MUESTRAS];
uint16_t vry_buffer[NUM_MUESTRAS];

char rx_buffer[128] = {0};
uint8_t rx_char = 0;
uint8_t rx_index=0;
volatile e_PosiblesEvents pending_event = IDLE;
volatile uint8_t data_snapshot = 0;

/* USER CODE END PV */

/* Private function prototypes -----*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_DMA_Init(void);
static void MX_USART2_UART_Init(void);
static void MX_TIM2_Init(void);
static void MX_TIM4_Init(void);
static void MX_ADC1_Init(void);
static void MX_TIM3_Init(void);
static void MX_TIM1_Init(void);
/* USER CODE BEGIN PFP */
void displayNumber(uint8_t digitValue);
void HandleLEDDelayCmd(const char *arg);
void HandleSampleFreqCmd(const char *arg);
void HandleRGBCmd(const char *arg);
void HandlePWMFreqCmd(const char *arg);
void HandleUnknownCmd(void);
void HandleStatusCmd(void);
void HandlePrintADC(void);
void HandleClearCmd(void);
/* USER CODE END PFP */

/* Private user code -----*/
/* USER CODE BEGIN 0 */

```

```
e_PosiblesStates state_machine_action(e_PosiblesEvents event) {
    switch (event) {
        case IDLE:
            break;
        case EVENT_ENCODER:
            if (data_snapshot) // DATA snapshot
                display_value = (display_value == 0) ? 4095 : display_value - 1;
            else
                display_value = (display_value == 4095) ? 0 : display_value + 1;
            break;

        case EVENT_SW:
            display_value = 0;
            nextDigit_FSM = 1;
            break;

        case EVENT_TIMER_TICK:{
            uint32_t suma_x = 0, suma_y = 0;
            for (int i = 0; i < NUM_MUESTRAS; i++) {
                suma_x += vr_x_buffer[i];
                suma_y += vr_y_buffer[i];
            }
            uint16_t prom_x = suma_x / NUM_MUESTRAS;
            uint16_t prom_y = suma_y / NUM_MUESTRAS;

            uint8_t x_val = (prom_x * 99) / 4095;
            uint8_t y_val = (prom_y * 99) / 4095;
            display_value = (y_val * 100) + x_val;
            // Apagar todos los dígitos
            HAL_GPIO_WritePin(GPIOA, GPIO_PIN_0, GPIO_PIN_RESET); // userDis1
            HAL_GPIO_WritePin(GPIOA, GPIO_PIN_1, GPIO_PIN_RESET); // userDis2
            HAL_GPIO_WritePin(GPIOB, GPIO_PIN_0, GPIO_PIN_RESET); // userDis3
            HAL_GPIO_WritePin(GPIOA, GPIO_PIN_4, GPIO_PIN_RESET); // userDis4

            switch (nextDigit_FSM) {
                case 1:
                    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_0, GPIO_PIN_SET);
                    displayNumber(display_value % 10);
                    break;
                case 2:
                    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_1, GPIO_PIN_SET);
                    displayNumber((display_value / 10) % 10);
                    break;
                case 3:
                    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_0, GPIO_PIN_SET);
                    displayNumber((display_value / 100) % 10);
                    break;
                case 4:
                    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_4, GPIO_PIN_SET);
                    displayNumber((display_value / 1000) % 10);
            }
        }
    }
}
```

```

        break;
    }
    nextDigit_FSM = (nextDigit_FSM < 4) ? nextDigit_FSM + 1 : 1;
    break;}
case EVENT_USART: {
    // eco
    int len = snprintf((char*)tx_buffer, sizeof(tx_buffer),
        "Comando recibido: %s\r\n", rx_buffer);
    HAL_UART_Transmit(&huart2, tx_buffer, len, 1000);

    // dispatch
    if (strncmp((char*)rx_buffer,"led=",4)==0)  HandleLEDDelayCmd(rx_buffer+4);
    else if (strncmp((char*)rx_buffer,"fmuestreo=",10)==0)
        HandleSampleFreqCmd(rx_buffer+10);
    else if (strncmp((char*)rx_buffer,"rgb=",4)==0)  HandleRGBCmd(rx_buffer+4);
    else if (strncmp((char*)rx_buffer,"status",6)==0) HandleStatusCmd();
    else if (strncmp((char*)rx_buffer,"help",4)==0) HandleHelpCmd();
    else if (strncmp((char*)rx_buffer, "clear", 5) == 0) HandleClearCmd();
    else
        ;

    memset(rx_buffer,0,sizeof(rx_buffer));
    break;
}return event;
}
}

void displayNumber(uint8_t digitValue) {
    uint32_t start = DWT->CYCCNT;
    static uint8_t contador_muestras = 0;
    // Apagar todos los segmentos
    HAL_GPIO_WritePin(GPIOC, GPIO_PIN_4, GPIO_PIN_RESET); // A
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_10, GPIO_PIN_RESET); // B
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_13, GPIO_PIN_RESET); // C
    HAL_GPIO_WritePin(GPIOC, GPIO_PIN_0, GPIO_PIN_RESET); // D
    HAL_GPIO_WritePin(GPIOC, GPIO_PIN_1, GPIO_PIN_RESET); // E
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_1, GPIO_PIN_RESET); // F
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_5, GPIO_PIN_RESET); // G

    switch (digitValue) {
        case 0:
            HAL_GPIO_WritePin(GPIOC, GPIO_PIN_4, GPIO_PIN_SET);
            HAL_GPIO_WritePin(GPIOB, GPIO_PIN_10, GPIO_PIN_SET);
            HAL_GPIO_WritePin(GPIOB, GPIO_PIN_13, GPIO_PIN_SET);
            HAL_GPIO_WritePin(GPIOC, GPIO_PIN_0, GPIO_PIN_SET);
            HAL_GPIO_WritePin(GPIOC, GPIO_PIN_1, GPIO_PIN_SET);
            HAL_GPIO_WritePin(GPIOB, GPIO_PIN_1, GPIO_PIN_SET);
            break;
        case 1:
            HAL_GPIO_WritePin(GPIOB, GPIO_PIN_10, GPIO_PIN_SET);
            HAL_GPIO_WritePin(GPIOB, GPIO_PIN_13, GPIO_PIN_SET);
            break;
    }
}

```

```

case 2:
    HAL_GPIO_WritePin(GPIOC, GPIO_PIN_4, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_10, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOC, GPIO_PIN_0, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOC, GPIO_PIN_1, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_5, GPIO_PIN_SET);
    break;
case 3:
    HAL_GPIO_WritePin(GPIOC, GPIO_PIN_4, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_10, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_13, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOC, GPIO_PIN_0, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_5, GPIO_PIN_SET);
    break;
case 4:
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_10, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_13, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_1, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_5, GPIO_PIN_SET);
    break;
case 5:
    HAL_GPIO_WritePin(GPIOC, GPIO_PIN_4, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_13, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOC, GPIO_PIN_0, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_1, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_5, GPIO_PIN_SET);
    break;
case 6:
    HAL_GPIO_WritePin(GPIOC, GPIO_PIN_4, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_13, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOC, GPIO_PIN_0, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOC, GPIO_PIN_1, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_1, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_5, GPIO_PIN_SET);
    break;
case 7:
    HAL_GPIO_WritePin(GPIOC, GPIO_PIN_4, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_10, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_13, GPIO_PIN_SET);
    break;
case 8:
    HAL_GPIO_WritePin(GPIOC, GPIO_PIN_4, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_10, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_13, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOC, GPIO_PIN_0, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOC, GPIO_PIN_1, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_1, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_5, GPIO_PIN_SET);
    break;
case 9:
    HAL_GPIO_WritePin(GPIOC, GPIO_PIN_4, GPIO_PIN_SET);

```

```

    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_10, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_13, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_1, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_5, GPIO_PIN_SET);
    break;
}
uint32_t elapsed = DWT->CYCCNT - start;

if (contador_muestras < 10) {
    printf("Ciclos para mostrar digito %d: %lu\n", digitValue, (unsigned long)elapsed);
    contador_muestras++;
}
}

void HandleLEDDelayCmd(const char *arg) {
    uint32_t nuevo = atoi(arg);
    // Ajusta TIM2 para el blinky
    __HAL_TIM_SET_AUTORELOAD(&htim2, nuevo);
    __HAL_TIM_SET_COUNTER(&htim2, 0);
    // Feedback
    int len = snprintf((char*)tx_buffer, sizeof(tx_buffer),
        "LED delay = %lu ms\r\n", (unsigned long)nuevo);
    HAL_UART_Transmit(&huart2, tx_buffer, len, 1000);
}

void HandleSampleFreqCmd(const char *arg) {
    uint32_t fs;
    char option = arg[0];
    // Verifica la opción seleccionada (1, 2, 3, 4)
    switch(option) {
        case '1': fs = 44100; break;
        case '2': fs = 48000; break;
        case '3': fs = 96000; break;
        case '4': fs = 128000; break;
        default: {
            const char *msg = "Opciones válidas para 'fmuestreo=' son: 1, 2, 3, 4\r\n";
            HAL_UART_Transmit(&huart2, (uint8_t*)msg, strlen(msg), 1000);
            return;
        }
    }

    // calcular ARR y PSC
    uint32_t arr = timer_clk / fs - 1;
    uint32_t psc = 0;
    if (arr > 0xFFFF) {
        // escalar PSC si supera 16 bits
        psc = (arr / 0x10000) + 1;
        arr = (timer_clk / (psc+1) / fs) - 1;
    }

    // aplicar a TIM3 y reiniciar
    __HAL_TIM_SET_PRESCALER(&htim3, psc);
    __HAL_TIM_SET_AUTORELOAD(&htim3, arr);
    __HAL_TIM_SET_COUNTER(&htim3, 0);
}

```



```

HAL_TIM_Base_Stop(&htim3);
HAL_TIM_Base_Start(&htim3);
// Feedback
int len = snprintf((char*)tx_buffer, sizeof(tx_buffer),
    "Sample TIM3 @ %lu Hz (PSC=%lu, ARR=%lu)\r\n",
    (unsigned long)fs, (unsigned long)psc, (unsigned long)arr);
HAL_UART_Transmit(&huart2, tx_buffer, len, 1000);
}

void HandleRGBCmd(const char *arg) {
    uint32_t r, g, b;
    if (sscanf(arg, "%lu %lu %lu", &r, &g, &b) == 3) {
        if (r > 255) r = 255;
        if (g > 255) g = 255;
        if (b > 255) b = 255;

