UNIVERSIDAD NACIONAL DE COLOMBIA
SEDE MANIZALES

Práctica: Manual Room Control Bare Metal with C

Santiago Galeano Castaño

Estructuras Computacionales

2025-1

Universidad Nacional de Colombia – Sede Manizales

UNIVERSIDAD NACIONAL DE COLOMBIA
SEDE MANIZALES

Empezamos analizando los códigos paso a paso:

Apartado RCC

Iniciando con la estructura de los registros de rcc.h

```c
#ifndef RCC_H
#define RCC_H

#include <stdint.h>
#include "gpio.h"


typedef struct {
    volatile uint32_t CR;
    volatile uint32_t ICSCR;
    volatile uint32_t CFGR;
    volatile uint32_t PLLCFGR;
    volatile uint32_t PLLSAI1CFGR;
    volatile uint32_t PLLSAI2CFGR;
    volatile uint32_t CIER;
    volatile uint32_t CIFR;
    volatile uint32_t CICR;
    uint32_t RESERVED1;
    volatile uint32_t AHB1RSTR;
    volatile uint32_t AHB2RSTR;
    volatile uint32_t AHB3RSTR;
    uint32_t RESERVED2;
    volatile uint32_t APB1RSTR1;
    volatile uint32_t APB1RSTR2;
    volatile uint32_t APB2RSTR;
    uint32_t RESERVED3;
    volatile uint32_t AHB1ENR;
    volatile uint32_t AHB2ENR;
    volatile uint32_t AHB3ENR;
    uint32_t RESERVED4;
    volatile uint32_t APB1ENR1;
    volatile uint32_t APB1ENR2;
    volatile uint32_t APB2ENR;
} RCC_TypeDef;

```

Esta estructura define los registros del periférico RCC en el microcontrolador.

Cada campo representa un registro específico del RCC.

volatile indica que estos registros pueden cambiar fuera del control del programa (son hardware).

UNIVERSIDAD NACIONAL DE COLOMBIA
SEDE MANIZALES

Haciendo un mapeo de Memoria

```
36  #define RCC_BASE (0x40021000U)
37  #define RCC ((RCC_TypeDef *)RCC_BASE)
```

RCC_BASE es la dirección base del periférico RCC en la memoria del microcontrolador.

RCC es una conexión a la estructura de registros, mapeado a la dirección base.

Configurando las frecuencias

```
39  // Macros
40  #define SYSCLK_FREQ_HZ    4000000UL      // 4MHz
41  #define HCLK_FREQ_HZ      SYSCLK_FREQ_HZ // HCLK Prescaler = 1
42  #define PCLK1_FREQ_HZ     HCLK_FREQ_HZ   // APB1 Prescaler = 1
43  #define PCLK2_FREQ_HZ     HCLK_FREQ_HZ   // APB2 Prescaler = 1
44  #define TIM_PCLK_FREQ_HZ  PCLK1_FREQ_HZ  // TIM3 está en APB1
```

Aquí se definen las frecuencias de reloj del sistema y buses.

En este caso, todas funcionan a 4MHz.

Prototipos de las funciones

```
46  // Prototipos de funciones
47  void rcc_gpio_clock_enable(GPIO_TypeDef *gpio_port);
48  void rcc_syscfg_clock_enable(void);
49  void rcc_usart2_clock_enable(void);
50  void rcc_tim3_clock_enable(void);
```

Declaraciones de funciones para habilitar relojes de diferentes periféricos.

UNIVERSIDAD NACIONAL DE COLOMBIA
SEDE MANIZALES

Mirando las estructuras del registro rcc.c

rcc gpio clock enable

```
3   void rcc_gpio_clock_enable(GPIO_TypeDef *gpio_port)
4   {
5       if (gpio_port == GPIOA) {
6           RCC->AHB2ENR |= 0x01 << 0;
7       } else if (gpio_port == GPIOB) {
8           RCC->AHB2ENR |= 0x01 << 1;
9       } else if (gpio_port == GPIOC) {
10          RCC->AHB2ENR |= 0x01 << 2;
11      }
```

Aquí se recibe un puntero al puerto GPIO (A, B o C).

Según el puerto, se activa el bit correspondiente en el registro AHB2ENR.

¿Esto para qué? para habilitar el reloj para el puerto GPIO especificado.

rcc syscfg clock enable

```
15  void rcc_syscfg_clock_enable(void)
16  {
17      RCC->APB2ENR |= 0x01 << 0; // SYSCFG clock enable
18  }
```

Aquí activa el bit 0 del registro APB2ENR, habilitando así el reloj para el módulo SYSCFG.

rcc usart2 clock enable

```
20  void rcc_usart2_clock_enable(void)
21  {
22      RCC->APB1ENR1 |= 0x01 << 17; // USART2 clock enable
23  }
```

Activa el bit 17 del registro APB1ENR1, habilitando el reloj para el periférico USART2.

Santiago Galeano Castaño

UNIVERSIDAD NACIONAL DE COLOMBIA
SEDE MANIZALES

rcc tim3 clock enable

```
25  void rcc_tim3_clock_enable(void)
26  {
27      RCC->APB1ENR1 |= 0x01 << 1; // TIM3 clock enable
28  }
```

Activa el bit 1 del registro APB1ENR1, habilitando el reloj para el temporizador TIM3.

Apartado Systick

Iniciando con la estructura de los registros de systick.h

```
5   typedef struct {
6       volatile uint32_t CTRL;  // Control and Status Register, Offset: 0x00
7       volatile uint32_t LOAD;  // Reload Value Register,       Offset: 0x04
8       volatile uint32_t VAL;   // Current Value Register,      Offset: 0x08
9       volatile uint32_t CALIB; // Calibration Register,        Offset: 0x0C
10  } SysTick_TypeDef;
```

Define los 4 registros del periférico SysTick:

- **CTRL**: Registro de control y estado.
- **LOAD**: Valor de recarga (define el intervalo).
- **VAL**: Valor actual del contador.
- **CALIB**: Registro de calibración (para ajuste fino).

Mapeo de memoria

```
13  #define SYSTICK_BASE        (0xE000E010UL)
14  #define SysTick             ((SysTick_TypeDef *)SYSTICK_BASE)
```

SYSTICK_BASE es la dirección base del periférico SysTick (0xE000E010).

SysTick es una conexión a la estructura de registros.

UNIVERSIDAD NACIONAL DE COLOMBIA
SEDE MANIZALES

Haciendo un mapeo de Memoria

```
16  // Prototipos de funciones
17  void systick_init_1ms(void);
18  uint32_t systick_get_tick(void);
19  void systick_delay_ms(uint32_t ms);
```

systick_init_1ms, inicializa SysTick para generar interrupciones cada 1ms.

systick_get_tick, obtiene el contador actual de milisegundos.

systick_delay_ms, implementa un retardo bloqueante en milisegundos.

Mirando las estructuras del registro systick.c

Variable Global

```
8   static volatile uint32_t g_systick_ms_count = 0;
```

Es el contador global en milisegundos. **volatile** es importante porque:

Se modifica en la interrupción ISR.

