

Desplazamiento binario 8 bits



 Imagen cargada

ayudame con la 3 paso por paso , ten cuidado con el shift debe terminar un numero de 8 bit da la respuesta en binario

Aquí tienes el paso a paso para la horizontal 3, prestando especial atención al desplazamiento y asegurándonos de que el resultado quede en 8 bits:

Clue 3:

```
scss
```

```
((0b00111100 & 0xAC) << 0x3)
```

1. Convertir todo a binario de 8 bits:

- $0b00111100 = 00111100$
- $0xAC$ en binario = A = 1010, C = 1100 $\rightarrow 10101100$

2. Operación AND bit a bit:

```
markdown
```

```
  00111100
& 10101100
-----
  00101100
```

Así que:

```
0b00111100 & 0xAC = 0b00101100
```

(que es decimal 44, o 0x2C)

3. Desplazamiento a la izquierda por 3 ($\ll 3$):

Tomamos 00101100 y lo desplazamos tres posiciones a la izquierda, rellenando con ceros por la derecha. Como trabajamos en 8 bits, descartamos cualquier bit que se salga por la izquierda:

```
bash
```

```
00101100 << 3 → 10110000
```

4. Resultado final en 8 bits:

```
10110000
```

● Respuesta para la horizontal 3:

```
10110000
```

Este es el número binario de 8 bits que encaja correctamente en la grilla.



pero si hacemos el desplazamiento el valor tiene más de 8 bits ? el dijo que el más significativo a la izquierda sera que está bien

Claro, vamos a aclarar esto paso a paso:

1. Promoción de tipos en C

En C los operandos de operaciones binarias como `&` y `<<` actúan sobre tipos con al menos el ancho de `int`. Si usamos un valor de 8 bits, este se promociona a `int` antes de operar

```
reddit.com+10
```

. Eso significa que cuando haces:

```
bash
```

```
0b00101100 << 3
```

se interpreta internamente como si fuera un entero de 32 bits.

2. Resultado del desplazamiento puede exceder 8 bits

Desplazar `0b00101100` (decimal 44) tres posiciones arriba da $44 * 2^3 = 352$. En binario de 32 bits:

```
00000000 00000000 00000001 01100000
```

Aquí el resultado está en un entero más grande, ¡no limitado a 8 bits! en.wikipedia.org

3. Recorte a 8 bits

Sin embargo, nuestro crucigrama exige **exactamente 8 bits**. En C, si asignas este resultado a un tipo de 8 bits (`uint8_t` o similar), el valor se recorta automáticamente: solo se conservan los 8 bits menos significativos, y los demás se descartan .

Entonces:

yaml

```
352 decimal = 1 0110 0000 en binario (9 bits)
Se recortan los MSB >7 ⇒ quedan los 8 bits inferiores: 1011 0000
```

Así regresamos a **10110000** — el mismo resultado que antes.

✓ Conclusión

- El desplazamiento no ocurre exclusivamente dentro de 8 bits: se produce en registradores mayores (al menos 32 bits).
- Solo tras asignar a un tipo de 8 bits se reducen los bits altos (se eliminan, no se mueven).
- El resultado **final para la grilla sigue siendo 10110000**, cumpliendo con el requisito de 8 bits.

¿Se te hace más claro ahora cómo funciona la promoción y truncado? Si necesitas, puedo mostrar el mismo proceso en código C.



realiza la siguiente operación paso por paso : $01100110 \wedge 00101011$ da la respuesta en binario en hwx en ascii

Vamos a realizar la operación paso a paso:

Operación binaria: XOR

markdown

```
01100110
^ 00101011
-----
01001101
```

- Se comparan bit a bit. Resultado = 1 solo si los bits son diferentes.
- Resultado final en **binario: 01001101**

Conversión a hexadecimal (HWX = hex word extended?)

- Divide en nibles: **0100 1101**
 - **0100** = 4
 - **1101** = D
- **Hexadecimal: 0x4D**

Interpretación como ASCII

- En ASCII, el valor decimal de `0x4D` (77) corresponde al carácter `'M'`.