        __HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_1, g); // G
        __HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_2, b); // B
        __HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_3, r); // R

        int len = snprintf((char*)tx_buffer, sizeof(tx_buffer),
            "RGB PWM -> R=%lu, G=%lu, B=%lu\r\n", r, g, b);
        HAL_UART_Transmit(&huart2, tx_buffer, len, 1000);
    } else {
        const char *msg = "Formato inválido. Usa: rgb=R G B@\r\n";
        HAL_UART_Transmit(&huart2, (uint8_t*)msg, strlen(msg), 1000);
    }
}

void HandleStatusCmd(void) {
    uint32_t psc = htim3.Init.Prescaler;
    uint32_t arr = htim3.Init.Period;
    float fs = 84000000.0f / ((psc + 1) * (arr + 1));

    int len = snprintf((char*)tx_buffer, sizeof(tx_buffer),
        "Config:\r\nSample TIM3 @ %.2f Hz (PSC=%lu, ARR=%lu)\r\nCanal ADC: 6 y 7\r\n",
        fs, (unsigned long)psc, (unsigned long)arr);
    HAL_UART_Transmit(&huart2, tx_buffer, len, 1000);
}

void HandleClearCmd(void) {
    const char *clear_screen = "\033[2J\033[H"; // Secuencia ANSI para limpiar la pantalla
    HAL_UART_Transmit(&huart2, (uint8_t*)clear_screen, strlen(clear_screen), 1000);
}

void HandleHelpCmd(void) {
    const char *help_msg =

```

```

"\r\n===== AYUDA ====="
"\r\nComandos disponibles:\r\n"
" led=<ms>@ - Cambia la frecuencia del LED Blinky\r\n"
" fmuestreo=<1|2|3|4>@ - Frecuencia de muestreo del ADC:\r\n"
"     - 1 -> 44100 Hz\r\n"
"     - 2 -> 48000 Hz\r\n"
"     - 3 -> 96000 Hz\r\n"
"     - 4 -> 128000 Hz\r\n"
" rgb=<RGB>@ - Control de LED RGB, ej: rgb=RG\r\n"
" status@ - Mostrar configuración actual\r\n"
" print@ - Imprimir datos crudos del ADC\r\n"
" help @ - Mostrar esta ayuda\r\n"
"===== \r\n";
HAL_UART_Transmit(&huart2, (uint8_t*)help_msg, strlen(help_msg), 1000);
}

void imprimirBinario(uint8_t valor) {
    for (int i = 7; i >= 0; i--) {
        printf("%d", (valor >> i) & 1);
    }
}

void mostrarOperacion(const char *desc, uint8_t resultado) {
    printf("%s\n", desc);
    printf("Binario: "); imprimirBinario(resultado);
    printf("\nHex: 0x%02X\n", resultado);
    printf("ASCII: %c\n\n", (resultado >= 32 && resultado <= 126) ? resultado : '.');
}

void verificarOperacionesCrucigrama(void) {
    // 1. Caracter ASCII 'B'
    uint8_t op1 = 'B';
    mostrarOperacion("1. ASCII 'B'", op1);

    // 2. ((!!0xC) << 0b11) | (~0x45)
    uint8_t op2 = ((!!0xC) << 0b11) | (~0x45);
    mostrarOperacion("2. ((!!0xC) << 0b11) | (~0x45)", op2);

    // 3. ((0b00111100 & 0xAC) << 0x3)
    uint8_t op3 = (0b00111100 & 0xAC) << 0x3;
    mostrarOperacion("3. ((0b00111100 & 0xAC) << 0x3)", op3);

    // 4. -121
    uint8_t op4 = (uint8_t)(-121);
    mostrarOperacion("4. -121 (uint8_t)", op4);

    // 5. (0xCAFE & 0xFF0) >> 7
    uint16_t val5 = (0xCAFE & 0xFF0) >> 7;
    uint8_t op5 = (uint8_t)(val5 & 0xFF);
    mostrarOperacion("5. (0xCAFE & 0xFF0) >> 7", op5);

    // 6. (1 << 4) + (0x3 << 4) - 10 + (0b11 << 0x6)

```

```

uint8_t op6 = (1 << 4) + (0x3 << 4) - 10 + (0b11 << 6);
mostrarOperacion("6. (1<<4)+(0x3<<4)-10+(0b11<<6)", op6);

// 7. ('T' | '5')
uint8_t op7 = 'T' | '5';
mostrarOperacion("7. ('T' | '5')", op7);

// 8. (2921 & 0xD6)
uint8_t op8 = 2921 & 0xD6;
mostrarOperacion("8. (2921 & 0xD6)", op8);

// 9. 01100110 ^ 00101011
uint8_t a9 = 0b01100110;
uint8_t b9 = 0b00101011;
uint8_t op9 = a9 ^ b9;
mostrarOperacion("9. 01100110 ^ 00101011", op9);
}
/* USER CODE END 0 */

/**
 * @brief The application entry point.
 * @retval int
 */
int main(void)
{

    /* USER CODE BEGIN 1 */

    /* USER CODE END 1 */

    /* MCU Configuration-----*/

    /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
    HAL_Init();

    /* USER CODE BEGIN Init */
    CoreDebug->DEMCR |= CoreDebug_DEMCR_TRCENA_Msk; // Habilitar DWT
    DWT->CYCCNT = 0; // Reiniciar contador
    DWT->CTRL |= DWT_CTRL_CYCCNTENA_Msk; // Encender contador de ciclos
    /* USER CODE END Init */

    /* Configure the system clock */
    SystemClock_Config();

    /* USER CODE BEGIN SysInit */

    /* USER CODE END SysInit */

    /* Initialize all configured peripherals */
    MX_GPIO_Init();
    MX_DMA_Init();

```

```

MX_USART2_UART_Init();
MX_TIM2_Init();
MX_TIM4_Init();
MX_ADC1_Init();
MX_TIM3_Init();
MX_TIM1_Init();
/* USER CODE BEGIN 2 */
HAL_TIM_Base_Start_IT(&htim2);
HAL_TIM_Base_Start_IT(&htim3);
HAL_TIM_Base_Start_IT(&htim4);
HAL_UART_Receive_IT(&huart2,&rx_char,1);
HAL_ADC_Start_DMA(&hadc1, (uint32_t*)adc_buffer, buffer_size);
HAL_TIM_IC_Start_IT(&htim3, TIM_CHANNEL_2);
HandleSampleFreqCmd(3); //96000 hz muestreo

verificarOperacionesCrucigrama();
medirTiempoEnvioUART();
HandleHelpCmd();

/* USER CODE END 2 */

/* Infinite loop */
/* USER CODE BEGIN WHILE */
while (1)
{
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
    if (pending_event != IDLE){
        state_machine_action(pending_event);
        pending_event = IDLE;
    }
}
/* USER CODE END 3 */
}

/**
 * @brief System Clock Configuration
 * @retval None
 */
void SystemClock_Config(void)
{
    RCC_OscInitTypeDef RCC_OscInitStruct = {0};
    RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};

    /** Configure the main internal regulator output voltage
     */
    __HAL_RCC_PWR_CLK_ENABLE();
    __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE1);

    /** Initializes the RCC Oscillators according to the specified parameters

```

```

* in the RCC_OscInitTypeDef structure.
*/
RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI;
RCC_OscInitStruct.HSISState = RCC_HSI_ON;
RCC_OscInitStruct.HSICalibrationValue = RCC_HSICALIBRATION_DEFAULT;
RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSI;
RCC_OscInitStruct.PLL.PLLM = 16;
RCC_OscInitStruct.PLL.PLLN = 336;
RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV4;
RCC_OscInitStruct.PLL.PLLQ = 4;
if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
{
    Error_Handler();
}

/** Initializes the CPU, AHB and APB buses clocks
*/
RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSClk
    |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
RCC_ClkInitStruct.SYSClkSource = RCC_SYSClkSOURCE_PLLCLK;
RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSClk_DIV1;
RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV2;
RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;

if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_2) != HAL_OK)
{
    Error_Handler();
}
}

/**
 * @brief ADC1 Initialization Function
 * @param None
 * @retval None
 */
static void MX_ADC1_Init(void)
{

    /* USER CODE BEGIN ADC1_Init 0 */

    /* USER CODE END ADC1_Init 0 */

    ADC_ChannelConfTypeDef sConfig = {0};

    /* USER CODE BEGIN ADC1_Init 1 */

    /* USER CODE END ADC1_Init 1 */

    /** Configure the global features of the ADC (Clock, Resolution, Data Alignment and number of
    conversion)

```

```

*/
hadc1.Instance = ADC1;
hadc1.Init.ClockPrescaler = ADC_CLOCK_SYNC_PCLK_DIV4;
hadc1.Init.Resolution = ADC_RESOLUTION_12B;
hadc1.Init.ScanConvMode = ENABLE;
hadc1.Init.ContinuousConvMode = DISABLE;
hadc1.Init.DiscontinuousConvMode = DISABLE;
hadc1.Init.ExternalTrigConvEdge = ADC_EXTERNALTRIGCONVEDGE_RISING;
hadc1.Init.ExternalTrigConv = ADC_EXTERNALTRIGCONV_T3_TRGO;
hadc1.Init.DataAlign = ADC_DATAALIGN_RIGHT;
hadc1.Init.NbrOfConversion = 2;
hadc1.Init.DMAContinuousRequests = ENABLE;
hadc1.Init.EOCSelection = ADC_EOC_SINGLE_CONV;
if (HAL_ADC_Init(&hadc1) != HAL_OK)
{
    Error_Handler();
}

/** Configure for the selected ADC regular channel its corresponding rank in the sequencer and its
sample time.
*/
sConfig.Channel = ADC_CHANNEL_6;
sConfig.Rank = 1;
sConfig.SamplingTime = ADC_SAMPLETIME_3CYCLES;
if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
{
    Error_Handler();
}

/** Configure for the selected ADC regular channel its corresponding rank in the sequencer and its
sample time.
*/
sConfig.Channel = ADC_CHANNEL_7;
sConfig.Rank = 2;
if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
{
    Error_Handler();
}
/* USER CODE BEGIN ADC1_Init 2 */

/* USER CODE END ADC1_Init 2 */

}

/**
 * @brief TIM1 Initialization Function
 * @param None
 * @retval None
 */
static void MX_TIM1_Init(void)
{

```

```

/* USER CODE BEGIN TIM1_Init 0 */

/* USER CODE END TIM1_Init 0 */

TIM_MasterConfigTypeDef sMasterConfig = {0};
TIM_OC_InitTypeDef sConfigOC = {0};
TIM_BreakDeadTimeConfigTypeDef sBreakDeadTimeConfig = {0};

/* USER CODE BEGIN TIM1_Init 1 */

/* USER CODE END TIM1_Init 1 */
htim1.Instance = TIM1;
htim1.Init.Prescaler = 83;
htim1.Init.CounterMode = TIM_COUNTERMODE_UP;
htim1.Init.Period = 255;
htim1.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
htim1.Init.RepetitionCounter = 0;
htim1.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
if (HAL_TIM_PWM_Init(&htim1) != HAL_OK)
{
    Error_Handler();
}
sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
if (HAL_TIMEx_MasterConfigSynchronization(&htim1, &sMasterConfig) != HAL_OK)
{
    Error_Handler();
}
sConfigOC.OCMode = TIM_OCMODE_PWM1;
sConfigOC.Pulse = 0;
sConfigOC.OCpolarity = TIM_OCPOLARITY_HIGH;
sConfigOC.OCNPolarity = TIM_OCNPOLARITY_HIGH;
sConfigOC.OCFastMode = TIM_OCFAST_DISABLE;
sConfigOC.OCIdleState = TIM_OCIDLESTATE_RESET;
sConfigOC.OCNIdleState = TIM_OCNIDLESTATE_RESET;
if (HAL_TIM_PWM_ConfigChannel(&htim1, &sConfigOC, TIM_CHANNEL_1) != HAL_OK)
{
    Error_Handler();
}
if (HAL_TIM_PWM_ConfigChannel(&htim1, &sConfigOC, TIM_CHANNEL_2) != HAL_OK)
{
    Error_Handler();
}
if (HAL_TIM_PWM_ConfigChannel(&htim1, &sConfigOC, TIM_CHANNEL_3) != HAL_OK)
{
    Error_Handler();
}
sBreakDeadTimeConfig.OffStateRunMode = TIM_OSSR_DISABLE;
sBreakDeadTimeConfig.OffStateIDLEMode = TIM_OSSI_DISABLE;
sBreakDeadTimeConfig.LockLevel = TIM_LOCKLEVEL_OFF;

```