Eso sí, se lee en el código principal, evitando optimizaciones del compilador que podrían causar problemas.

UNIVERSIDAD NACIONAL DE COLOMBIA
SEDE MANIZALES

Systick init 1ms

```c
void systick_init_1ms(void)
{
    // 1. Calcular el valor de recarga para 1 ms
    uint32_t reload_value = (SYSCLK_FREQ_HZ / 1000U) - 1U; // (4000000 / 1000) - 1 = 3999

    // 2. Configurar el registro de recarga (SysTick_LOAD)
    SysTick->LOAD = reload_value & 0x00FFFFFFUL;

    // 3. Poner a cero el valor actual del temporizador (SysTick_VAL)
    SysTick->VAL = 0x00000000UL;

    // 4. Configurar el registro de control (SysTick_CTRL)
    SysTick->CTRL = (0x01 << 2) | // Usa reloj del procesador (HCLK)
                    (0x01 << 1) | // Habilita interrupción de SysTick
                    (0x01 << 0) ; // Habilita el contador SysTick
}
```

1. **Cálculo del valor de recarga:** Se sabe que **SYSCLK_FREQ_HZ** es 4MHz (unos 4,000,000 de ciclos)
   Para 1ms, nos da 4000.
   Restando 1 porque el contador llega a 0, sería 3999.
2. **Configuración de LOAD:** Asigna el valor calculado (en 24 bits válidos)
   El dato **0x00FFFFFF** asegura que solo se usen 24 bits.
3. **Reiniciar VAL:** Al escribir cualquier valor, se limpia el contador.
4. **Configuración de CTRL:** El bit 2, selecciona el reloj del procesador (HCLK)
   El bit 1, habilita las interrupciones.
   El bit 0, habilita el contador.

Systick get tick

```c
uint32_t systick_get_tick(void)
{
    return g_systick_ms_count;
}
```

Simplemente devuelve el valor actual del contador global.

Permite conocer el tiempo actual del sistema.

UNIVERSIDAD NACIONAL DE COLOMBIA SEDE MANIZALES

Systick delay ms

```
34  void systick_delay_ms(uint32_t ms)
35  {
36      uint32_t start_tick = systick_get_tick();
37      while ((systick_get_tick() - start_tick) < ms) {
38          // Espera hasta que hayan transcurridos los milisegundos deseados
39      }
40  }
```

Guarda el valor actual del contador **Start_tick.**

Luego entra en bucle, haciendo que:

- Calcula la diferencia entre el tiempo actual y el inicial.
- Continúa hasta que la diferencia sea menor o igual al retardo solicitado.

Systick handler

```
43  void SysTick_Handler(void)
44  {
45      g_systick_ms_count++; // Incrementar el contador de ticks global
46  }
```

Función de interrupción (se ejecuta automáticamente cada 1ms).

Va incrementando el contador global de milisegundos.

Apartado GPIO

Iniciando con la estructura de los registros de gpio.h

```c
typedef struct {
    volatile uint32_t MODER;   // 00: Input, 01: Output, 10: Alternate, 11: Analog
    volatile uint32_t OTYPER;  // 0: Push-Pull, 1: Open Drain
    volatile uint32_t OSPEEDR; // 00: Low speed, 01: Medium, 10: High, 11: Very High
    volatile uint32_t PUPDR;   // 00: No-Pull, 01: Pull-Up, 10: Pull-Down
    volatile uint32_t IDR;     // 0: Reset (low), 1: Set (High)
    volatile uint32_t ODR;     // 0: Reset (low), 1: Set (High)
    volatile uint32_t BSRR;
    volatile uint32_t LCKR;
    volatile uint32_t AFRL;    // To connect a pin(0-7) to a peripheral like UART, SPI, PWM, etc
    volatile uint32_t AFRH;    // To connect a pin(8-15) to a peripheral like UART, SPI, PWM, etc
} GPIO_TypeDef;
```

Cada campo representa un registro específico del periférico GPIO.

volatile indica que son registros de hardware que pueden cambiar asíncronamente.

Mapeo de puertos GPIO

```c
#define GPIOA_BASE (0x48000000U)
#define GPIOB_BASE (0x48000400U)
#define GPIOC_BASE (0x48000800U)

#define GPIOA ((GPIO_TypeDef *)GPIOA_BASE)
#define GPIOB ((GPIO_TypeDef *)GPIOB_BASE)
#define GPIOC ((GPIO_TypeDef *)GPIOC_BASE)
```

Define las direcciones base de los puertos GPIO A, B y C.

Crea punteros a estructuras para acceder fácilmente a los registros.

Definiciones de modos y estados

```c
// Modos de Pin (para GPIOx_MODER)
#define GPIO_MODE_INPUT     0x00U // 00: Input mode (reset state)
#define GPIO_MODE_OUTPUT    0x01U // 01: General purpose output mode
#define GPIO_MODE_AF        0x02U // 10: Alternate function mode
#define GPIO_MODE_ANALOG    0x03U // 11: Analog mode

// Estados de Pin
#define GPIO_PIN_RESET      0U
#define GPIO_PIN_SET        1U
```

Modo entrada.

Modo salida.

Función alternativa.

Modo analógico.

Estado bajo.

Estado alto.

Definición de pines específicos

```
#define HEARTBEAT_LED_PIN           5     // PA5 para LD2
#define HEARTBEAT_LED_PORT       GPIOA // Ya conocido por el driver GPIO

#define EXTERNAL_LED_PWM_PIN        6     // PA6 (TIM3_CH1)
#define EXTERNAL_LED_PWM_PORT    GPIOA

#define EXTERNAL_LED_ONOFF_PIN      7     // PA7 para emulacion de estado de puerta
#define EXTERNAL_LED_ONOFF_PORT  GPIOA

#define USER_BUTTON_PIN             13    // PC13 para B1
#define USER_BUTTON_PORT         GPIOC
```

Asigna los nombres significativos a los pines específicos.

