PRACTICA

Universitat Oberta de Catalunya

- **Asignatura**:
  Sistemas Empotrados
- **Semestre**:
  2024/25
- **Nombre Estudiante/a**:
  CHRISTIAN MOYA ROYO

Contenido

Actividad 1 [30%] Volumen Táctil

En este apartado, se ha implementado la función TocandoTarea() cuya finalidad es procesar la información recibida desde el sensor táctil y actualizar tanto el estado de los LEDs como el factor de volumen.

```
void TocandoTarea(){
      if (TSL_USER_STATUS_BUSY != MX_TOUCHSENSING_Task()) {
                  if (TSL_STATEID_DETECT == MX_TOUCHSENSING_GetStatus())
{
                        position = MX_TOUCHSENSING_GetPosition() /
MAX_SENSIBILITY;
                        int volumen = MX_TOUCHSENSING_GetPosition() /
MAX_VOLUMEN;
                        int multiplicador = 1 << volumen; // Usamos el
desplazamiento de bits para 2^volumen
                  volumeFactor = multiplicador;
                  // Actualizamos los LEDs según la posición detectada
                  LEDClear();

                        for (int i = 0; i <= position; i++) {
                              LEDTurnOn(i);
                        }
                  }
            }
      }
```

La función hace uso de dos constantes importantes para el procesamiento de datos:

- MAX_SENSIBILITY: Esta constante se utiliza para calcular cuántos bits corresponden a cada LED. El sensor táctil devuelve valores en un rango de [0, 128]. Dividiendo este rango entre 6 (el número de LEDs), obtenemos 21.3 por LED, pero se redondea a 21. Esto permite calcular el número de LEDs que deben encenderse en función de la posición detectada.

- MAX_VOLUMEN: Sirve para dividir el rango del sensor en grupos de 5 niveles de volumen (ya que la barra tiene cinco zonas). Se emplean valores escalonados en potencias de 2: {1, 2, 4, 8, 16}. Esto se consigue mediante desplazamiento de bits (1 << volumen), que es una forma de calcular potencias de 2. Es decir;

    - 1 << 0 → Desplaza el bit '1' 0 posiciones a la izquierda → 1 (2^0)
    - 1 << 1 → Desplaza el bit '1' 1 posición a la izquierda → 2 (2^1)
    - 1 << 2 → Desplaza el bit '1' 2 posiciones a la izquierda → 4 (2^2)
    - 1 << 3 → Desplaza el bit '1' 3 posiciones a la izquierda → 8 (2^3)
    - 1 << 4 → Desplaza el bit '1' 4 posiciones a la izquierda → 16 (2^4)

Nota añadida en este apartado (derivada de la actividad 2):

El valor de **MAX_VOLUMEN** determina el número de niveles de volumen generados a partir de los datos del sensor táctil. Para evitar distorsiones en el audio, se ha optado por un valor ligeramente superior a 25 e inferior a 32, como 26. Este valor permite dividir el rango del sensor táctil (**0-128**) en 5 niveles, asignando aproximadamente 26 unidades por nivel, y garantiza que los multiplicadores calculados mediante potencias de 2 (**{1, 2, 4, 8, 16}**) no excedan los límites aceptables para el hardware de audio. Al evitar multiplicadores mayores, como **32 o 64**, se preserva la calidad del sonido sin saturación. Además, se ha creado una variable global, **volumeFactor**, que almacena el factor por el cual debe incrementarse el volumen, permitiendo un control más preciso del nivel de amplificación.

El control de los LEDs se gestiona mediante las siguientes funciones:

- LEDClear(): Su función es apagar todos los LEDs antes de proceder a encender los necesarios. Esto asegura que el estado anterior no afecte a la nueva configuración.

```
void LEDClear() {
      for (int i = 0; i < LED_MAX; i++) {
            LEDTurnOff(i);
      }
}
```

- LEDTurnOn(): Hemos aprovechado al función ya existente para después de borrar el estado previo, se recorra un bucle desde el índice 0 hasta la posición calculada. Cada LED dentro de este rango se enciende secuencialmente, proporcionando una representación visual de la posición detectada en el sensor táctil.

```
void LEDTurnOn(bar_led led)
{
      HAL_GPIO_WritePin(gpio_led_bar[led].pPort,
gpio_led_bar[led].Pin, GPIO_PIN_RESET);
}
```

La función TocandoTarea() se invoca dentro de AudioTask() como parte del manejo de eventos. El sistema verifica el estado de los flags generados por el joystick y, dependiendo del flag activado, ejecuta las acciones correspondientes, como actualizar LEDs o reproducir audio. Si no se detectan eventos específicos, se llama a TocandoTarea() para procesar la información táctil y actualizar los LEDs y el volumen.