```

sBreakDeadTimeConfig.DeadTime = 0;
sBreakDeadTimeConfig.BreakState = TIM_BREAK_DISABLE;
sBreakDeadTimeConfig.BreakPolarity = TIM_BREAKPOLARITY_HIGH;
sBreakDeadTimeConfigAutomaticOutput = TIM_AUTOMATICOUTPUT_DISABLE;
if (HAL_TIMEx_ConfigBreakDeadTime(&htim1, &sBreakDeadTimeConfig) != HAL_OK)
{
    Error_Handler();
}
/* USER CODE BEGIN TIM1_Init 2 */
HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_1); // PA8 - g
HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_2); // PA9 - B
HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_3); // PA10 - R
/* USER CODE END TIM1_Init 2 */
HAL_TIM_MspPostInit(&htim1);

}

/**
 * @brief TIM2 Initialization Function
 * @param None
 * @retval None
 */
static void MX_TIM2_Init(void)
{

    /* USER CODE BEGIN TIM2_Init 0 */

    /* USER CODE END TIM2_Init 0 */

    TIM_ClockConfigTypeDef sClockSourceConfig = {0};
    TIM_MasterConfigTypeDef sMasterConfig = {0};

    /* USER CODE BEGIN TIM2_Init 1 */

    /* USER CODE END TIM2_Init 1 */
    htim2.Instance = TIM2;
    htim2.Init.Prescaler = 42000-1;
    htim2.Init.CounterMode = TIM_COUNTERMODE_UP;
    htim2.Init.Period = 500;
    htim2.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
    htim2.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_ENABLE;
    if (HAL_TIM_Base_Init(&htim2) != HAL_OK)
    {
        Error_Handler();
    }
    sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
    if (HAL_TIM_ConfigClockSource(&htim2, &sClockSourceConfig) != HAL_OK)
    {
        Error_Handler();
    }
    sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;

```



```

sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
if (HAL_TIMEx_MasterConfigSynchronization(&htim2, &sMasterConfig) != HAL_OK)
{
    Error_Handler();
}
/* USER CODE BEGIN TIM2_Init 2 */

/* USER CODE END TIM2_Init 2 */

}

/**
 * @brief TIM3 Initialization Function
 * @param None
 * @retval None
 */
static void MX_TIM3_Init(void)
{

    /* USER CODE BEGIN TIM3_Init 0 */

    /* USER CODE END TIM3_Init 0 */

    TIM_ClockConfigTypeDef sClockSourceConfig = {0};
    TIM_MasterConfigTypeDef sMasterConfig = {0};
    TIM_IC_InitTypeDef sConfigIC = {0};

    /* USER CODE BEGIN TIM3_Init 1 */

    /* USER CODE END TIM3_Init 1 */
    htim3.Instance = TIM3;
    htim3.Init.Prescaler = 0;
    htim3.Init.CounterMode = TIM_COUNTERMODE_UP;
    htim3.Init.Period = 65535;
    htim3.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
    htim3.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
    if (HAL_TIM_Base_Init(&htim3) != HAL_OK)
    {
        Error_Handler();
    }
    sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
    if (HAL_TIM_ConfigClockSource(&htim3, &sClockSourceConfig) != HAL_OK)
    {
        Error_Handler();
    }
    if (HAL_TIM_IC_Init(&htim3) != HAL_OK)
    {
        Error_Handler();
    }
    sMasterConfig.MasterOutputTrigger = TIM_TRGO_UPDATE;
    sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;

```

```

if (HAL_TIMEx_MasterConfigSynchronization(&htim3, &sMasterConfig) != HAL_OK)
{
    Error_Handler();
}
sConfigIC.ICPolarity = TIM_INPUTCHANNELPOLARITY_RISING;
sConfigIC.ICSelection = TIM_ICSELECTION_DIRECTTI;
sConfigIC.ICPrescaler = TIM_ICPSC_DIV1;
sConfigIC.ICFilter = 8;
if (HAL_TIM_IC_ConfigChannel(&htim3, &sConfigIC, TIM_CHANNEL_2) != HAL_OK)
{
    Error_Handler();
}
/* USER CODE BEGIN TIM3_Init 2 */
HAL_TIM_IC_Start_IT(&htim3, TIM_CHANNEL_2);
/* USER CODE END TIM3_Init 2 */

}

/**
 * @brief TIM4 Initialization Function
 * @param None
 * @retval None
 */
static void MX_TIM4_Init(void)
{

    /* USER CODE BEGIN TIM4_Init 0 */

    /* USER CODE END TIM4_Init 0 */

    TIM_ClockConfigTypeDef sClockSourceConfig = {0};
    TIM_MasterConfigTypeDef sMasterConfig = {0};

    /* USER CODE BEGIN TIM4_Init 1 */

    /* USER CODE END TIM4_Init 1 */
    htim4.Instance = TIM4;
    htim4.Init.Prescaler = 840-1;
    htim4.Init.CounterMode = TIM_COUNTERMODE_UP;
    htim4.Init.Period = 200-1;
    htim4.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
    htim4.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
    if (HAL_TIM_Base_Init(&htim4) != HAL_OK)
    {
        Error_Handler();
    }
    sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
    if (HAL_TIM_ConfigClockSource(&htim4, &sClockSourceConfig) != HAL_OK)
    {
        Error_Handler();
    }
}

```

```
sMasterConfig.MasterOutputTrigger = TIM_TRGO_UPDATE;
sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
if (HAL_TIMEx_MasterConfigSynchronization(&htim4, &sMasterConfig) != HAL_OK)
{
    Error_Handler();
}
/* USER CODE BEGIN TIM4_Init 2 */

/* USER CODE END TIM4_Init 2 */

}

/**
 * @brief USART2 Initialization Function
 * @param None
 * @retval None
 */
static void MX_USART2_UART_Init(void)
{

    /* USER CODE BEGIN USART2_Init 0 */

    /* USER CODE END USART2_Init 0 */

    /* USER CODE BEGIN USART2_Init 1 */

    /* USER CODE END USART2_Init 1 */
    huart2.Instance = USART2;
    huart2.Init.BaudRate = 115200;
    huart2.Init.WordLength = UART_WORDLENGTH_8B;
    huart2.Init.StopBits = UART_STOPBITS_1;
    huart2.Init.Parity = UART_PARITY_NONE;
    huart2.Init.Mode = UART_MODE_TX_RX;
    huart2.Init.HwFlowCtl = UART_HWCONTROL_NONE;
    huart2.Init.OverSampling = UART_OVERSAMPLING_16;
    if (HAL_UART_Init(&huart2) != HAL_OK)
    {
        Error_Handler();
    }
    /* USER CODE BEGIN USART2_Init 2 */

    /* USER CODE END USART2_Init 2 */

}

/**
 * Enable DMA controller clock
 */
static void MX_DMA_Init(void)
{
```

```

/* DMA controller clock enable */
__HAL_RCC_DMA2_CLK_ENABLE();
__HAL_RCC_DMA1_CLK_ENABLE();

/* DMA interrupt init */
/* DMA1_Stream5_IRQn interrupt configuration */
HAL_NVIC_SetPriority(DMA1_Stream5_IRQn, 0, 0);
HAL_NVIC_EnableIRQ(DMA1_Stream5_IRQn);
/* DMA2_Stream0_IRQn interrupt configuration */
HAL_NVIC_SetPriority(DMA2_Stream0_IRQn, 0, 0);
HAL_NVIC_EnableIRQ(DMA2_Stream0_IRQn);

}

/**
 * @brief GPIO Initialization Function
 * @param None
 * @retval None
 */
static void MX_GPIO_Init(void)
{
    GPIO_InitTypeDef GPIO_InitStruct = {0};
    /* USER CODE BEGIN MX_GPIO_Init_1 */
    /* USER CODE END MX_GPIO_Init_1 */

    /* GPIO Ports Clock Enable */
    __HAL_RCC_GPIOC_CLK_ENABLE();
    __HAL_RCC_GPIOH_CLK_ENABLE();
    __HAL_RCC_GPIOA_CLK_ENABLE();
    __HAL_RCC_GPIOB_CLK_ENABLE();

    /*Configure GPIO pin Output Level */
    HAL_GPIO_WritePin(GPIOC, LedB_Pin | segD_Pin | segE_Pin | LedR_Pin
        | LedG_Pin | segA_Pin, GPIO_PIN_RESET);

    /*Configure GPIO pin Output Level */
    HAL_GPIO_WritePin(userLed_GPIO_Port, userLed_Pin, GPIO_PIN_SET);

    /*Configure GPIO pin Output Level */
    HAL_GPIO_WritePin(GPIOA, dis1_Pin | dis2_Pin | dis4_Pin, GPIO_PIN_RESET);

    /*Configure GPIO pin Output Level */
    HAL_GPIO_WritePin(GPIOB, dis3_Pin | segF_Pin | segB_Pin | segC_Pin
        | segG_Pin, GPIO_PIN_RESET);

    /*Configure GPIO pins : LedB_Pin segD_Pin segE_Pin LedR_Pin
        LedG_Pin segA_Pin */
    GPIO_InitStruct.Pin = LedB_Pin | segD_Pin | segE_Pin | LedR_Pin
        | LedG_Pin | segA_Pin;
    GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
    GPIO_InitStruct.Pull = GPIO_NOPULL;

```

```

GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);

/*Configure GPIO pin : userLed_Pin */
GPIO_InitStruct.Pin = userLed_Pin;
GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
GPIO_InitStruct.Pull = GPIO_NOPULL;
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
HAL_GPIO_Init(userLed_GPIO_Port, &GPIO_InitStruct);

/*Configure GPIO pins : dis1_Pin dis2_Pin dis4_Pin */
GPIO_InitStruct.Pin = dis1_Pin | dis2_Pin | dis4_Pin;
GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
GPIO_InitStruct.Pull = GPIO_NOPULL;
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

/*Configure GPIO pins : dis3_Pin segF_Pin segB_Pin segC_Pin
segG_Pin */
GPIO_InitStruct.Pin = dis3_Pin | segF_Pin | segB_Pin | segC_Pin
| segG_Pin;
GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
GPIO_InitStruct.Pull = GPIO_NOPULL;
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);