Mirando las estructuras del registro gpio.c

Gpio setup pin

```c
void gpio_setup_pin(GPIO_TypeDef *gpio_port, uint8_t pin_number,
                    uint8_t mode, uint8_t alternate_function)
{

    // 1. Habilitar el reloj para el puerto GPIO correspondiente
    //    La función rcc_gpio_clock_enable se encarga de esto.
    rcc_gpio_clock_enable(gpio_port);

    // 2. Configurar el modo del pin (Input, Output, AF, Analog)
    //    Cada pin usa 2 bits en MODER. Limpiar y luego establecer.
    gpio_port->MODER &= ~(0x03U << (pin_number * 2)); // Limpiar los 2 bits del pin
    gpio_port->MODER |= (mode << (pin_number * 2));   // Establecer el modo

    // 3. Configurar la función alternativa
    //    Solo si es modo AF. Cada pin usa 4 bits.
    //    Pines 0-7 usan AFRL, pines 8-15 usan AFRH.
    if (mode == GPIO_MODE_AF) {
        uint32_t temp_af_val = alternate_function;
        if (pin_number < 8) { // AFRL
            gpio_port->AFRL &= ~(0x0FU << (pin_number * 4));          // Limpiar los 4 bits del pin
            gpio_port->AFRL |= (temp_af_val << (pin_number * 4)); // Establecer AF
        } else { // AFRH
            gpio_port->AFRH &= ~(0x0FU << ((pin_number - 8) * 4));       // Limpiar los 4 bits del pin
            gpio_port->AFRH |= (temp_af_val << ((pin_number - 8) * 4)); // Establecer AF
        }
    }
}
```

Habilita el reloj para el puerto GPIO (requisito para cualquier operación GPIO)

Configura el modo del pin (cada pin usa 2 bits en MODER)

Si el modo es función alternativa, configura los registros AFRL/AFRH (4 bits por pin)

Gpio write pin

```c
void gpio_write_pin(GPIO_TypeDef *gpio_port, uint8_t pin_number, uint8_t pin_state)
{
    if (pin_state == GPIO_PIN_SET) {
        // Establecer el bit correspondiente en BSRR (parte baja para SET)
        gpio_port->BSRR = (1U << pin_number);
    } else {
        // Establecer el bit correspondiente en BSRR (parte alta para RESET)
        gpio_port->BSRR = (1U << (pin_number + 16));
    }
}
```

Usa el registro BSRR (Bit Set/Reset Register) para cambiar el estado del pin.

Bits 0-15: Set (1 = poner pin a HIGH)

Bits 16-31: Reset (1 = poner pin a LOW)

Santiago Galeano Castaño

UNIVERSIDAD NACIONAL DE COLOMBIA
SEDE MANIZALES

Gpio toggle pin

```
44  void gpio_toggle_pin(GPIO_TypeDef *gpio_port, uint8_t pin_number)
45  {
46      gpio_port->ODR ^= (1U << pin_number);
47  }
```

Usa un XOR para invertir el estado actual del pin (ODR = Output Data Register)

Gpio read pin

```
49  uint8_t gpio_read_pin(GPIO_TypeDef *gpio_port, uint8_t pin_number)
50  {
51      // Leer el bit correspondiente del Input Data Register (IDR)
52      if ((gpio_port->IDR >> pin_number) & 0x01U) {
53          return GPIO_PIN_SET;
54      } else {
55          return GPIO_PIN_RESET;
56      }
57  }
```

Aquí lee el estado del pin desde IDR (Input Dara Register)

EXTI15 10 IRQHandler

```
65  void EXTI15_10_IRQHandler(void) {
66      // 1. Verificar si la interrupción fue de la línea EXTI13
67      if ((EXTI->PR1 & (1U << 13)) != 0) {
68          // 2. Limpiar el flag de pendiente de la interrupción (escribiendo '1')
69          EXTI->PR1 |= (1U << 13);
70          // 3. Procesar boton presionado
71          room_control_on_button_press();
72      }
73  }
```

Verifica si la interrupción fue generada por el pin 13 (botón)

Limpia el flag de interrupción (escribiendo 1)

Santiago Galeano Castaño

Llama a la función de manejo del botón

Apartado UART

Iniciando con la estructura de los registros de uart.h

```
// Estructura para los registros de USART
...
typedef struct {
    volatile uint32_t CR1;
    volatile uint32_t CR2;
    volatile uint32_t CR3;
    volatile uint32_t BRR;
    volatile uint32_t GTPR;
    volatile uint32_t RTOR;
    volatile uint32_t RQR;
    volatile uint32_t ISR;
    volatile uint32_t ICR;
    volatile uint32_t RDR;
    volatile uint32_t TDR;
} USART_TypeDef;
```

Cada campo corresponde a un registro específico del periférico USART.

Los registros más importantes son:

- **CR1**: Configuración básica (habilitación, bits de datos, etc.)
- **BRR**: Configuración del baud rate
- **ISR**: Estado e interrupciones
- **RDR/TDR**: Registros de datos recibidos/transmitidos

Mapeo de Memoria

```
#define USART2_BASE          (0x40004400UL)
#define USART2               ((USART_TypeDef *) USART2_BASE)
```

Define la dirección base de USART2 y crea un puntero a la estructura de registros.

UNIVERSIDAD NACIONAL DE COLOMBIA
SEDE MANIZALES

Prototipos de Funciones

```
25  // Prototipos de funciones
26  void uart2_init(uint32_t baud_rate);
27
28  void uart2_send_char(char c);
29  void uart2_send_string(const char *str);
```

uart2_init, inicializa el periférico USART2.

uart2_send_char/send_string, son funciones para transmisión de datos.

Mirando las estructuras del registro uart.c

Uart2 init

```
12  void uart2_init(uint32_t baud_rate)
13  {
14      // 1. Configurar pines PA2 (TX) y PA3 (RX) como Alternate Function (AF7)
15      gpio_setup_pin(GPIOA, 2, GPIO_MODE_AF, 7);
16      gpio_setup_pin(GPIOA, 3, GPIO_MODE_AF, 7);
17
18      // 2. Habilitar el reloj para USART2
19      rcc_usart2_clock_enable();
20
21      // 3. Configurar USART2
22      //    Deshabilitar USART antes de configurar (importante si se reconfigura)
23      USART2->CR1 &= ~(0x01 << 0);
24
25      // Configurar Baud Rate (USARTDIV en BRR)
26      // USARTDIV = fCK_USART / BaudRate
27      uint32_t usart_div = (PCLK1_FREQ_HZ + (baud_rate / 2U)) / baud_rate; // Con redondeo
28      USART2->BRR = usart_div;
29
30      // Habilitar Transmisor (TE) y Receptor (RE)
31      USART2->CR1 |= (0x01 << 2 | 0x01 << 3);
32
33      // Finalmente, habilitar USART (UE bit en CR1)
34      USART2->CR1 |= 0x01 << 0;
35  }
```

1. **Configuración de pines GPIO:** PA2 (TX) y PA3 (RX) se configuran como función alternativa (AF7)
   Esto conecta físicamente los pines al periférico USART2.
2. **Habilitación del reloj:** El reloj del periférico debe estar habilitado antes de cualquier configuración.
3. **Configuración del baud rate:** USARTDIV = PCLK1_FREQ_HZ / baud_rate.
   Se añade **(baud_rate / 2)** para redondeo automático.
   El valor calculado se escribe en BRR (Baud Rate Register)
4. **Habilitación de transmisor/receptor:** Bit TE (Transmitter Enable), habilita TX
   Bit RE (Receiver Enable), habilita RX
5. **Habilitación final del USART:** Bit UE (USART Enable), activa el periférico.

Uart2 send char

```
37  void uart2_send_char(char c)
38  {
39      // Esperar hasta que el buffer de transmisión (TDR) esté vacío.
40      // Se verifica el flag TXE (Transmit data register empty) en el registro ISR.
41      while (!(USART2->ISR & USART_ISR_TXE));
42
43      // Escribir el dato en el Transmit Data Register (TDR).
44      USART2->TDR = (uint8_t)c;
45  }
```

Espera hasta que el registro de transmisión esté vacío (flag TXE).