Actividad 2 [30%] Barra de reproducción

He optado en esta actividad por el modo de implementación COMPLETO, para ello se ha ajustado el código existente en la función **AudioOutOneShot** para adaptarlo a los nuevos requerimientos del ejercicio. Dentro de la función se ha añadido un par de líneas en el bucle **for** con la condición de **(sample_num = 0; AUDIO_BUFFER_LEN > sample_num; sample_num += DMA_TRANSFER_LEN)**.

Con el propósito de calcular el valor máximo posible en los registros de audio, se ha analizado el registro de ejemplo. Con esto se ha podido ver que cada valor almacenado representa un número entero de **8 bits sin signo**, lo que implica un rango de valores entre **0 y 255**. Por este motivo, se ha decidido modificar **DMA_TRANSFER_LEN** a **1024**, que es el resultado de **8192 / 8**, para optimizar la transferencia de datos.

Partiendo de esta premisa y considerando que el factor óptimo de amplificación del audio (volumen) en todas las grabaciones es **x16**, se puede estimar que el rango de valores obtenidos estará entre **[0 × 16, 255 × 16] = [0, 4080]**. El multiplicador **16** se ha elegido porque permite escalar los valores de 8 bits (máximo 255) a un rango mayor sin perder resolución ni introducir distorsión en el sonido amplificado, manteniendo la precisión del audio en el procesamiento digital. Dado que el sistema cuenta con **6 LEDs**, se ha dividido el rango máximo **4080 / 6 = 680**, creando una constante llamada **MAX_AUDIO** para asignar correctamente los valores del sensor a cada LED.

```
 */
for (sample_num = 0; AUDIO_BUFFER_LEN > sample_num; sample_num +=
DMA_TRANSFER_LEN) {

    //LEDTurnOff(led_num--);
    LEDClear();
    /*
     * We initialize tha DAC with a block (at least) of 8192 samples
     */
    if (HAL_OK != MX_DAC1_Start(&u32DataBlock_Audio[sample_num],
    DMA_TRANSFER_LEN)) {
        return HAL_ERROR;
    }
    int ledsToActivate = (u32DataBlock_Audio[sample_num] * 1.9)
            / (MAX_AUDIO);
    for (int i = 0; i < ledsToActivate; i++) {
        LEDTurnOn(i);
    }

    /*
     * We wait for the completion reproduction using the Flag DAC_CONV_CPLT_FLAG
     */
    dac_flag = osEventFlagsWait(Dac1EventHandle, DAC_CONV_CPLT_FLAG,
    osFlagsWaitAny, osWaitForever);
    if (DAC_CONV_CPLT_FLAG != dac_flag) {
        return HAL_ERROR;
    }
```

El fragmento de código añadido entre la línea donde se define la variable **entera ledsToActivate** y la llamada a la función **LEDTurnOn(i)** tiene como objetivo **calcular y controlar la cantidad de LEDs que deben encenderse** en función del valor de la muestra de audio procesada. Se ha añadido (1.9) para que se encendieran de una forma más estética.