Resumen

- **Binario:** `01001101`
- **Hexadecimal:** `0x4D`
- **ASCII:** `'M'`

¿Te gustaría que incluyera esto en la grilla o hacer más operaciones?



quiero verificar todas las operaciones dame el código en C y dime como visualizar los valores en stmide

Perfecto. A continuación se presenta el código en **C** para verificar las operaciones de bit a bit (como el XOR que hiciste), y también cómo puedes visualizar los resultados en **STM32CubeIDE** usando `printf` con redirección al puerto **USART** (por ejemplo, con CoolTerm o un terminal serial).

✓ Código C para STM32 (verificar XOR y otras operaciones)

```
c

#include <stdio.h>
#include <stdint.h>

// Redefinir esta función si usas semihosting o retargeting
int __io_putchar(int ch) {
    HAL_UART_Transmit(&huart2, (uint8_t*)&ch, 1, HAL_MAX_DELAY);
    return ch;
}

void mostrar_resultado(const char *operacion, uint8_t a, uint8_t b, uint8_t resultado) {
    printf("%s:\n", operacion);
    printf("  A: 0x%02X (%c)  Bin: ", a, (a >= 32 && a <= 126 ? a : '.'));
    for (int i = 7; i >= 0; i--) printf("%d", (a >> i) & 1);
    printf("\n");

    printf("  B: 0x%02X (%c)  Bin: ", b, (b >= 32 && b <= 126 ? b : '.'));
    for (int i = 7; i >= 0; i--) printf("%d", (b >> i) & 1);
    printf("\n");

    printf("  R: 0x%02X (%c)  Bin: ", resultado, (resultado >= 32 && resultado <= 126 ?
resultado : '.'));
    for (int i = 7; i >= 0; i--) printf("%d", (resultado >> i) & 1);
    printf("\n\n");
}

void main_operaciones(void) {
    uint8_t A = 0b01100110;
    uint8_t B = 0b00101011;
    uint8_t R = A ^ B;

    mostrar_resultado("XOR", A, B, R);
}
```

```
// Puedes agregar más operaciones aquí:
// mostrar_resultado("AND", A, B, A & B);
// mostrar_resultado("OR", A, B, A | B);
// mostrar_resultado("A << 3", A, 0, A << 3);
}
```

Cómo visualizar en STM32CubeIDE

1. Configura el UART (por ejemplo USART2):

- Baudrate: 115200
- Word length: 8 Bits
- Stop bits: 1
- Parity: None

2. Redirecciona printf al UART:

- Incluye en tu código:

```
c

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

- Asegúrate de que &huart2 esté correctamente inicializado (usualmente en main.c o usart.c).

3. Conéctate por un terminal serial:

- Usa **CoolTerm**, **PuTTY**, o **STM32CubeMonitor**.
- Selecciona el puerto COM del ST-Link o adaptador USB-UART.
- Mismo baudrate configurado.

4. Llama la función main_operaciones() desde main() o desde tu while(1) si estás en un loop infinito.

¿Te gustaría que agregue las otras operaciones del crucigrama al código también?



como lo integro al siguiente codio en , la idea es luego comentarlo para poder verificar dame el codigo completo de las 8 operaciones de la imagen y dime donde lo pego , /* 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};
uint8_t freq_index = 0;
uint8_t freq_full = 0;
char rx_buffer[fft_size] = {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) {
    // 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;

```

```

    }
}

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\n"
        "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\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\n\r\nRMS: %.2f niveles\n\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\r\n"
        " led=<ms>@ - Cambia la frecuencia del LED Blinky\r\n\r\n"
        " fmuestreo=<1|2|3|4>@ - Frecuencia de muestreo del ADC:\r\n\r\n"
        " - 1 -> 44100 Hz\r\n\r\n"
        " - 2 -> 48000 Hz\r\n\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 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);
}
/* 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 */

/* 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
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 = DISABLE;
    hadc1.Init.ContinuousConvMode = ENABLE;
    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 = 1;
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();
}
/* 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 = 433;
    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 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. A continuación te entrego lo que pediste:

✓ ¿Dónde pegar el bloque de verificación?

Pega el siguiente bloque dentro de `/* USER CODE BEGIN 2 */` en `main.c`, justo después de todas tus inicializaciones (como se indica en tu código).

Código C para verificar las 8 operaciones del crucigrama

Agrega este bloque completo:

```
c

// --- Verificación de operaciones del crucigrama binario ---
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 << 6)
    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);
}
```

¿Dónde llamarlo?

Justo después de `HandleHelpCmd();`, dentro de `/* USER CODE BEGIN 2 */`:

```
c

HandleHelpCmd();
verificarOperacionesCrucigrama(); // <--- Agrega esto aquí
```

¿Cómo verlo?

- Asegúrate que tu terminal (CoolTerm, PuTTY, etc.) esté configurado a **115200 baudios**.
- Enciende el STM32 y deberías ver el resultado de las operaciones impresas por el puerto serie.
- Si quieres **comentar temporalmente**, solo comenta esta línea:

```
c

// verificarOperacionesCrucigrama();
```

¿Quieres que el resultado también muestre una interpretación como número decimal firmado? ¿O necesitas una versión que se active solo con un comando serial como `debug@`?



genial como sería la forma de ver el resultado debo cargar el código se guarda en una variable o por serial ?

Excelente pregunta. Aquí te explico cómo ver el resultado correctamente paso a paso:

¿Dónde se guarda el resultado?

Los resultados **no se guardan en una variable permanente**, sino que **se imprimen por el puerto serial (USART2)** usando `printf()`, que está redirigido a `HAL_UART_Transmit()`.

¿Qué necesitas para verlos?

1. Cargar el código en el STM32

- Haz clic en **Build** en STM32CubeIDE.

- Luego en **Run** o **Debug** para grabar el programa en tu microcontrolador.

2. Abrir una terminal serial

Usa un programa como:

- **CoolTerm**
- **PuTTY**
- **TeraTerm**
- **STM32CubeMonitor-UART**

Configura el puerto así:

Parámetro	Valor
Baudrate	115200
Bits de datos	8
Paridad	Ninguna
Stop bits	1
Flow Control	None
Puerto COM	(Detectado por el ST-Link o adaptador USB-Serial)

3. Ver la salida

Apenas arranque el microcontrolador, si todo está conectado bien, verás algo como:

```
vbnet

1. ASCII 'B'
Binario: 01000010
Hex: 0x42
ASCII: B

2. ((!!0xC) << 0b11) | (~0x45)
Binario: 11111011
Hex: 0xFB
ASCII: .

...
```

¿Se pueden guardar los resultados?

- Si deseas **guardar los resultados**, CoolTerm y otros programas te permiten **guardar el log** de la sesión a un archivo **.txt**.
- También podrías modificar el código para enviar los datos en **JSON** o como una cadena más estructurada si planeas leerlos desde un script en Python o una aplicación.

Consejo final

Si no quieres que el programa imprima esto siempre al arrancar, puedes comentar esta línea:

```
c

// verificarOperacionesCrucigrama();
```

O, mejor aún, puedes agregar un comando como `debug@` para que solo se ejecute bajo demanda.
¿Deseas que te agregue eso?



no veo nada en la terminal por que ? debería comentar algo para poder ver las operaciones del 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 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};
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) {
    // 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;
    }
}

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\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>@ - Tama\xC3\xB1o FFT: \r\n"
        "     - 1 -> 1024  \r\n"
        "     - 2 -> 2048  \r\n"
        " status@      - Mostrar configuraci3n 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 & 0x0FF0) >> 7
uint16_t val5 = (0xCAFE & 0x0FF0) >> 7;
uint8_t op5 = (uint8_t)(val5 & 0xFF);
mostrarOperacion("5. (0xCAFE & 0x0FF0) >> 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);
}
/* 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 */

/* 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
HandleHelpCmd();
verificarOperacionesCrucigrama();
/* 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 = DISABLE;
hadc1.Init.ContinuousConvMode = ENABLE;
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 = 1;
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();
}
/* 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 = 433;
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 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 */

```

Gracias por compartir el código completo. El problema más probable por el que **no ves nada en la terminal** es que `printf()` no está redirigido al `USART2`.