Escribe el carácter en el registro de transmisión (TDR).

Uart3 send string

```
47  void uart2_send_string(const char *str)
48  {
49      while (*str != '\0') {
50          uart2_send_char(*str);
51          str++;
52      }
53  }
```

Itera sobre la cadena de caracteres, enviando uno por uno

Usa uart2_send_char internamente

Se detiene al encontrar el carácter nulo ('\0')

USART2 IRQHandler

```
55  void USART2_IRQHandler(void)
56  {
57      // Verificar si la interrupción fue por RXNE (dato recibido y RDR no vacío)
58      if (USART2->ISR & USART_ISR_RXNE) {
59          // Leer el dato del RDR. Esta acción usualmente limpia el flag RXNE.
60          char received_char = (char)(USART2->RDR & 0xFF);
61          uart2_send_char(received_char); // Eco del carácter recibido
62          // Procesar el carácter recibido.
63          room_control_on_uart_receive(received_char);
64      }
65  }
```

Verifica si la interrupción fue por recepción de dato (flag RXNE)

Lee el dato del registro RDR (esto limpia automáticamente RXNE)

Opcionalmente envía un eco del carácter recibido.

Llama a la función de procesamiento del dato.

Apartado TIM

Iniciando con la estructura de los registros de tim.h

```c
typedef struct {
    volatile uint32_t CR1;
    volatile uint32_t CR2;
    volatile uint32_t SMCR;
    volatile uint32_t DIER;
    volatile uint32_t SR;
    volatile uint32_t EGR;
    volatile uint32_t CCMR1;
    volatile uint32_t CCMR2;
    volatile uint32_t CCER;
    volatile uint32_t CNT;
    volatile uint32_t PSC;
    volatile uint32_t ARR;
    volatile uint32_t RESERVED1;
    volatile uint32_t CCR1;
    volatile uint32_t CCR2;
    volatile uint32_t CCR3;
    volatile uint32_t CCR4;
    volatile uint32_t RESERVED2;
    volatile uint32_t DCR;
    volatile uint32_t DMAR;
} TIM_TypeDef;
```

Cada campo corresponde a un registro específico del periférico TIM.

Registros clave para PWM:

- **PSC**: Prescaler (divisor de frecuencia)
- **ARR**: Auto-reload (define el periodo)
- **CCR1**: Compare register (define el duty cycle)
- **CCMR1**: Configuración del modo PWM.
- **CCER**: Habilitación de canales.

Mapeo de memoria

```c
#define TIM3_BASE           (0x40000400UL) // TIM3 está en APB1
#define TIM3                ((TIM_TypeDef *) TIM3_BASE)
```

Define la dirección base de TIM3 y crea un puntero a la estructura de registros.

UNIVERSIDAD NACIONAL DE COLOMBIA
SEDE MANIZALES

Prototipos de funciones

```
33  // Prototipos de funciones
34  void tim3_ch1_pwm_init(uint32_t pwm_freq_hz);
35
36  void tim3_ch1_pwm_set_duty_cycle(uint8_t duty_cycle_percent); // duty_cycle en % (0-100)
```

tim3_ch1_pwm_init, configura TIM3 en modo PWM en el canal 1.

tim3_ch1_pwm_set_duty_cycle, ajusta el ciclo de trabajo del PWM.

Mirando las estructuras del registro tim.c

tim3 ch1 pwm init

```
5   void tim3_ch1_pwm_init(uint32_t pwm_freq_hz)
6   {
7       // 1. Configurar PA6 como Alternate Function (AF2) para TIM3_CH1
8       gpio_setup_pin(GPIOA, 6, GPIO_MODE_AF, 2);
9
10      // 2. Habilitar el reloj para TIM3
11      rcc_tim3_clock_enable();
12
13      // 3. Configurar TIM3
14      TIM3->PSC = 100 - 1; // (4MHz / 100 = 40kHz)
15      TIM3->ARR = (TIM_PCLK_FREQ_HZ / 100 / pwm_freq_hz) - 1; // 40kHz / pwm_freq_hz
16
17      // Configurar el Canal 1 (CH1) en modo PWM 1
18      TIM3->CCMR1 = (6U << 4);                    // PWM mode 1 on CH1
19      TIM3->CCER  |= (1 << 0);                    // Enable CH1 output
20
21      // Finalmente, habilitar el contador del timer (CEN bit en CR1)
22      TIM3->CR1 |= 0x01 << 0;
23  }
```

1. **Configuración del pin GPIO:** PA6 se configura como función alternativa (AF2) para TIM3_CH1.
   Esto conecta físicamente el pin al periférico TIM3.
2. **Habilitación del reloj:** El reloj del periférico debe estar habilitado antes de cualquier configuración.
3. **Configuración del prescaler (PSC):** Divide la frecuencia del reloj (4MHz) por 100 → 40kHz. Se escribe PSC-1 porque el contador cuenta desde 0.
4. **Configuración del auto-reload (ARR):** Define el periodo del PWM.
   ARR = (Frecuencia_timer / pwm_freq_hz) - 1

Ejemplo para 1kHz: (40kHz / 1kHz) - 1 = 39

5. **Configuración del modo PWM: CCMR1 = (6U << 4),** configura el canal 1 en modo PWM1. **CCER |= (1 << 0),** habilita la salida del canal 1.
6. **Habilitación del timer**: Bit CEN (Counter Enable) en CR1 inicia el contador.

Tim3 ch1 pwm set duty cycle

```
26  void tim3_ch1_pwm_set_duty_cycle(uint8_t duty_cycle_percent)
27  {
28      if (duty_cycle_percent > 100) {
29          duty_cycle_percent = 100;
30      }
31
32      // Calcular el valor de CCR1 basado en el porcentaje y el valor de ARR guardado
33      // CCR1 = ( (ARR + 1) * DutyCycle_Percent ) / 100
34      // Cuidado con el orden de operaciones para evitar truncamiento prematuro.
35      uint16_t tim3_ch1_arr_value = TIM3->ARR;
36      uint32_t ccr_value = (((uint32_t)tim3_ch1_arr_value + 1U) * duty_cycle_percent) / 100U;
37
38      TIM3->CCR1 = ccr_value;
39  }
```

Aquí asegura que el duty cycle no exceda el 100%

Calcula el valor de CCR1 basado en el porcentaje:

- **CCR1 = (ARR + 1) * duty_cycle_percent / 100**
- El +1 es necesario porque el contador va de 0 a ARR (inclusive)

Escribe el valor calculado en CCR1 (Compare Register 1)

Apartado Nvic

Iniciando con la estructura de los registros de nvic.h

UNIVERSIDAD NACIONAL DE COLOMBIA
SEDE MANIZALES

SYSCFG TypeDef

```
5   typedef struct {
6       volatile uint32_t MEMRMP;
7       volatile uint32_t CFGR1;
8       volatile uint32_t EXTICR[4];
9       volatile uint32_t SCSR;
10      volatile uint32_t CFGR2;
11      volatile uint32_t SWPR;
12      volatile uint32_t SKR;
13  } SYSCFG_TypeDef;
```

Aquí, controla la configuración del sistema, especialmente importante para mapear EXTI a pines GPIO.