/*Configure GPIO pin : userData_Pin */
GPIO_InitStruct.Pin = userData_Pin;
GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
GPIO_InitStruct.Pull = GPIO_NOPULL;
HAL_GPIO_Init(userData_GPIO_Port, &GPIO_InitStruct);

/*Configure GPIO pin : userSw_Pin */
GPIO_InitStruct.Pin = userSw_Pin;
GPIO_InitStruct.Mode = GPIO_MODE_IT_RISING;
GPIO_InitStruct.Pull = GPIO_NOPULL;
HAL_GPIO_Init(userSw_GPIO_Port, &GPIO_InitStruct);

/*Configure GPIO pin : userClk_Pin */
GPIO_InitStruct.Pin = userClk_Pin;
GPIO_InitStruct.Mode = GPIO_MODE_IT_RISING;
GPIO_InitStruct.Pull = GPIO_NOPULL;
HAL_GPIO_Init(userClk_GPIO_Port, &GPIO_InitStruct);

/* EXTI interrupt init*/
HAL_NVIC_SetPriority(EXTI9_5_IRQn, 0, 0);
HAL_NVIC_EnableIRQ(EXTI9_5_IRQn);

HAL_NVIC_SetPriority(EXTI15_10_IRQn, 0, 0);
HAL_NVIC_EnableIRQ(EXTI15_10_IRQn);

```

```

/* USER CODE BEGIN MX_GPIO_Init_2 */
/* USER CODE END MX_GPIO_Init_2 */
}

/* USER CODE BEGIN 4 */
void medirTiempoEnvioUART(void) {
    const char *msg = "Hola mundo! - Examen Taller V\n";
    uint32_t start = DWT->CYCCNT;

    printf("%s", msg);

    uint32_t elapsed = DWT->CYCCNT - start;
    printf("Ciclos para enviar mensaje UART: %lu\n", (unsigned long)elapsed);
}
int __io_putchar(int ch) {
    HAL_UART_Transmit(&huart2, (uint8_t*)&ch, 1, HAL_MAX_DELAY);
    return ch;
}
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim){
    if(htim->Instance==TIM2){
        HAL_GPIO_TogglePin(userLed_GPIO_Port,userLed_Pin);
    }else if(htim->Instance==TIM4){
        if (pending_event == IDLE)
            pending_event = EVENT_TIMER_TICK;
    }else if (htim->Instance==TIM3){

    }
}
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
{
    if (GPIO_Pin == userClk_Pin) {
        data_snapshot = HAL_GPIO_ReadPin(userData_GPIO_Port, userData_Pin);
        if (pending_event == IDLE)
            pending_event = EVENT_ENCODER;
    } else if (GPIO_Pin == userSw_Pin) {
        if (pending_event == IDLE)
            pending_event = EVENT_SW;
    }
}
void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart){
    if(huart->Instance == USART2){
        if(rx_index<sizeof(rx_buffer)-1){
            rx_buffer[rx_index++]=rx_char;
        }
        if (rx_char == '@') {
            rx_buffer[rx_index]='\0';
            rx_index=0;
            pending_event = EVENT_USART; //aquí levanta tu bandera
        }
        HAL_UART_Receive_IT(huart, &rx_char,1);
    }
}

```

```

    }
}

void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef* hadc)
{
    if (hadc->Instance == ADC1) {
        for (int i = 0; i < NUM_MUESTRAS; i++) {
            vr_x_buffer[i] = adc_buffer[2 * i];    // Canal 6
            vr_y_buffer[i] = adc_buffer[2 * i + 1]; // Canal 7
        }
    }
}

/* USER CODE END 4 */

/**
 * @brief This function is executed in case of error occurrence.
 * @retval None
 */
void Error_Handler(void)
{
    /* USER CODE BEGIN Error_Handler_Debug */
    /* User can add his own implementation to report the HAL error return state */
    __disable_irq();
    while (1)
    {
    }
    /* USER CODE END Error_Handler_Debug */
}

#ifdef USE_FULL_ASSERT
/**
 * @brief Reports the name of the source file and the source line number
 *        where the assert_param error has occurred.
 * @param file: pointer to the source file name
 * @param line: assert_param error line source number
 * @retval None
 */
void assert_failed(uint8_t *file, uint32_t line)
{
    /* USER CODE BEGIN 6 */
    /* User can add his own implementation to report the file name and line number,
       ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */
    /* USER CODE END 6 */
}
#endif /* USE_FULL_ASSERT */

```

Perfecto, para cumplir con tu solicitud:

1. **Eliminar el cálculo del promedio** (`prom_x`, `prom_y`) y usar directamente los valores más recientes del `adc_buffer`.

2. **Invertir el orden del display**, mostrando el eje **Y** a la izquierda (dígitos más significativos) y el eje **X** a la derecha (menos significativos). Actualmente es **XXYY**, se debe cambiar a **YYXX**.

✓ Cambios sugeridos (dentro de `state_machine_action`, en el caso `EVENT_TIMER_TICK`):

Reemplaza este bloque:

```
c
uint32_t suma_x = 0, suma_y = 0;
for (int i = 0; i < NUM_MUESTRAS; i++) {
    suma_x += vrx_buffer[i];
    suma_y += vry_buffer[i];
}
uint16_t prom_x = suma_x / NUM_MUESTRAS;
uint16_t prom_y = suma_y / NUM_MUESTRAS;

uint8_t x_val = (prom_x * 99) / 4095;
uint8_t y_val = (prom_y * 99) / 4095;
display_value = (y_val * 100) + x_val;
```

Por este bloque corregido:

```
c
uint16_t x_val = (vrx_buffer[NUM_MUESTRAS - 1] * 99) / 4095;
uint16_t y_val = (vry_buffer[NUM_MUESTRAS - 1] * 99) / 4095;
display_value = (x_val * 100) + y_val; // Ahora se muestra como YYXX -> pero en realidad es
XXYY, entonces se invierte aquí
```

🔄 ¿Por qué este cambio funciona?

Si el valor `display_value` tiene el formato **XXYY**, se puede invertir a **YYXX** haciendo:

```
c
display_value = (x_val * 100) + y_val; // YYXX en el display
```

🔧 Resultado esperado en display

- Si `x_val = 42`, `y_val = 17` → `display_value = 4217`
 - Display: **4** → Y decena
 - **2** → Y unidad
 - **1** → X decena
 - **7** → X unidad

✓ Resumen de lo modificado

- Se eliminó el promedio (`suma_x`, `suma_y`)
- Se tomó el último valor directamente del DMA (`vrx_buffer[NUM_MUESTRAS - 1]`)
- Se invirtió la codificación del número mostrado para que sea **YYXX**

¿Deseas también imprimir los valores `x_val` y `y_val` por UART para monitoreo?



no quiero esto por que seía confuso decir X val necesito cambiar los pines en donde ? `display_value = (x_val * 100) + y_val;`

Perfecto, para lograr que el valor se visualice como **YYXX** (es decir, **Y** en los dos primeros dígitos *de la derecha* y **X** en los dos últimos *de la izquierda* del display), **solo necesitas invertir la forma en que se calcula `display_value`**. Actualmente tienes:

```
c
display_value = (y_val * 100) + x_val;
```

Esto pone **YY** en los dígitos más significativos (izquierda) y **XX** en los menos (derecha), lo cual es justo lo contrario a lo que necesitas.

✓ Solución:

Simplemente invierte la fórmula:

```
c
display_value = (x_val * 100) + y_val;
```

Con esto:

- Los dos **primeros dígitos de la derecha** mostrarán **YY**.
- Los dos **últimos dígitos de la izquierda** mostrarán **XX**.

¿Dónde hacer el cambio?

Ubica este bloque en tu función `state_machine_action()` dentro del `case EVENT_TIMER_TICK::`

```
c
uint8_t x_val = (prom_x * 99) / 4095;
uint8_t y_val = (prom_y * 99) / 4095;
display_value = (y_val * 100) + x_val; // <-- CAMBIA ESTA LÍNEA
```

Y reemplázala por:

```
c
display_value = (x_val * 100) + y_val;
```

Este cambio **no requiere modificar ningún pin**, ya que el orden de activación de los dígitos en el display multiplexado se mantiene igual. Solo estamos cambiando el número que se desea visualizar.

¿Deseas que también se muestre en pantalla "X=xx, Y=yy" por UART o basta con visualizarlo en el display?



```

ahora no me muestra el valor del joystick en el display por que ? si ahorita si : /* USER CODE BEGIN
Header */
/**
*****
* @file      : main.c
* @brief     : Main program body
*****
* @attention
*
* Copyright (c) 2025 STMicroelectronics.
* All rights reserved.
*
* This software is licensed under terms that can be found in the LICENSE file
* in the root directory of this software component.
* If no LICENSE file comes with this software, it is provided AS-IS.
*
*****
*/
/* USER CODE END Header */
/* Includes -----*/
#include "main.h"

/* Private includes -----*/
/* USER CODE BEGIN Includes */
#include "fsm.h"
#include "stdint.h"
#include "stdio.h"
#include "string.h"
#include <stdlib.h>
/* USER CODE END Includes */

/* Private typedef -----*/
/* USER CODE BEGIN PTD */

/* USER CODE END PTD */

/* Private define -----*/
/* USER CODE BEGIN PD */

/* USER CODE END PD */

/* Private macro -----*/
/* USER CODE BEGIN PM */

/* USER CODE END PM */

```

```

/* Private variables -----*/
ADC_HandleTypeDef hadc1;
DMA_HandleTypeDef hdma_adc1;

TIM_HandleTypeDef htim1;
TIM_HandleTypeDef htim2;
TIM_HandleTypeDef htim3;
TIM_HandleTypeDef htim4;
DMA_HandleTypeDef hdma_tim3_ch2;

UART_HandleTypeDef huart2;

/* USER CODE BEGIN PV */
const uint32_t timer_clk = 84000000UL;
static uint16_t buffer_size = 1024;
static uint16_t display_value = 0;
static uint8_t nextDigit_FSM = 1;
static uint8_t tx_buffer[256] = {0};
#define FREQ_BUFFER_SIZE 16
float freq_buffer[FREQ_BUFFER_SIZE] = {0};
#define NUM_MUESTRAS 1024
uint16_t adc_buffer[2 * NUM_MUESTRAS];
uint16_t vrx_buffer[NUM_MUESTRAS];
uint16_t vry_buffer[NUM_MUESTRAS];

char rx_buffer[128] = {0};
uint8_t rx_char = 0;
uint8_t rx_index=0;
volatile e_PosiblesEvents pending_event = IDLE;
volatile uint8_t data_snapshot = 0;

/* USER CODE END PV */

/* Private function prototypes -----*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_DMA_Init(void);
static void MX_USART2_UART_Init(void);
static void MX_TIM2_Init(void);
static void MX_TIM4_Init(void);
static void MX_ADC1_Init(void);
static void MX_TIM3_Init(void);
static void MX_TIM1_Init(void);
/* USER CODE BEGIN PFP */
void displayNumber(uint8_t digitValue);
void HandleLEDDelayCmd(const char *arg);
void HandleSampleFreqCmd(const char *arg);
void HandleRGBCmd(const char *arg);
void HandlePWMFreqCmd(const char *arg);
void HandleUnknownCmd(void);

```