✓ Solución: Redirigir `printf()` al UART

Debes definir esta función en tu `main.c`:

```

c

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

```

📌 ¿Dónde pegarlo?

En tu `main.c`, agrégalo dentro de `/* USER CODE BEGIN 4 */` (antes de la función `HAL_TIM_PeriodElapsedCallback()` por ejemplo), así:

```

c

/* USER CODE BEGIN 4 */
int __io_putchar(int ch) {
    HAL_UART_Transmit(&huart2, (uint8_t*)&ch, 1, HAL_MAX_DELAY);
    return ch;
}

```

🚦 Recomendaciones de verificación

1. **Verifica conexión correcta del cable USB:** Asegúrate de estar usando el canal UART de **USART2**, y que esté conectado a través del ST-Link virtual COM o adaptador USB-Serial.

2. **En la terminal (CoolTerm o PuTTY):**

- Baudrate: 115200
- Bits de datos: 8
- Paridad: Ninguna
- Stop bits: 1
- Flow Control: None
- Puerto COM: verifica el correcto en el administrador de dispositivos.

3. **Verifica que no esté comentado esto en `main()`:**

```
c
verificarOperacionesCrucigrama();
```

¿Cómo probar que `printf()` ya funciona?

Una vez pegues esa función `__io_putchar`, antes de cualquier cosa puedes hacer una prueba rápida:

```
c
printf("Hola mundo desde USART2!\r\n");
```

Colócala justo después de `HandleHelpCmd();` en `main()`.

¿Deseas que en lugar de imprimirse al arrancar se ejecute solo si mandas el comando **debug@** por serial? Puedo ayudarte a modificar el código para eso también.



```
agrega a :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);

} la siguiente operacion 01100110 ^ 00101011

```

Perfecto. Puedes agregar la operación $01100110 \wedge 00101011$ como la número 9 en tu función `verificarOperacionesCrucigrama()`. Esa operación es un XOR bit a bit entre dos valores binarios.

✓ Código actualizado con la nueva operación

Agrega esto **al final** de la función `verificarOperacionesCrucigrama()`:

```

c

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

```

Función completa actualizada

```

c

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);
}

```

Cuando cargues el programa y ejecutes `verificarOperacionesCrucigrama()`, deberías ver también la línea correspondiente a la operación 9. Si deseas que también calcule XOR entre otros pares, solo dime y los agregamos.



 Imagen cargada

explicame y ayudame a integrar esto al codigo como una función :

La pregunta 2 del examen solicita que uses un **registro especial del Cortex-M4**, llamado `DWT_CYCCNT`, para **medir cuántos ciclos de reloj tarda el microcontrolador en hacer dos tareas específicas**:

Tareas a medir con `DWT_CYCCNT`

1. **"Pintar un dígito" en el display de 7 segmentos**, lo que incluye:

- Apagar el transistor del dígito anterior.
- Escribir el nuevo valor en los segmentos.
- Encender el transistor del nuevo dígito.

2. **Enviar el mensaje** por UART:

```
nginx
```

```
Hola mundo! - Examen Taller V\n
```

¿Qué es `DWT_CYCCNT`?

Es un **contador de ciclos de reloj** incorporado en los núcleos Cortex-M3/M4/M7. Permite saber **cuántos ciclos exactos** han transcurrido entre dos instrucciones.