EXTI TypeDef

```
15  typedef struct {
16      volatile uint32_t IMR1;
17      volatile uint32_t EMR1;
18      volatile uint32_t RTSR1;
19      volatile uint32_t FTSR1;
20      volatile uint32_t SWIER1;
21      volatile uint32_t PR1;
22      uint32_t RESERVED1[2];
23      volatile uint32_t IMR2;
24      volatile uint32_t EMR2;
25      volatile uint32_t RTSR2;
26      volatile uint32_t FTSR2;
27      volatile uint32_t SWIER2;
28      volatile uint32_t PR2;
29  } EXTI_TypeDef;
```

Controla las líneas de interrupción externas (EXTI).

UNIVERSIDAD NACIONAL DE COLOMBIA
SEDE MANIZALES

NVIC Type

```
32  typedef struct {
33      volatile uint32_t ISER[8U];         /*!< Offset: 0x000 (R/W)  Interrupt Set Enable Register */
34      uint32_t RESERVED0[24U];
35      volatile uint32_t ICER[8U];         /*!< Offset: 0x080 (R/W)  Interrupt Clear Enable Register */
36      uint32_t RESERVED1[24U];
37      volatile uint32_t ISPR[8U];         /*!< Offset: 0x100 (R/W)  Interrupt Set Pending Register */
38      uint32_t RESERVED2[24U];
39      volatile uint32_t ICPR[8U];         /*!< Offset: 0x180 (R/W)  Interrupt Clear Pending Register */
40      uint32_t RESERVED3[24U];
41      volatile uint32_t IABR[8U];         /*!< Offset: 0x200 (R/W)  Interrupt Active bit Register */
42      uint32_t RESERVED4[56U];
43      volatile uint8_t  IP[240U];         /*!< Offset: 0x300 (R/W)  Interrupt Priority Register (8Bit wide) */
44      uint32_t RESERVED5[644U];
45      volatile uint32_t STIR;             /*!< Offset: 0xE00 ( /W)  Software Trigger Interrupt Register */
46  } NVIC_Type;
```

Esto, controla el núcleo NVIC del procesador Cortex-M.

Mapeo de Memoria

```
48  #define SYSCFG_BASE         (0x40010000UL)
49  #define EXTI_BASE           (0x40010400UL)
50  #define NVIC_BASE           (0xE000E100UL)
51
52  #define SYSCFG              ((SYSCFG_TypeDef *) SYSCFG_BASE)
53  #define EXTI                ((EXTI_TypeDef *)   EXTI_BASE)
54  #define NVIC                ((NVIC_Type *)      NVIC_BASE)
```

Define las direcciones base de los periféricos y crea punteros a estructuras.