```

void HandleStatusCmd(void);
void HandlePrintADC(void);
void HandleClearCmd(void);
/* USER CODE END PFP */

/* Private user code -----*/
/* USER CODE BEGIN 0 */

e_PosiblesStates state_machine_action(e_PosiblesEvents event) {
    switch (event) {
        case IDLE:
            break;
        case EVENT_ENCODER:
            if (data_snapshot) // DATA snapshot
                display_value = (display_value == 0) ? 4095 : display_value - 1;
            else
                display_value = (display_value == 4095) ? 0 : display_value + 1;
            break;

        case EVENT_SW:
            display_value = 0;
            nextDigit_FSM = 1;
            break;

        case EVENT_TIMER_TICK:{
            uint16_t x_val = (vr_x_buffer[NUM_MUESTRAS - 1] * 99) / 4095;
            uint16_t y_val = (vry_buffer[NUM_MUESTRAS - 1] * 99) / 4095;
            display_value = (y_val * 100) + x_val;
            // Apagar todos los dígitos
            HAL_GPIO_WritePin(GPIOA, GPIO_PIN_0, GPIO_PIN_RESET); // userDis1
            HAL_GPIO_WritePin(GPIOA, GPIO_PIN_1, GPIO_PIN_RESET); // userDis2
            HAL_GPIO_WritePin(GPIOB, GPIO_PIN_0, GPIO_PIN_RESET); // userDis3
            HAL_GPIO_WritePin(GPIOA, GPIO_PIN_4, GPIO_PIN_RESET); // userDis4

            switch (nextDigit_FSM) {
                case 1:
                    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_0, GPIO_PIN_SET);
                    displayNumber(display_value % 10);
                    break;
                case 2:
                    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_1, GPIO_PIN_SET);
                    displayNumber((display_value / 10) % 10);
                    break;
                case 3:
                    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_0, GPIO_PIN_SET);
                    displayNumber((display_value / 100) % 10);
                    break;
                case 4:
                    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_4, GPIO_PIN_SET);
                    displayNumber((display_value / 1000) % 10);
                    break;
            }
        }
    }
}

```

```

    }
    nextDigit_FSM = (nextDigit_FSM < 4) ? nextDigit_FSM + 1 : 1;
    break;}
case EVENT_USART: {
    // eco
    int len = snprintf((char*)tx_buffer, sizeof(tx_buffer),
        "Comando recibido: %s\r\n", rx_buffer);
    HAL_UART_Transmit(&huart2, tx_buffer, len, 1000);

    // dispatch
    if (strncmp((char*)rx_buffer,"led=",4)==0) HandleLEDDelayCmd(rx_buffer+4);
    else if (strncmp((char*)rx_buffer,"fmuestreo=",10)==0)
HandleSampleFreqCmd(rx_buffer+10);
    else if (strncmp((char*)rx_buffer,"rgb=",4)==0) HandleRGBCmd(rx_buffer+4);
    else if (strncmp((char*)rx_buffer,"status",6)==0) HandleStatusCmd();
    else if (strncmp((char*)rx_buffer,"help",4)==0) HandleHelpCmd();
    else if (strncmp((char*)rx_buffer, "clear", 5) == 0) HandleClearCmd();
    else
        ;

    memset(rx_buffer,0,sizeof(rx_buffer));
    break;
}return event;
}
}

```

```

void displayNumber(uint8_t digitValue) {
    uint32_t start = DWT->CYCCNT;
    static uint8_t contador_muestras = 0;
    // Apagar todos los segmentos
    HAL_GPIO_WritePin(GPIOC, GPIO_PIN_4, GPIO_PIN_RESET); // A
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_10, GPIO_PIN_RESET); // B
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_13, GPIO_PIN_RESET); // C
    HAL_GPIO_WritePin(GPIOC, GPIO_PIN_0, GPIO_PIN_RESET); // D
    HAL_GPIO_WritePin(GPIOC, GPIO_PIN_1, GPIO_PIN_RESET); // E
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_1, GPIO_PIN_RESET); // F
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_5, GPIO_PIN_RESET); // G

    switch (digitValue) {
        case 0:
            HAL_GPIO_WritePin(GPIOC, GPIO_PIN_4, GPIO_PIN_SET);
            HAL_GPIO_WritePin(GPIOB, GPIO_PIN_10, GPIO_PIN_SET);
            HAL_GPIO_WritePin(GPIOB, GPIO_PIN_13, GPIO_PIN_SET);
            HAL_GPIO_WritePin(GPIOC, GPIO_PIN_0, GPIO_PIN_SET);
            HAL_GPIO_WritePin(GPIOC, GPIO_PIN_1, GPIO_PIN_SET);
            HAL_GPIO_WritePin(GPIOB, GPIO_PIN_1, GPIO_PIN_SET);
            break;
        case 1:
            HAL_GPIO_WritePin(GPIOB, GPIO_PIN_10, GPIO_PIN_SET);
            HAL_GPIO_WritePin(GPIOB, GPIO_PIN_13, GPIO_PIN_SET);
            break;
        case 2:

```

```

        HAL_GPIO_WritePin(GPIOC, GPIO_PIN_4, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_10, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOC, GPIO_PIN_0, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOC, GPIO_PIN_1, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_5, GPIO_PIN_SET);
        break;
    case 3:
        HAL_GPIO_WritePin(GPIOC, GPIO_PIN_4, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_10, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_13, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOC, GPIO_PIN_0, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_5, GPIO_PIN_SET);
        break;
    case 4:
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_10, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_13, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_1, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_5, GPIO_PIN_SET);
        break;
    case 5:
        HAL_GPIO_WritePin(GPIOC, GPIO_PIN_4, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_13, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOC, GPIO_PIN_0, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_1, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_5, GPIO_PIN_SET);
        break;
    case 6:
        HAL_GPIO_WritePin(GPIOC, GPIO_PIN_4, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_13, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOC, GPIO_PIN_0, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOC, GPIO_PIN_1, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_1, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_5, GPIO_PIN_SET);
        break;
    case 7:
        HAL_GPIO_WritePin(GPIOC, GPIO_PIN_4, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_10, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_13, GPIO_PIN_SET);
        break;
    case 8:
        HAL_GPIO_WritePin(GPIOC, GPIO_PIN_4, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_10, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_13, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOC, GPIO_PIN_0, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOC, GPIO_PIN_1, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_1, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_5, GPIO_PIN_SET);
        break;
    case 9:
        HAL_GPIO_WritePin(GPIOC, GPIO_PIN_4, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_10, GPIO_PIN_SET);

```

```

        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_13, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_1, GPIO_PIN_SET);
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_5, GPIO_PIN_SET);
        break;
    }
    uint32_t elapsed = DWT->CYCCNT - start;

    if (contador_muestras < 10) {
        printf("Ciclos para mostrar digito %d: %lu\n", digitValue, (unsigned long)elapsed);
        contador_muestras++;
    }
}

void HandleLEDDelayCmd(const char *arg) {
    uint32_t nuevo = atoi(arg);
    // Ajusta TIM2 para el blinky
    __HAL_TIM_SET_AUTORELOAD(&htim2, nuevo);
    __HAL_TIM_SET_COUNTER(&htim2, 0);
    // Feedback
    int len = snprintf((char*)tx_buffer, sizeof(tx_buffer),
        "LED delay = %lu ms\r\n", (unsigned long)nuevo);
    HAL_UART_Transmit(&huart2, tx_buffer, len, 1000);
}

void HandleSampleFreqCmd(const char *arg) {
    uint32_t fs;
    char option = arg[0];
    // Verifica la opción seleccionada (1, 2, 3, 4)
    switch(option) {
        case '1': fs = 44100; break;
        case '2': fs = 48000; break;
        case '3': fs = 96000; break;
        case '4': fs = 128000; break;
        default: {
            const char *msg = "Opciones válidas para 'fmuestreo=' son: 1, 2, 3, 4\r\n";
            HAL_UART_Transmit(&huart2, (uint8_t*)msg, strlen(msg), 1000);
            return;
        }
    }

    // calcular ARR y PSC
    uint32_t arr = timer_clk / fs - 1;
    uint32_t psc = 0;
    if (arr > 0xFFFF) {
        // escalar PSC si supera 16 bits
        psc = (arr / 0x10000) + 1;
        arr = (timer_clk / (psc+1) / fs) - 1;
    }
    // aplicar a TIM3 y reiniciar
    __HAL_TIM_SET_PRESCALER(&htim3, psc);
    __HAL_TIM_SET_AUTORELOAD(&htim3, arr);
    __HAL_TIM_SET_COUNTER(&htim3, 0);
    HAL_TIM_Base_Stop(&htim3);
}

```

```

HAL_TIM_Base_Start(&htim3);
// Feedback
int len = snprintf((char*)tx_buffer, sizeof(tx_buffer),
                  "Sample TIM3 @ %lu Hz (PSC=%lu, ARR=%lu)\r\n",
                  (unsigned long)fs, (unsigned long)psc, (unsigned long)arr);
HAL_UART_Transmit(&huart2, tx_buffer, len, 1000);
}

void HandleRGBCmd(const char *arg) {
    uint32_t r, g, b;
    if (sscanf(arg, "%lu %lu %lu", &r, &g, &b) == 3) {
        if (r > 255) r = 255;
        if (g > 255) g = 255;
        if (b > 255) b = 255;

        __HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_1, g); // G
        __HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_2, b); // B
        __HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_3, r); // R

        int len = snprintf((char*)tx_buffer, sizeof(tx_buffer),
                          "RGB PWM -> R=%lu, G=%lu, B=%lu\r\n", r, g, b);
        HAL_UART_Transmit(&huart2, tx_buffer, len, 1000);
    } else {
        const char *msg = "Formato inválido. Usa: rgb=R G B@\r\n";
        HAL_UART_Transmit(&huart2, (uint8_t*)msg, strlen(msg), 1000);
    }
}

void HandleStatusCmd(void) {
    uint32_t psc = htim3.Init.Prescaler;
    uint32_t arr = htim3.Init.Period;
    float fs = 84000000.0f / ((psc + 1) * (arr + 1));

    int len = snprintf((char*)tx_buffer, sizeof(tx_buffer),
                      "Config:\r\nSample TIM3 @ %.2f Hz (PSC=%lu, ARR=%lu)\r\nCanal ADC: 6 y 7\r\n",
                      fs, (unsigned long)psc, (unsigned long)arr);
    HAL_UART_Transmit(&huart2, tx_buffer, len, 1000);
}

void HandleClearCmd(void) {
    const char *clear_screen = "\033[2J\033[H"; // Secuencia ANSI para limpiar la pantalla
    HAL_UART_Transmit(&huart2, (uint8_t*)clear_screen, strlen(clear_screen), 1000);
}

void HandleHelpCmd(void) {
    const char *help_msg =
        "\r\n===== AYUDA ====="

```