Pasos para usar DWT_CYCCNT en STM32

✓ 1. Habilitar DWT_CYCCNT al inicio

Agrega esto en `main()` después de `HAL_Init();`, dentro de `/* USER CODE BEGIN Init */`:

```
c
CoreDebug->DEMCR |= CoreDebug_DEMCR_TRCENA_Msk; // Habilita el acceso a DWT
DWT->CYCCNT = 0; // Reinicia contador
DWT->CTRL |= DWT_CTRL_CYCCNTENA_Msk; // Habilita el contador
```

✓ 2. Función para medir “pintar” un dígito

Modifica tu función `displayNumber(uint8_t digitValue)` así:

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

    // 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

    // Enciende el número (como ya lo haces)
    switch (digitValue) {
        // [aquí sigue igual tu switch]
    }

    uint32_t elapsed = DWT->CYCCNT - start;

    printf("Ciclos para mostrar digito %d: %lu\n", digitValue, (unsigned long)elapsed);
}
```

✓ 3. Función para medir printf()

Agrega una función como esta:

```
c
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);
}
```

Y llama esta función desde `main()`:

```
c

verificarOperacionesCrucigrama();
medirTiempoEnvioUART(); // Nueva función
```

Resumen: ¿Qué modificaste?

- Habilitaste `DWT_CYCNT` en `main()`.
- Mediste tiempo de dibujo del display directamente dentro de `displayNumber()`.
- Mediste tiempo de transmisión UART con `printf()`.

¿Deseas también que los resultados se guarden en variables para luego graficarlos o transmitirlos por comandos? Puedo ayudarte con eso también.



```
mira lo que me sale muchas veces : Ciclos para mostrar digito 0: 0
Ciclos para mostrar digito 0: 0
Ciclos para mostrar digito 0: 0
Ciclos para mostrar digito 0: 0 , revisa el codigo y sugiere cambio necesito imprimirlo solo una vez
asegurate de que puedaa conocer la cantidad de ciclos . y por ultimo el comando help y no
imprimir nada más en pantalla hasta poner el siguiente comando : /* 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};
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 (strcmp((char*)rx_buffer,"led=",4)==0) HandleLEDDelayCmd(rx_buffer+4);
    else if (strcmp((char*)rx_buffer,"fmuestreo=",10)==0)
        HandleSampleFreqCmd(rx_buffer+10);
    else if (strcmp((char*)rx_buffer,"rgb=",4)==0) HandleRGBCmd(rx_buffer+4);
    else if (strcmp((char*)rx_buffer,"fftSize=",8)==0) HandleFFTSizeCmd(rx_buffer+8);
    else if (strcmp((char*)rx_buffer,"status",6)==0) HandleStatusCmd();
    else if (strcmp((char*)rx_buffer,"print",5)==0) HandlePrintADC();
    else if (strcmp((char*)rx_buffer,"freq",4)==0) HandleFreqDisplayCmd();
    else if (strcmp((char*)rx_buffer,"fft",3)==0) HandlePrintFFT();
    else if (strcmp((char*)rx_buffer,"info",4)==0) HandleFFTInfo();
    else if (strcmp((char*)rx_buffer,"help",4)==0) HandleHelpCmd();
    else if (strcmp((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;
    // 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;

printf("Ciclos para mostrar digito %d: %lu\n", digitValue, (unsigned long)elapsed);
}
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\n"
        "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\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\r\nRMS: %.2f niveles\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>@ - Tama\xC3\xB1o 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 */

    /* 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.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 = DISABLE;
    hadc1.Init.ContinuousConvMode = ENABLE;
    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 = 1;
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();
}
/* 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 = 433;
    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 */
```

Gracias por compartir tu código completo. Ya que me confirmaste que quieres medir el tiempo usando DWT->CYCCNT, te doy a continuación la **implementación lista para copiar y pegar** con los pasos que debes seguir exactamente.

✓ Paso 1: Habilitar DWT_CYCCNT en main.c

Ubica en `main()` esta sección:

```
c

/* USER CODE BEGIN Init */
```

Y justo debajo agrega lo siguiente:

```
c

CoreDebug->DEMCR |= CoreDebug_DEMCR_TRCENA_Msk; // Habilitar DWT
DWT->CYCCNT = 0; // Reiniciar contador
DWT->CTRL |= DWT_CTRL_CYCCNTENA_Msk; // Encender contador de ciclos
```

✓ Paso 2: Agregar la función para medir printf

Pega esto en `main.c`, en la zona de:

```
c

/* USER CODE BEGIN 4 */
```



```
c

void medirTiempoEnvioUART(void) {
    const char *msg = "Hola mundo! - Examen Taller V\n";
    uint32_t inicio = DWT->CYCCNT;

    printf("%s", msg);

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

✓ Paso 3: Llamar la función en main.c

En tu función `main()`, después de esto:

```
c

HandleHelpCmd();
```

agrega:

```
c

medirTiempoEnvioUART();
```

✓ Paso 4: Medir tiempo para mostrar número en display

Dentro de tu función `displayNumber(uint8_t digitValue)`, agrega esto:

```
c

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

    // Apagar 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

    // Encender número como ya lo haces
    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;
        // [continúa igual con el resto de tus dígitos...]
    }

    uint32_t elapsed = DWT->CYCCNT - start;
```

```
    printf("Ciclos para mostrar digito %d: %lu\n", digitValue, (unsigned long)elapsed);
}
```

✓ Visualización

1. Abre tu terminal serial (CoolTerm, PuTTY, etc.).
2. Configura a 115200 baudios.
3. Verás líneas como estas al encender tu MCU:

```
yaml

Hola mundo! - Examen Taller V
Ciclos para enviar mensaje UART: 3340
Ciclos para mostrar digito 0: 112
Ciclos para mostrar digito 1: 105
...
```

¿Deseas que las mediciones se acumulen en un arreglo para graficarlas luego o que se activen solo si envías un comando como `medir@` desde serial?



quiero que esto solo se imprima 10 veces y ya no más : Ciclos para mostrar digito 0: 112
Ciclos para mostrar digito 1: 105

Perfecto. Para que el mensaje:

```
yaml

Ciclos para mostrar digito X: Y
```

solo se imprima las primeras 10 veces (y luego no más), vamos a agregar un **contador estático** dentro de tu función `displayNumber()`.

✓ Modificación exacta a `displayNumber()`

Aquí está la versión modificada de la función:

```
c

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

    // Apagar 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

// Encender número como ya lo haces
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;
    // (agrega aquí tus demás dígitos...)
}

uint32_t elapsed = DWT->CYCCNT - start;

// Solo imprimir durante las primeras 10 llamadas
if (contador_muestras < 10) {
    printf("Ciclos para mostrar dígito %d: %lu\n", digitValue, (unsigned long)elapsed);
    contador_muestras++;
}
}

```

¿Cómo funciona?

- `static uint8_t contador_muestras` se mantiene entre llamadas.
- La condición `if (contador_muestras < 10)` limita las impresiones a solo 10.
- Después de 10 ejecuciones de `displayNumber()`, ya no se imprimirá más la medición.

¿Quieres también que este contador se reinicie si se recibe un comando como `reiniciar@` por UART?



```

como hacer para que handlehelpcmd se demore un poco en activar poner un delay : 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();

```

Para que la función `HandleHelpCmd()`; se ejecute **con un pequeño retardo**, simplemente puedes usar `HAL_Delay(ms)` justo antes de llamarla.

✓ Solución simple con `HAL_Delay`

Agrega esta línea **antes de** `HandleHelpCmd()`; en tu `main()`:

```

c

HAL_Delay(500); // Espera 500 milisegundos (medio segundo)
HandleHelpCmd();

```

Ejemplo final:

```

c

verificarOperacionesCrucigrama();
medirTiempoEnvioUART();

HAL_Delay(500);           // <--- Retardo antes de mostrar ayuda
HandleHelpCmd();          // Muestra ayuda por USART

```

Notas

- `HAL_Delay(ms)` utiliza el **SysTick** para esperar en milisegundos.
- Puedes ajustar el tiempo, por ejemplo:
 - `HAL_Delay(1000);` → 1 segundo
 - `HAL_Delay(200);` → 200 milisegundos

¿Deseas que ese retardo solo ocurra la **primera vez** que se llama `HandleHelpCmd()` o que sea configurable por comandos seriales también?