UNIVERSIDAD NACIONAL DE COLOMBIA
SEDE MANIZALES

Enumeración de Interrupciones (IRQnType)

```
59  typedef enum {
60      /* STM32L4xxxx Specific Interrupt Numbers */
61      WWDG_IRQn                   = 0,      /*!< Window WatchDog Interrupt                         */
62      PVD_PVM_IRQn                = 1,      /*!< PVD/PVM through EXTI Line detection Interrupt     */
63      TAMP_STAMP_IRQn             = 2,      /*!< Tamper and TimeStamp interrupts through the EXTI */
64      RTC_WKUP_IRQn               = 3,      /*!< RTC Wakeup interrupt through the EXTI line        */
65      FLASH_IRQn                  = 4,      /*!< FLASH global Interrupt                            */
66      RCC_IRQn                    = 5,      /*!< RCC global Interrupt                              */
67      EXTI0_IRQn                  = 6,      /*!< EXTI Line0 Interrupt                              */
68      EXTI1_IRQn                  = 7,      /*!< EXTI Line1 Interrupt                              */
69      EXTI2_IRQn                  = 8,      /*!< EXTI Line2 Interrupt                              */
70      EXTI3_IRQn                  = 9,      /*!< EXTI Line3 Interrupt                              */
71      EXTI4_IRQn                  = 10,     /*!< EXTI Line4 Interrupt                              */
72      DMA1_Channel1_IRQn          = 11,     /*!< DMA1 Channel 1 global Interrupt                   */
73      DMA1_Channel2_IRQn          = 12,     /*!< DMA1 Channel 2 global Interrupt                   */
74      DMA1_Channel3_IRQn          = 13,     /*!< DMA1 Channel 3 global Interrupt                   */
75      DMA1_Channel4_IRQn          = 14,     /*!< DMA1 Channel 4 global Interrupt                   */
76      DMA1_Channel5_IRQn          = 15,     /*!< DMA1 Channel 5 global Interrupt                   */
77      DMA1_Channel6_IRQn          = 16,     /*!< DMA1 Channel 6 global Interrupt                   */
78      DMA1_Channel7_IRQn          = 17,     /*!< DMA1 Channel 7 global Interrupt                   */
79      ADC1_2_IRQn                 = 18,     /*!< ADC1, ADC2 global Interrupts                      */
80      CAN1_TX_IRQn                = 19,     /*!< CAN1 TX Interrupt                                 */
81      CAN1_RX0_IRQn               = 20,     /*!< CAN1 RX0 Interrupt                                */
82      CAN1_RX1_IRQn               = 21,     /*!< CAN1 RX1 Interrupt                                */
83      CAN1_SCE_IRQn               = 22,     /*!< CAN1 SCE Interrupt                                */
84      EXTI9_5_IRQn                = 23,     /*!< External Line[9:5] Interrupts                     */
85      TIM1_BRK_TIM15_IRQn         = 24,     /*!< TIM1 Break interrupt and TIM15 global interrupt   */
86      TIM1_UP_TIM16_IRQn          = 25,     /*!< TIM1 Update Interrupt and TIM16 global interrupt  */
87      TIM1_TRG_COM_TIM17_IRQn     = 26,     /*!< TIM1 Trigger,Commutation Interrupt, TIM17 global  */
88      TIM1_CC_IRQn                = 27,     /*!< TIM1 Capture Compare Interrupt                    */
89      TIM2_IRQn                   = 28,     /*!< TIM2 global Interrupt                             */
90      TIM3_IRQn                   = 29,     /*!< TIM3 global Interrupt                             */
91      TIM4_IRQn                   = 30,     /*!< TIM4 global Interrupt                             */
92      I2C1_EV_IRQn                = 31,     /*!< I2C1 Event Interrupt                              */
93      I2C1_ER_IRQn                = 32,     /*!< I2C1 Error Interrupt                              */
```

```
94      I2C2_EV_IRQn                = 33,     /*!< I2C2 Event Interrupt                                */
95      I2C2_ER_IRQn                = 34,     /*!< I2C2 Error Interrupt                                */
96      SPI1_IRQn                   = 35,     /*!< SPI1 global Interrupt                               */
97      SPI2_IRQn                   = 36,     /*!< SPI2 global Interrupt                               */
98      USART1_IRQn                 = 37,     /*!< USART1 global Interrupt                             */
99      USART2_IRQn                 = 38,     /*!< USART2 global Interrupt                             */
100     USART3_IRQn                 = 39,     /*!< USART3 global Interrupt                             */
101     EXTI15_10_IRQn              = 40,     /*!< External Line[15:10] Interrupts                     */
102     RTC_Alarm_IRQn              = 41,     /*!< RTC Alarm (A and B) through EXTI Line Interrupt     */
103     DFSDM1_FLT0_IRQn            = 42,     /*!< DFSDM1 Filter 0 global Interrupt                    */
104     TIM8_BRK_IRQn               = 43,     /*!< TIM8 Break Interrupt                                */
105     TIM8_UP_IRQn                = 44,     /*!< TIM8 Update Interrupt                               */
106     TIM8_TRG_COM_IRQn           = 45,     /*!< TIM8 Trigger and Commutation Interrupt              */
107     TIM8_CC_IRQn                = 46,     /*!< TIM8 Capture Compare Interrupt                      */
108     ADC3_IRQn                   = 47,     /*!< ADC3 global Interrupt                               */
109     FMC_IRQn                    = 48,     /*!< FMC global Interrupt                                */
110     SDMMC1_IRQn                 = 49,     /*!< SDMMC1 global Interrupt                             */
111     TIM5_IRQn                   = 50,     /*!< TIM5 global Interrupt                               */
112     SPI3_IRQn                   = 51,     /*!< SPI3 global Interrupt                               */
113     UART4_IRQn                  = 52,     /*!< UART4 global Interrupt                              */
114     UART5_IRQn                  = 53,     /*!< UART5 global Interrupt                              */
115     TIM6_DAC_IRQn               = 54,     /*!< TIM6 global and DAC1&2 underrun error  interrupts*/
116     TIM7_IRQn                   = 55,     /*!< TIM7 global interrupt                               */
117     DMA2_Channel1_IRQn          = 56,     /*!< DMA2 Channel 1 global Interrupt                     */
118     DMA2_Channel2_IRQn          = 57,     /*!< DMA2 Channel 2 global Interrupt                     */
119     DMA2_Channel3_IRQn          = 58,     /*!< DMA2 Channel 3 global Interrupt                     */
120     DMA2_Channel4_IRQn          = 59,     /*!< DMA2 Channel 4 global Interrupt                     */
121     DMA2_Channel5_IRQn          = 60,     /*!< DMA2 Channel 5 global Interrupt                     */
122     DFSDM1_FLT1_IRQn            = 61,     /*!< DFSDM1 Filter 1 global Interrupt                    */
123     DFSDM1_FLT2_IRQn            = 62,     /*!< DFSDM1 Filter 2 global Interrupt                    */
124     DFSDM1_FLT3_IRQn            = 63,     /*!< DFSDM1 Filter 3 global Interrupt                    */
125     COMP_IRQn                   = 64,     /*!< COMP1 and COMP2 Interrupts                          */
126     LPTIM1_IRQn                 = 65,     /*!< LP TIM1 interrupt                                   */
127     LPTIM2_IRQn                 = 66,     /*!< LP TIM2 interrupt                                   */
```

```
128     OTG_FS_IRQn                 = 67,     /*!< USB OTG FS global Interrupt                         */
129     DMA2_Channel6_IRQn          = 68,     /*!< DMA2 Channel 6 global Interrupt                     */
130     DMA2_Channel7_IRQn          = 69,     /*!< DMA2 Channel 7 global Interrupt                     */
131     LPUART1_IRQn                = 70,     /*!< LP UART1 interrupt                                  */
132     QUADSPI_IRQn                = 71,     /*!< Quad SPI global interrupt                           */
133     I2C3_EV_IRQn                = 72,     /*!< I2C3 event interrupt                                */
134     I2C3_ER_IRQn                = 73,     /*!< I2C3 error interrupt                                */
135     SAI1_IRQn                   = 74,     /*!< Serial Audio Interface 1 global interrupt           */
136     SAI2_IRQn                   = 75,     /*!< Serial Audio Interface 2 global interrupt           */
137     SWPMI1_IRQn                 = 76,     /*!< SWPMI1 global interrupt                             */
138     TSC_IRQn                    = 77,     /*!< Touch Sense Controller global interrupt             */
139     LCD_IRQn                    = 78,     /*!< LCD global interrupt                                */
140     AES_IRQn                    = 79,     /*!< AES global interrupt                                */
141     RNG_IRQn                    = 80,     /*!< RNG global interrupt                                */
142     FPU_IRQn                    = 81      /*!< FPU global interrupt                                */
143 } IRQn_Type;
```

Aquí, se enumera todas las posibles interrupciones del microcontrolador.

UNIVERSIDAD NACIONAL DE COLOMBIA
SEDE MANIZALES

Prototipos de Funciones

```
146 // Prototipos de funciones
147 void nvic_exti_pc13_button_enable(void); // Configura EXTI13 y habilita su IRQ en NVIC
148 void nvic_usart2_irq_enable(void);       // Habilita USART2 IRQ en NVIC (la config. de USART2 se hace en uart.c)
```

Configura interrupciones para el botón en PC13 y para USART2.

Mirando las estructuras del registro nvic.c

Nvic enable irq (Función estática)

```
6  static void nvic_enable_irq(uint32_t IRQn)
7  {
8      NVIC->ISER[IRQn / 32U] |= (1UL << (IRQn % 32U));
9  }
```

Habilita una interrupción en el NVIC.

ISER es un array de registros donde cada bit controla una interrupción.

Divide el número de interrupción por 32 para seleccionar el registro correcto.

Usa el módulo 32 para seleccionar el bit específico.

Nvic exti pc13 button enable

```
11 void nvic_exti_pc13_button_enable(void) {
12     // 1. Habilitar el reloj para SYSCFG
13     rcc_syscfg_clock_enable(); // SYSCFG es necesario para mapear EXTI a GPIO
14
15     // 2. Configurar la línea EXTI13 (SYSCFG_EXTICR)
16     SYSCFG->EXTICR[3] &= ~(0x000FU << 4);  // Limpiar campo EXTI13 (bits 7-4)
17     SYSCFG->EXTICR[3] |=  (0x0002U << 4);  // Conectar EXTI13 a PC13 (0b0010 para PCx)
18
19     // 3. Configurar la línea EXTI13 para interrupción (EXTI_IMR1)
20     EXTI->IMR1 |= (1U << 13);
21
22     // 5. Configurar el trigger de flanco de bajada (EXTI_FTSR1)
23     EXTI->FTSR1 |= (1U << 13); // Habilitar trigger de flanco de bajada para línea 13
24     EXTI->RTSR1 &= ~(1U << 13); // Deshabilitar (no necesitamos detectar el flanco de subida)
25
26     // 6. Habilitar la interrupción EXTI15_10 en el NVIC
27     nvic_enable_irq(EXTI15_10_IRQn);
28 }
```