```

        "\r\nComandos disponibles:\r\n"
        " led=<ms>@      - Cambia la frecuencia del LED Blinky\r\n"
        " fmuestreo=<1|2|3|4>@ - Frecuencia de muestreo del ADC:\r\n"
        "      - 1 -> 44100 Hz\r\n"
        "      - 2 -> 48000 Hz\r\n"
        "      - 3 -> 96000 Hz\r\n"
        "      - 4 -> 128000 Hz\r\n"
        " rgb=<RGB>@      - Control de LED RGB, ej: rgb=RG\r\n"
        " status@      - Mostrar configuración actual\r\n"
        " print@      - Imprimir datos crudos del ADC\r\n"
        " help @      - Mostrar esta ayuda\r\n"
        "===== \r\n";
    HAL_UART_Transmit(&huart2, (uint8_t*)help_msg, strlen(help_msg), 1000);
}

void imprimirBinario(uint8_t valor) {
    for (int i = 7; i >= 0; i--) {
        printf("%d", (valor >> i) & 1);
    }
}

void mostrarOperacion(const char *desc, uint8_t resultado) {
    printf("%s\n", desc);
    printf("Binario: "); imprimirBinario(resultado);
    printf("\nHex: 0x%02X\n", resultado);
    printf("ASCII: %c\n\n", (resultado >= 32 && resultado <= 126) ? resultado : '.');
}

void verificarOperacionesCrucigrama(void) {
    // 1. Caracter ASCII 'B'
    uint8_t op1 = 'B';
    mostrarOperacion("1. ASCII 'B'", op1);

    // 2. ((!!0xC) << 0b11) | (~0x45)
    uint8_t op2 = ((!!0xC) << 0b11) | (~0x45);
    mostrarOperacion("2. ((!!0xC) << 0b11) | (~0x45)", op2);

    // 3. ((0b00111100 & 0xAC) << 0x3)
    uint8_t op3 = (0b00111100 & 0xAC) << 0x3;
    mostrarOperacion("3. ((0b00111100 & 0xAC) << 0x3)", op3);

    // 4. -121
    uint8_t op4 = (uint8_t)(-121);
    mostrarOperacion("4. -121 (uint8_t)", op4);

    // 5. (0xCAFE & 0xFF0) >> 7
    uint16_t val5 = (0xCAFE & 0xFF0) >> 7;
    uint8_t op5 = (uint8_t)(val5 & 0xFF);
    mostrarOperacion("5. (0xCAFE & 0xFF0) >> 7", op5);

    // 6. (1 << 4) + (0x3 << 4) - 10 + (0b11 << 0x6)
    uint8_t op6 = (1 << 4) + (0x3 << 4) - 10 + (0b11 << 6);

```

```

mostrarOperacion("6. (1<<4)+(0x3<<4)-10+(0b11<<6)", op6);

// 7. ('T' | '5')
uint8_t op7 = 'T' | '5';
mostrarOperacion("7. ('T' | '5')", op7);

// 8. (2921 & 0xD6)
uint8_t op8 = 2921 & 0xD6;
mostrarOperacion("8. (2921 & 0xD6)", op8);

// 9. 01100110 ^ 00101011
uint8_t a9 = 0b01100110;
uint8_t b9 = 0b00101011;
uint8_t op9 = a9 ^ b9;
mostrarOperacion("9. 01100110 ^ 00101011", op9);
}
/* USER CODE END 0 */

/**
 * @brief The application entry point.
 * @retval int
 */
int main(void)
{

/* USER CODE BEGIN 1 */

/* USER CODE END 1 */

/* MCU Configuration-----*/

/* Reset of all peripherals, Initializes the Flash interface and the Systick. */
HAL_Init();

/* USER CODE BEGIN Init */
CoreDebug->DEMCR |= CoreDebug_DEMCR_TRCENA_Msk; // Habilitar DWT
DWT->CYCCNT = 0; // Reiniciar contador
DWT->CTRL |= DWT_CTRL_CYCCNTENA_Msk; // Encender contador de ciclos
/* USER CODE END Init */

/* Configure the system clock */
SystemClock_Config();

/* USER CODE BEGIN SysInit */

/* USER CODE END SysInit */

/* Initialize all configured peripherals */
MX_GPIO_Init();
MX_DMA_Init();
MX_USART2_UART_Init();

```

```

MX_TIM2_Init();
MX_TIM4_Init();
MX_ADC1_Init();
MX_TIM3_Init();
MX_TIM1_Init();
/* USER CODE BEGIN 2 */
HAL_TIM_Base_Start_IT(&htim2);
HAL_TIM_Base_Start_IT(&htim3);
HAL_TIM_Base_Start_IT(&htim4);
HAL_UART_Receive_IT(&huart2,&rx_char,1);
HAL_ADC_Start_DMA(&hadc1, (uint32_t*)adc_buffer, buffer_size);
HAL_TIM_IC_Start_IT(&htim3, TIM_CHANNEL_2);
HandleSampleFreqCmd(3); //96000 hz muestreo

verificarOperacionesCrucigrama();
medirTiempoEnvioUART();
HandleHelpCmd();

/* USER CODE END 2 */

/* Infinite loop */
/* USER CODE BEGIN WHILE */
while (1)
{
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
    if (pending_event != IDLE){
        state_machine_action(pending_event);
        pending_event = IDLE;
    }
}
/* USER CODE END 3 */
}

/**
 * @brief System Clock Configuration
 * @retval None
 */
void SystemClock_Config(void)
{
    RCC_OscInitTypeDef RCC_OscInitStruct = {0};
    RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};

    /** Configure the main internal regulator output voltage
     */
    __HAL_RCC_PWR_CLK_ENABLE();
    __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE1);

    /** Initializes the RCC Oscillators according to the specified parameters
     * in the RCC_OscInitTypeDef structure.

```

```

*/
RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI;
RCC_OscInitStruct.HSIState = RCC_HSI_ON;
RCC_OscInitStruct.HSICalibrationValue = RCC_HSICALIBRATION_DEFAULT;
RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSI;
RCC_OscInitStruct.PLL.PLLM = 16;
RCC_OscInitStruct.PLL.PLLN = 336;
RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV4;
RCC_OscInitStruct.PLL.PLLQ = 4;
if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
{
    Error_Handler();
}

/** Initializes the CPU, AHB and APB buses clocks
*/
RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
                              |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV2;
RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;

if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_2) != HAL_OK)
{
    Error_Handler();
}
}

/**
 * @brief ADC1 Initialization Function
 * @param None
 * @retval None
 */
static void MX_ADC1_Init(void)
{
    /* USER CODE BEGIN ADC1_Init 0 */

    /* USER CODE END ADC1_Init 0 */

    ADC_ChannelConfTypeDef sConfig = {0};

    /* USER CODE BEGIN ADC1_Init 1 */

    /* USER CODE END ADC1_Init 1 */

    /** Configure the global features of the ADC (Clock, Resolution, Data Alignment and number of
    conversion)
    */

```

```

hadc1.Instance = ADC1;
hadc1.Init.ClockPrescaler = ADC_CLOCK_SYNC_PCLK_DIV4;
hadc1.Init.Resolution = ADC_RESOLUTION_12B;
hadc1.Init.ScanConvMode = ENABLE;
hadc1.Init.ContinuousConvMode = DISABLE;
hadc1.Init.DiscontinuousConvMode = DISABLE;
hadc1.Init.ExternalTrigConvEdge = ADC_EXTERNALTRIGCONVEDGE_RISING;
hadc1.Init.ExternalTrigConv = ADC_EXTERNALTRIGCONV_T3_TRGO;
hadc1.Init.DataAlign = ADC_DATAALIGN_RIGHT;
hadc1.Init.NbrOfConversion = 2;
hadc1.Init.DMAContinuousRequests = ENABLE;
hadc1.Init.EOCSelection = ADC_EOC_SINGLE_CONV;
if (HAL_ADC_Init(&hadc1) != HAL_OK)
{
    Error_Handler();
}

/** Configure for the selected ADC regular channel its corresponding rank in the sequencer and its
sample time.
*/
sConfig.Channel = ADC_CHANNEL_6;
sConfig.Rank = 1;
sConfig.SamplingTime = ADC_SAMPLETIME_3CYCLES;
if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
{
    Error_Handler();
}

/** Configure for the selected ADC regular channel its corresponding rank in the sequencer and its
sample time.
*/
sConfig.Channel = ADC_CHANNEL_7;
sConfig.Rank = 2;
if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
{
    Error_Handler();
}
/* USER CODE BEGIN ADC1_Init 2 */

/* USER CODE END ADC1_Init 2 */

}

/**
 * @brief TIM1 Initialization Function
 * @param None
 * @retval None
 */
static void MX_TIM1_Init(void)
{

```

```

/* USER CODE BEGIN TIM1_Init 0 */

/* USER CODE END TIM1_Init 0 */

TIM_MasterConfigTypeDef sMasterConfig = {0};
TIM_OC_InitTypeDef sConfigOC = {0};
TIM_BreakDeadTimeConfigTypeDef sBreakDeadTimeConfig = {0};

/* USER CODE BEGIN TIM1_Init 1 */

/* USER CODE END TIM1_Init 1 */
htim1.Instance = TIM1;
htim1.Init.Prescaler = 83;
htim1.Init.CounterMode = TIM_COUNTERMODE_UP;
htim1.Init.Period = 255;
htim1.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
htim1.Init.RepetitionCounter = 0;
htim1.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
if (HAL_TIM_PWM_Init(&htim1) != HAL_OK)
{
    Error_Handler();
}
sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
if (HAL_TIMEx_MasterConfigSynchronization(&htim1, &sMasterConfig) != HAL_OK)
{
    Error_Handler();
}
sConfigOC.OCMode = TIM_OCMODE_PWM1;
sConfigOC.Pulse = 0;
sConfigOC.OCpolarity = TIM_OCPOLARITY_HIGH;
sConfigOC.OCNPolarity = TIM_OCNPOLARITY_HIGH;
sConfigOC.OCFastMode = TIM_OCFAST_DISABLE;
sConfigOC.OCIdleState = TIM_OCIDLESTATE_RESET;
sConfigOC.OCNIdleState = TIM_OCNIDLESTATE_RESET;
if (HAL_TIM_PWM_ConfigChannel(&htim1, &sConfigOC, TIM_CHANNEL_1) != HAL_OK)
{
    Error_Handler();
}
if (HAL_TIM_PWM_ConfigChannel(&htim1, &sConfigOC, TIM_CHANNEL_2) != HAL_OK)
{
    Error_Handler();
}
if (HAL_TIM_PWM_ConfigChannel(&htim1, &sConfigOC, TIM_CHANNEL_3) != HAL_OK)
{
    Error_Handler();
}
sBreakDeadTimeConfig.OffStateRunMode = TIM_OSSR_DISABLE;
sBreakDeadTimeConfig.OffStateIDLEMode = TIM_OSSI_DISABLE;
sBreakDeadTimeConfig.LockLevel = TIM_LOCKLEVEL_OFF;
sBreakDeadTimeConfig.DeadTime = 0;

```