Habilita el reloj de SYSCFG (necesario para configurar EXTI)

Configura EXTI13 para que use PC13:

- **EXTICR[3]** controla EXTI12-15.
- Bits 4-7 controlan EXTI13 (0x2 = PC13)

Habilita la línea EXTI13 en el registro de máscara (IMR1)

Configura para detectar flanco de bajada (FTSR1) y deshabilita flanco de subida (RTSR1)

Habilita la interrupción en el NVIC (EXTI15_10 maneja líneas 10-15)

Nvic usart2 irq enable

```
30  void nvic_usart2_irq_enable(void) {
31      // Habilitar interrupción de recepción (RXNEIE - Read Data Register Not Empty Interrupt Enable)
32      // Esto hará que se genere una interrupción cuando RDR tenga un dato.
33      USART2->CR1 |= 0x01 << 5;
34      nvic_enable_irq(USART2_IRQn);
35  }
```

Habilita la interrupción por recepción en USART2 (RXNEIE)

Habilita la interrupción USART2 en el NVIC.

Apartado room_control

Iniciando con la estructura de los registros de room_control.h

```c
void room_control_on_button_press(void);

/**
 * @brief Función a ser llamada por USART2_IRQHandler cuando se recibe un carácter.
 * @param received_char El carácter recibido por UART.
 */
void room_control_on_uart_receive(char received_char);

/**
 * @brief (Opcional) Función para realizar inicializaciones específicas de la lógica
 *        de room_control, si las hubiera (ej. inicializar variables de estado).
 *        Las inicializaciones de periféricos se harán en main().
 */
void room_control_app_init(void);
```

Estas funciones definen la interfaz pública del módulo:

- Manejo de eventos del botón
- Procesamiento de datos recibidos por UART
- Inicialización de la aplicación

Mirando las estructuras del registro room_control.c

Variables de Estado

```c
// Variables de estado (static para mantener privacidad)
static volatile uint32_t led_onoff_start_time = 0;
static volatile uint8_t led_onoff_active = 0;
static volatile uint32_t last_button_press_time = 0;
```

led_onoff_start_time, marca de tiempo cuando se encendió el LED.

led_onoff_active, flag que indica si el LED está activo.

last_button_press_time, última vez que se presionó el botón (para anti-rebote)

room control app init

```c
void room_control_app_init(void)
{
    // Asegurar que los LEDs estén apagados al inicio
    gpio_write_pin(GPIOA, 7, GPIO_PIN_RESET); // LED ON/OFF apagado
    tim3_ch1_pwm_set_duty_cycle(70); // PWM al 70%
}
```

Establece el estado inicial de los LEDs.

Configura el LED PWM con un ciclo de trabajo inicial del 70%.

Room control on button press

```c
void room_control_on_button_press(void)
{
    uint32_t current_time = systick_get_tick();

    // Anti-rebote: ignorar pulsaciones muy cercanas (200ms)
    if ((current_time - last_button_press_time) < 200) {
        return;
    }

    last_button_press_time = current_time;

    // Encender LED ON/OFF
    gpio_write_pin(GPIOA, 7, GPIO_PIN_SET);
    led_onoff_start_time = current_time;
    led_onoff_active = 1;

    uart2_send_string("Boton B1: Presionado. LED encendido por 3 segundos.\r\n");
}
```

- Verifica el anti-rebote (ignora pulsaciones <200ms)
- Actualiza el tiempo del último botón presionado.
- Enciende el LED ON/OFF (PA7)
- Guarda el tiempo de inicio.
- Activa el flag de LED activo.
- Envía mensaje por UART.

UNIVERSIDAD NACIONAL DE COLOMBIA SEDE MANIZALES

Room control on uart receive

```c
void room_control_on_uart_receive(char received_char)
{
    // Eco del carácter recibido (ya se hace en USART2_IRQHandler)

    // Procesar comandos
    switch(received_char) {
        case 'h':
        case 'H':
            tim3_ch1_pwm_set_duty_cycle(100); // 100% duty cycle
            uart2_send_string("PWM al 100%\r\n");
            break;

        case 'l':
        case 'L':
            tim3_ch1_pwm_set_duty_cycle(0); // 0% duty cycle (apagado)
            uart2_send_string("PWM al 0%\r\n");
            break;

        case 't':
            gpio_toggle_pin(GPIOA, 7); // Toggle LED ON/OFF
            uart2_send_string("Toggle LED ON/OFF\r\n");
            break;

        default:
            // Opcional: enviar mensaje de comando no reconocido
            break;
    }
}
```

Estos son los comandos UART:

- **h/H**, máximo brillo PWM (100%)
- **l/L**, apaga PWM (0%)
- **t**, alterna estado del LED ON/OFF

UNIVERSIDAD NACIONAL DE COLOMBIA
SEDE MANIZALES

Room control process

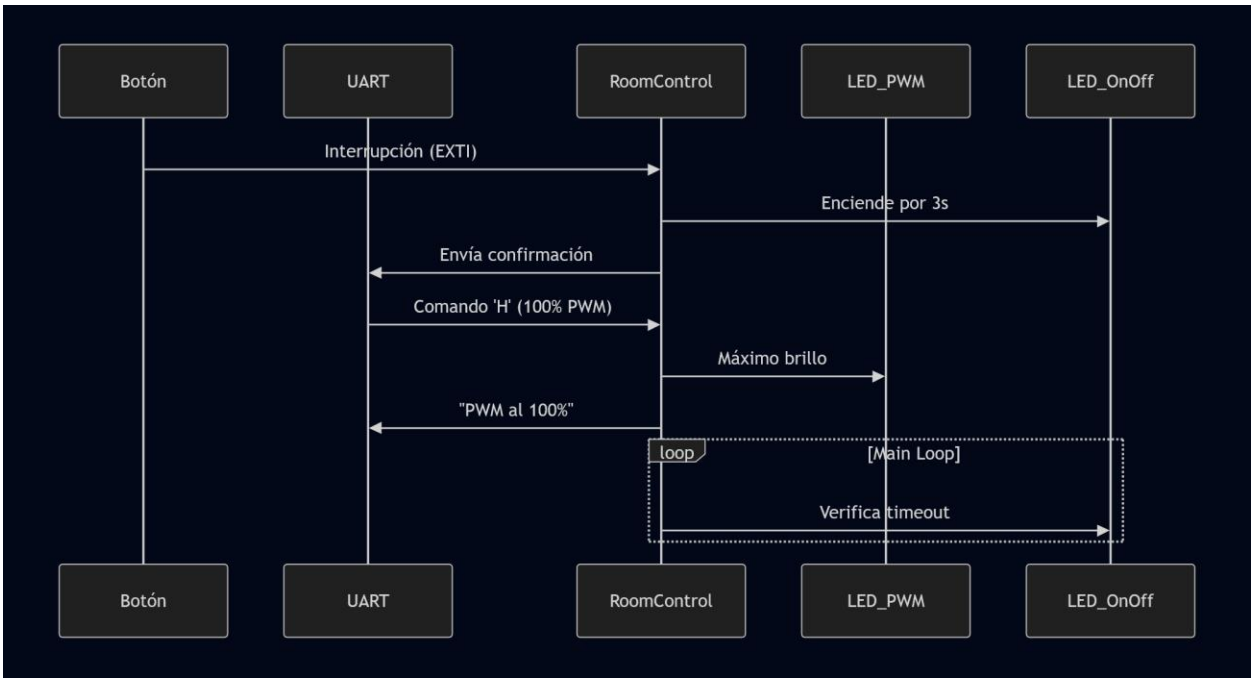
```c
void room_control_process(void)
{
    // Esta función debe ser llamada periódicamente (ej. desde el main loop)
    // para manejar tareas temporizadas

    if (led_onoff_active && (systick_get_tick() - led_onoff_start_time >= 3000)) {
        gpio_write_pin(GPIOA, 7, GPIO_PIN_RESET);
        led_onoff_active = 0;
        uart2_send_string("LED apagado después de 3 segundos.\r\n");
    }
}
```