```

sBreakDeadTimeConfig.BreakState = TIM_BREAK_DISABLE;
sBreakDeadTimeConfig.BreakPolarity = TIM_BREAKPOLARITY_HIGH;
sBreakDeadTimeConfigAutomaticOutput = TIM_AUTOMATICOUTPUT_DISABLE;
if (HAL_TIMEx_ConfigBreakDeadTime(&htim1, &sBreakDeadTimeConfig) != HAL_OK)
{
    Error_Handler();
}
/* USER CODE BEGIN TIM1_Init 2 */
HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_1); // PA8 - g
HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_2); // PA9 - B
HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_3); // PA10 - R
/* USER CODE END TIM1_Init 2 */
HAL_TIM_MspPostInit(&htim1);

}

/**
 * @brief TIM2 Initialization Function
 * @param None
 * @retval None
 */
static void MX_TIM2_Init(void)
{

    /* USER CODE BEGIN TIM2_Init 0 */

    /* USER CODE END TIM2_Init 0 */

    TIM_ClockConfigTypeDef sClockSourceConfig = {0};
    TIM_MasterConfigTypeDef sMasterConfig = {0};

    /* USER CODE BEGIN TIM2_Init 1 */

    /* USER CODE END TIM2_Init 1 */
    htim2.Instance = TIM2;
    htim2.Init.Prescaler = 42000-1;
    htim2.Init.CounterMode = TIM_COUNTERMODE_UP;
    htim2.Init.Period = 500;
    htim2.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
    htim2.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_ENABLE;
    if (HAL_TIM_Base_Init(&htim2) != HAL_OK)
    {
        Error_Handler();
    }
    sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
    if (HAL_TIM_ConfigClockSource(&htim2, &sClockSourceConfig) != HAL_OK)
    {
        Error_Handler();
    }
    sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
    sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;

```

```

if (HAL_TIMEx_MasterConfigSynchronization(&htim2, &sMasterConfig) != HAL_OK)
{
    Error_Handler();
}
/* USER CODE BEGIN TIM2_Init 2 */

/* USER CODE END TIM2_Init 2 */

}

/**
 * @brief TIM3 Initialization Function
 * @param None
 * @retval None
 */
static void MX_TIM3_Init(void)
{
    /* USER CODE BEGIN TIM3_Init 0 */

    /* USER CODE END TIM3_Init 0 */

    TIM_ClockConfigTypeDef sClockSourceConfig = {0};
    TIM_MasterConfigTypeDef sMasterConfig = {0};
    TIM_IC_InitTypeDef sConfigIC = {0};

    /* USER CODE BEGIN TIM3_Init 1 */

    /* USER CODE END TIM3_Init 1 */
    htim3.Instance = TIM3;
    htim3.Init.Prescaler = 0;
    htim3.Init.CounterMode = TIM_COUNTERMODE_UP;
    htim3.Init.Period = 65535;
    htim3.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
    htim3.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
    if (HAL_TIM_Base_Init(&htim3) != HAL_OK)
    {
        Error_Handler();
    }
    sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
    if (HAL_TIM_ConfigClockSource(&htim3, &sClockSourceConfig) != HAL_OK)
    {
        Error_Handler();
    }
    if (HAL_TIM_IC_Init(&htim3) != HAL_OK)
    {
        Error_Handler();
    }
    sMasterConfig.MasterOutputTrigger = TIM_TRGO_UPDATE;
    sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
    if (HAL_TIMEx_MasterConfigSynchronization(&htim3, &sMasterConfig) != HAL_OK)

```



```

{
    Error_Handler();
}
sConfigIC.ICPolarity = TIM_INPUTCHANNELPOLARITY_RISING;
sConfigIC.ICSelection = TIM_ICSELECTION_DIRECTTI;
sConfigIC.ICPrescaler = TIM_ICPSC_DIV1;
sConfigIC.ICFilter = 8;
if (HAL_TIM_IC_ConfigChannel(&htim3, &sConfigIC, TIM_CHANNEL_2) != HAL_OK)
{
    Error_Handler();
}
/* USER CODE BEGIN TIM3_Init 2 */
HAL_TIM_IC_Start_IT(&htim3, TIM_CHANNEL_2);
/* USER CODE END TIM3_Init 2 */

}

/**
 * @brief TIM4 Initialization Function
 * @param None
 * @retval None
 */
static void MX_TIM4_Init(void)
{

    /* USER CODE BEGIN TIM4_Init 0 */

    /* USER CODE END TIM4_Init 0 */

    TIM_ClockConfigTypeDef sClockSourceConfig = {0};
    TIM_MasterConfigTypeDef sMasterConfig = {0};

    /* USER CODE BEGIN TIM4_Init 1 */

    /* USER CODE END TIM4_Init 1 */
    htim4.Instance = TIM4;
    htim4.Init.Prescaler = 840-1;
    htim4.Init.CounterMode = TIM_COUNTERMODE_UP;
    htim4.Init.Period = 200-1;
    htim4.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
    htim4.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
    if (HAL_TIM_Base_Init(&htim4) != HAL_OK)
    {
        Error_Handler();
    }
    sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
    if (HAL_TIM_ConfigClockSource(&htim4, &sClockSourceConfig) != HAL_OK)
    {
        Error_Handler();
    }
    sMasterConfig.MasterOutputTrigger = TIM_TRGO_UPDATE;

```

```
sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
if (HAL_TIMEx_MasterConfigSynchronization(&htim4, &sMasterConfig) != HAL_OK)
{
    Error_Handler();
}
/* USER CODE BEGIN TIM4_Init 2 */

/* USER CODE END TIM4_Init 2 */

}

/**
 * @brief USART2 Initialization Function
 * @param None
 * @retval None
 */
static void MX_USART2_UART_Init(void)
{

/* USER CODE BEGIN USART2_Init 0 */

/* USER CODE END USART2_Init 0 */

/* USER CODE BEGIN USART2_Init 1 */

/* USER CODE END USART2_Init 1 */
huart2.Instance = USART2;
huart2.Init.BaudRate = 115200;
huart2.Init.WordLength = UART_WORDLENGTH_8B;
huart2.Init.StopBits = UART_STOPBITS_1;
huart2.Init.Parity = UART_PARITY_NONE;
huart2.Init.Mode = UART_MODE_TX_RX;
huart2.Init.HwFlowCtl = UART_HWCONTROL_NONE;
huart2.Init.OverSampling = UART_OVERSAMPLING_16;
if (HAL_UART_Init(&huart2) != HAL_OK)
{
    Error_Handler();
}
/* USER CODE BEGIN USART2_Init 2 */

/* USER CODE END USART2_Init 2 */

}

/**
 * Enable DMA controller clock
 */
static void MX_DMA_Init(void)
{

/* DMA controller clock enable */
```

```

__HAL_RCC_DMA2_CLK_ENABLE();
__HAL_RCC_DMA1_CLK_ENABLE();

/* DMA interrupt init */
/* DMA1_Stream5_IRQn interrupt configuration */
HAL_NVIC_SetPriority(DMA1_Stream5_IRQn, 0, 0);
HAL_NVIC_EnableIRQ(DMA1_Stream5_IRQn);
/* DMA2_Stream0_IRQn interrupt configuration */
HAL_NVIC_SetPriority(DMA2_Stream0_IRQn, 0, 0);
HAL_NVIC_EnableIRQ(DMA2_Stream0_IRQn);

}

/**
 * @brief GPIO Initialization Function
 * @param None
 * @retval None
 */
static void MX_GPIO_Init(void)
{
    GPIO_InitTypeDef GPIO_InitStruct = {0};
    /* USER CODE BEGIN MX_GPIO_Init_1 */
    /* USER CODE END MX_GPIO_Init_1 */

    /* GPIO Ports Clock Enable */
    __HAL_RCC_GPIOC_CLK_ENABLE();
    __HAL_RCC_GPIOH_CLK_ENABLE();
    __HAL_RCC_GPIOA_CLK_ENABLE();
    __HAL_RCC_GPIOB_CLK_ENABLE();

    /*Configure GPIO pin Output Level */
    HAL_GPIO_WritePin(GPIOC, LedB_Pin|segD_Pin|segE_Pin|LedR_Pin
                      |LedG_Pin|segA_Pin, GPIO_PIN_RESET);

    /*Configure GPIO pin Output Level */
    HAL_GPIO_WritePin(userLed_GPIO_Port, userLed_Pin, GPIO_PIN_SET);

    /*Configure GPIO pin Output Level */
    HAL_GPIO_WritePin(GPIOA, dis1_Pin|dis2_Pin|dis4_Pin, GPIO_PIN_RESET);

    /*Configure GPIO pin Output Level */
    HAL_GPIO_WritePin(GPIOB, dis3_Pin|segF_Pin|segB_Pin|segC_Pin
                      |segG_Pin, GPIO_PIN_RESET);

    /*Configure GPIO pins : LedB_Pin segD_Pin segE_Pin LedR_Pin
                      LedG_Pin segA_Pin */
    GPIO_InitStruct.Pin = LedB_Pin|segD_Pin|segE_Pin|LedR_Pin
                      |LedG_Pin|segA_Pin;
    GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
    GPIO_InitStruct.Pull = GPIO_NOPULL;
    GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;

```

```

HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);

/*Configure GPIO pin : userLed_Pin */
GPIO_InitStruct.Pin = userLed_Pin;
GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
GPIO_InitStruct.Pull = GPIO_NOPULL;
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
HAL_GPIO_Init(userLed_GPIO_Port, &GPIO_InitStruct);

/*Configure GPIO pins : dis1_Pin dis2_Pin dis4_Pin */
GPIO_InitStruct.Pin = dis1_Pin | dis2_Pin | dis4_Pin;
GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
GPIO_InitStruct.Pull = GPIO_NOPULL;
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

/*Configure GPIO pins : dis3_Pin segF_Pin segB_Pin segC_Pin
                        segG_Pin */
GPIO_InitStruct.Pin = dis3_Pin | segF_Pin | segB_Pin | segC_Pin
                        | segG_Pin;
GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
GPIO_InitStruct.Pull = GPIO_NOPULL;
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);

/*Configure GPIO pin : userData_Pin */
GPIO_InitStruct.Pin = userData_Pin;
GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
GPIO_InitStruct.Pull = GPIO_NOPULL;
HAL_GPIO_Init(userData_GPIO_Port, &GPIO_InitStruct);

/*Configure GPIO pin : userSw_Pin */
GPIO_InitStruct.Pin = userSw_Pin;
GPIO_InitStruct.Mode = GPIO_MODE_IT_RISING;
GPIO_InitStruct.Pull = GPIO_NOPULL;
HAL_GPIO_Init(userSw_GPIO_Port, &GPIO_InitStruct);

/*Configure GPIO pin : userClk_Pin */
GPIO_InitStruct.Pin = userClk_Pin;
GPIO_InitStruct.Mode = GPIO_MODE_IT_RISING;
GPIO_InitStruct.Pull = GPIO_NOPULL;
HAL_GPIO_Init(userClk_GPIO_Port, &GPIO_InitStruct);

/* EXTI interrupt init*/
HAL_NVIC_SetPriority(EXTI9_5_IRQn, 0, 0);
HAL_NVIC_EnableIRQ(EXTI9_5_IRQn);

HAL_NVIC_SetPriority(EXTI15_10_IRQn, 0, 0);
HAL_NVIC_EnableIRQ(EXTI15_10_IRQn);