Debe llamarse periódicamente desde el main loop.

Apaga el LED después de 3 segundos si está activo.

Envía confirmación por UART.

Interacción entre Componentes

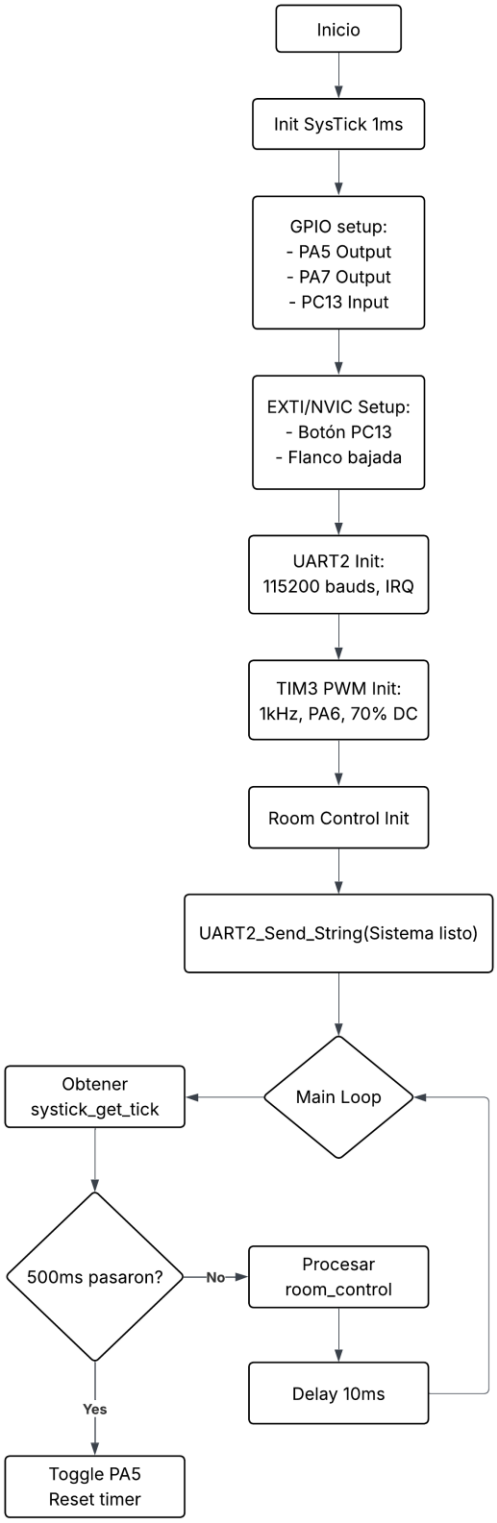


El sistema llega a interactuar con diversos periféricos, como el Botón (a través de una interrupción EXTI), el UART (para enviar confirmaciones) y módulos de control de iluminación (LED_PWM y LED_OnOff).

Santiago Galeano Castaño

UNIVERSIDAD NACIONAL DE COLOMBIA
SEDE MANIZALES

Diagrama del Main



Santiago Galeano Castaño

UNIVERSIDAD NACIONAL DE COLOMBIA
SEDE MANIZALES

Puntos Clave del Funcionamiento

El **Botón** genera una interrupción, encendiendo el led por 3 segundos.

El **UART** recibe los comandos **H**=100% PWM, **L**=0% PWM, **T**=toggle LED

LED Heartbeat (PA5), parpadeo automático cada 500ms

LED PWM (PA6), brillo controlado por TIM3 (inicial 70%)

LED ON/OFF (PA7), control manual por botón/commandos

SysTick es para retardos no bloqueantes, siendo este un mecanismo de temporización.

Gestión de tiempos en room_control_process()

Funcionamiento Principal

1. **Cuando se presiona el botón:**

- EXTI detecta el flanco de bajada, alias Interrupción.
- Enciende LED ON/OFF y activa el temporizador de 3s.
- Envía el mensaje UART, Botón presionado.

2. **Ahora el comando UART H:**

- Cambia el PWM a 100% de brillo
- Responde, PWM al 100%

3. **Hay un timeout de 3s**:

- Apaga automáticamente el LED ON/OFF
- Notifica por UART, LED apagado

Mientras que el otro LED:

1. **Parpadeo del LED:**

- El led parpadea automáticamente cada 500ms.
- Generando una simulación de pulsaciones.

Santiago Galeano Castaño

UNIVERSIDAD NACIONAL DE COLOMBIA
SEDE MANIZALES

Una manera de entender el flujo de todo:

El **corazón** (main.c) late rítmicamente gracias a SysTick, generando pulsos cada 500ms que hacen parpadear el LED heartbeat (PA5). Mientras tanto, sus **sentidos** están siempre alerta:

1. Cuando el dedo humano presiona el botón (PC13), es como un pinchazo que activa una **reacción refleja**:
    - La interrupción EXTI/NVIC dispara al instante, como un sistema nervioso
    - Enciende el LED ON/OFF (PA7) y activa un temporizador de 3 segundos en el cerebro (room_control)
    - Simultáneamente, la boca (UART) grita: ¡Me tocaron!
2. Si alguien susurra comandos por UART:
    - El oído serial (USART2) capta cada letra y desencadena acciones:
        - Con un H, los músculos PWM (TIM3) se tensan al máximo (100% duty cycle)
        - Un L los relaja completamente (0%)
        - Un T hace que el LED ON/OFF parpadee como un pestañeo
3. En segundo plano, el **sistema digestivo** (room_control_process) trabaja silenciosamente:
    - Verifica constantemente si los 3 segundos del temporizador han pasado
    - Cuando termina, apaga el LED ON/OFF y emite un mensaje UART de confirmación

Todo esto ocurre mientras el **metabolismo** (main loop) mantiene un ritmo constante:

- Cada 10ms respira (delay) para no agotar energía
- Alterna entre revisar el estado interno y reaccionar a estímulos externos