/* USER CODE BEGIN MX_GPIO_Init_2 */

```

```

/* USER CODE END MX_GPIO_Init_2 */
}

/* USER CODE BEGIN 4 */
void medirTiempoEnvioUART(void) {
    const char *msg = "Hola mundo! - Examen Taller V\n";
    uint32_t start = DWT->CYCCNT;

    printf("%s", msg);

    uint32_t elapsed = DWT->CYCCNT - start;
    printf("Ciclos para enviar mensaje UART: %lu\n", (unsigned long)elapsed);
}
int __io_putchar(int ch) {
    HAL_UART_Transmit(&huart2, (uint8_t*)&ch, 1, HAL_MAX_DELAY);
    return ch;
}
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim){
    if(htim->Instance==TIM2){
        HAL_GPIO_TogglePin(userLed_GPIO_Port,userLed_Pin);
    }else if(htim->Instance==TIM4){
        if (pending_event == IDLE)
            pending_event = EVENT_TIMER_TICK;
    }else if (htim->Instance==TIM3){

    }
}
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
{
    if (GPIO_Pin == userClk_Pin) {
        data_snapshot = HAL_GPIO_ReadPin(userData_GPIO_Port, userData_Pin);
        if (pending_event == IDLE)
            pending_event = EVENT_ENCODER;
    } else if (GPIO_Pin == userSw_Pin) {
        if (pending_event == IDLE)
            pending_event = EVENT_SW;
    }
}
void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart){
    if(huart->Instance == USART2){
        if(rx_index<sizeof(rx_buffer)-1){
            rx_buffer[rx_index++]=rx_char;
        }
        if (rx_char == '@') {
            rx_buffer[rx_index]='\0';
            rx_index=0;
            pending_event = EVENT_USART; //aquí levanta tu bandera
        }
        HAL_UART_Receive_IT(huart, &rx_char,1);
    }
}

```

```

}

void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef* hadc)
{
    if (hadc->Instance == ADC1) {
        for (int i = 0; i < NUM_MUESTRAS; i++) {
            vry_buffer[i] = adc_buffer[2 * i];    // Canal 6
            vrx_buffer[i] = adc_buffer[2 * i + 1]; // Canal 7
        }
    }
}

/* USER CODE END 4 */

/**
 * @brief This function is executed in case of error occurrence.
 * @retval None
 */
void Error_Handler(void)
{
    /* USER CODE BEGIN Error_Handler_Debug */
    /* User can add his own implementation to report the HAL error return state */
    __disable_irq();
    while (1)
    {
    }
    /* USER CODE END Error_Handler_Debug */
}

#ifdef USE_FULL_ASSERT
/**
 * @brief Reports the name of the source file and the source line number
 *        where the assert_param error has occurred.
 * @param file: pointer to the source file name
 * @param line: assert_param error line source number
 * @retval None
 */
void assert_failed(uint8_t *file, uint32_t line)
{
    /* USER CODE BEGIN 6 */
    /* User can add his own implementation to report the file name and line number,
       ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */
    /* USER CODE END 6 */
}
#endif /* USE_FULL_ASSERT */

```

Gracias por compartir el código completo. El problema por el cual **no se están mostrando los valores del joystick en el display de 7 segmentos** probablemente se debe a un **mal orden en la separación de los canales del ADC** dentro del `HAL_ADC_ConvCpltCallback`. En tu configuración actual, tienes:

```
c
vry_buffer[i] = adc_buffer[2 * i];      // Canal 6
vrx_buffer[i] = adc_buffer[2 * i + 1];  // Canal 7
```

Pero según la configuración del ADC en `MX_ADC1_Init()`, el **canal 6** está primero (Rank 1) y el **canal 7** segundo (Rank 2). Esto significa:

- `adc_buffer[2*i]` → **VRx** (canal 6)
- `adc_buffer[2*i+1]` → **VRy** (canal 7)

Entonces debes corregir el callback así:

```
c
vrx_buffer[i] = adc_buffer[2 * i];      // Canal 6 - VRx
vry_buffer[i] = adc_buffer[2 * i + 1];  // Canal 7 - VRy
```

Y en tu `EVENT_TIMER_TICK`, también debes corregir el orden si estás usando:

```
c
uint16_t x_val = (vrx_buffer[NUM_MUESTRAS - 1] * 99) / 4095;
uint16_t y_val = (vry_buffer[NUM_MUESTRAS - 1] * 99) / 4095;
```

Asegúrate de que VRx y VRy correspondan a los ejes del joystick correctamente, tanto físicamente como en software.

Verificación adicional

También asegúrate de que el evento `EVENT_TIMER_TICK` esté ocurriendo regularmente. Esto depende del `TIM4`, que debe estar activo y generar interrupciones, y que `pending_event` esté en `IDLE` para que se procese.

Puedes insertar una línea de depuración dentro del `EVENT_TIMER_TICK`:

```
c
printf("VRx: %d, VRy: %d\n", vrx_buffer[NUM_MUESTRAS - 1], vry_buffer[NUM_MUESTRAS - 1]);
```

Para confirmar que los valores están siendo leídos correctamente.

Resumen: cambios sugeridos

1. En `HAL_ADC_ConvCpltCallback`:

```
c
for (int i = 0; i < NUM_MUESTRAS; i++) {
    vrx_buffer[i] = adc_buffer[2 * i];      // Canal 6 (VRx)
    vry_buffer[i] = adc_buffer[2 * i + 1];  // Canal 7 (VRy)
}
```

2. En EVENT_TIMER_TICK:

Verifica que `x_val` y `y_val` estén en el orden correcto según tus necesidades visuales.

¿Deseas que también te proponga pines alternativos para los canales PWM del LED RGB, según los disponibles en tu placa STM32?



`vr_x` va a canal 6 y `vry` a canal 7

Gracias por la aclaración. Ya que `vr_x_buffer` está asociado al **canal 7** y `vry_buffer` al **canal 6**, el problema está en cómo se está leyendo el buffer ADC en la función `HAL_ADC_ConvCpltCallback`.

Actualmente tienes:

```
c
vr_y_buffer[i] = adc_buffer[2 * i];      // Canal 6
vr_x_buffer[i] = adc_buffer[2 * i + 1];  // Canal 7
```

Pero si **`vr_x` es canal 6** y **`vry` es canal 7**, deberías invertir esas asignaciones:

```
c
vr_x_buffer[i] = adc_buffer[2 * i];      // Canal 6 (VRx)
vr_y_buffer[i] = adc_buffer[2 * i + 1];  // Canal 7 (VRy)
```

Esto es muy importante porque si lo dejas mal, estarás mostrando el valor incorrecto en el display (como si X e Y estuvieran cruzados), o simplemente verás resultados que no se actualizan como esperas.

Solución

Corrige la función así:

```
c
void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef* hadc)
{
    if (hadc->Instance == ADC1) {
        for (int i = 0; i < NUM_MUESTRAS; i++) {
            vr_x_buffer[i] = adc_buffer[2 * i];      // Canal 6 - VRx
            vr_y_buffer[i] = adc_buffer[2 * i + 1];  // Canal 7 - VRy
        }
    }
}
```

Después de hacer ese cambio, tu línea para mostrar en el display:

```
c
display_value = (y_val * 100) + x_val;
```

debería empezar a mostrar correctamente el valor combinado del joystick en los 4 dígitos del display.

¿Deseas invertir ese orden en el display también (mostrar primero VRx y luego VRy)? Si es así, podemos modificar la expresión a:

```
c
display_value = (x_val * 100) + y_val;
```

Esto mostraría VRx primero y VRy después (por ejemplo, 5237 sería VRx=52, VRy=37).



dame un resumen de las consultas y respuestas

Claro, aquí tienes un **resumen organizado** de tus consultas y las respuestas proporcionadas:

1. Verificación de operaciones del crucigrama binario

- **Consulta:** Cómo agregar una función que imprima operaciones lógicas del crucigrama binario por USART2 (en binario, hexadecimal y ASCII), y en qué parte del código pegarla.
- **Respuesta:** Se entregó una función `verificarOperacionesCrucigrama()` y `mostrarOperacion()` que imprime los resultados, además de instrucciones para insertarla dentro del `main()` y comentarla fácilmente para pruebas.

2. Limpieza del código para cumplir el enunciado

- **Consulta:** Qué partes del código deben eliminarse para que cumpla con el enunciado del examen (eliminar FFT, frecuencia, etc.) manteniendo solo lo necesario.
- **Respuesta:** Se indicó qué borrar: variables y funciones de captura por Input Capture, medición de frecuencia, librería CMSIS DSP, mensaje de ayuda `printf`, y parte del `dispatcher`. También se detalló qué conservar: display, FSM, ADC, RGB, UART.

3. Configuración de pines para el LED RGB por PWM

- **Consulta:** Qué pines usar para controlar un LED RGB por PWM y recibir valores por UART (`rgb=255 255 255@`).
- **Respuesta:** Se propuso usar `TIM1` con pines:
 - CH1 → PA8 (Rojo)
 - CH2 → PA9 (Verde)
 - CH3 → PA10 (Azul)

Se entregó el código de inicialización, arranque de PWM y la función `HandleRGBCmd()` para cambiar colores desde UART.

4. Comprobación de lectura de joystick por ADC

- **Consulta:** El display siempre mostraba el mismo valor (24), y se sospechaba que no se leía el eje X.
- **Respuesta:** El problema era que no se estaba separando correctamente `adc_buffer[]` en `vrx_buffer[]` y `vry_buffer[]`. Se explicó cómo implementar correctamente `HAL_ADC_ConvCpltCallback()`.

⚙️ 5. Corregir la configuración del ADC en `MX_ADC1_Init()`

- **Consulta:** STM32CubeMX generaba mal la configuración del canal 7 (faltaba cambiar el canal en el segundo `HAL_ADC_ConfigChannel()`).
- **Respuesta:** Se explicó que debe cambiarse manualmente:

```
c
sConfig.Channel = ADC_CHANNEL_7;
sConfig.Rank = 2;
```

Y que es seguro editar manualmente `main.c` si STM32CubeMX no actualiza bien.

📊 6. Visualización del valor del joystick en el display

- **Consulta:** Cómo mostrar los valores x y y en el display de 7 segmentos, primero YYXX, luego cambiarlo a XXYY.
- **Respuesta:**
 - Se explicó cómo cambiar `display_value = (y_val * 100) + x_val;` a `display_value = (x_val * 100) + y_val;` para invertir el orden.
 - También se explicó que no se deben cambiar los pines físicos del display, solo el cálculo del valor numérico.

🔄 7. Corrección final: `vrx` y `vry` estaban invertidos

- **Consulta:** Confirmaste que `vrx` va al canal 6 y `vry` al canal 7, y querías saber si el código estaba bien.
- **Respuesta:** Se detectó que estaban invertidos en el `HAL_ADC_ConvCpltCallback`. Se entregó el bloque corregido:

```
c
vrx_buffer[i] = adc_buffer[2 * i];           // Canal 6 - VRx
vry_buffer[i] = adc_buffer[2 * i + 1];       // Canal 7 - VRY
```

¿Te gustaría que te prepare un solo archivo `.c` con todo ya integrado y limpio según esta configuración final?