Actividad 3 [30%] Modo de reproducción

En la función AudioTask(), se ha modificado el caso JOYSTICK_DO_FLAG y se ha creado un búcle for. La modificación se realiza en el siguiente bloque de código:

```
case JOYSTICK_DO_FLAG:
                if (HAL_OK != AudioOutOneShot()) {
                      goto error;
                }
              for (; loopEnable == 1;) {
                      joystick_flag =
osEventFlagsGet(JoystickEventHandle);
                      if (joystick_flag == JOYSTICK_LE_FLAG) {
                          osEventFlagsClear(JoystickEventHandle,
JOYSTICK_LE_FLAG);
                      } else {
                          if (HAL_OK != AudioOutOneShot()) {
                              goto error;
                          }
                      }
                  }
                osEventFlagsClear(JoystickEventHandle,
JOYSTICK_DO_FLAG);
                break;
```

En cada iteración del bucle, se obtiene el valor de joystick_flag usando osEventFlagsGet(JoystickEventHandle), lo que permite detectar eventos del joystick. Si el joystick_flag es igual a JOYSTICK_LE_FLAG, se limpia la bandera correspondiente con osEventFlagsClear(). Si el joystick_flag no es JOYSTICK_LE_FLAG, se ejecuta la función AudioOutOneShot() nuevamente para continuar la reproducción de audio. Si hay un error en la función, el flujo de control va al bloque error con goto error. Para finalizar, se ha establecido loopEnable = 1 para la dirección derecha y loopEnable = 0 para la dirección izquierda.

```
void JoystickRiTask(void *argument) {
    /* Infinite loop */
    for (;;) {
        // Wait for Joystick Button Pulsed
        if (GPIO_PIN_SET == JoystickReadButton(BUTTON_RIGHT)) {
            // Wait 50ms for bouncing
            osDelay(50);
            loopEnable = 1; //Se hará un bucle
            // Check if Joystick Button is still pulsed
            if (GPIO_PIN_SET == JoystickReadButton(BUTTON_RIGHT)) {
                // Send Event
                osEventFlagsSet(JoystickEventHandle, JOYSTICK_RI_FLAG);
                // Wait for Joystick Button Release
                while (GPIO_PIN_SET == JoystickReadButton(BUTTON_RIGHT))
                    ;
                // Wait 50ms for bouncing
                osDelay(50);
            }
        } else {
            osDelay(1);
        }
    }
}

/**
  * @brief Function implementing the JoystickLeTask thread.
  * @param argument: Not used
  * @retval None
  */
void JoystickLeTask(void *argument) {
    /* Infinite loop */
    for (;;) {
        // Wait for Joystick Button Pulsed
        if (GPIO_PIN_SET == JoystickReadButton(BUTTON_LEFT)) {
            // Wait 50ms for bouncing
            osDelay(50);
            loopEnable = 0; //Se terminara el bucle
            // Check if Joystick Button is still pulsed
            if (GPIO_PIN_SET == JoystickReadButton(BUTTON_LEFT)) {
                // Send Event
                osEventFlagsSet(JoystickEventHandle, JOYSTICK_LE_FLAG);
                // Wait for Joystick Button Release
                while (GPIO_PIN_SET == JoystickReadButton(BUTTON_LEFT))
                    ;
                // Wait 50ms for bouncing
                osDelay(50);
            }
        } else {
            osDelay(1);
        }
    }
```

Actividad 4 [10%] Grabadora completa

Para este ejercicio se ha empleado como base la Solución de la PEC 3, adaptándola para cumplir con los requisitos de la práctica actual. Se ha reutilizado parte del código previo, realizando los ajustes necesarios para optimizar su funcionalidad y adaptarlo al nuevo contexto. Conforme al código inicial, tenemos que el primer cambio es el tamaño de la transferencia de datos de DMA_TRANSFER_LEN a DAC_TRANSFER_LEN, como en el ejercicio 2 he modificado el DMA he tenido que añadir el DAC con el valor original.

Se ha definido una nueva constante denominada **DAC_TRANSFER_LEN** que reemplaza a la constante **DMA_TRANSFER_LEN**, la cual había sido modificada previamente en el ejercicio 2. Esta nueva constante se define de la siguiente manera:

```
#define DMA_TRANSFER_LEN   1024 /*Dividimos el valor original en 8*/
#define DAC_TRANSFER_LEN   8192  /* Maximum value */
```

Además, he agregado un escalado de los datos de audio (AUDIO_ESCALA) después de la captura. El valor asignado a esta constante es 1/16, y se define de la siguiente manera:

```
/*Creamos una constante necesaria para el ej4*/
#define AUDIO_ESCALA 0.0625
```

Por otra parte, se ha implementado una variable global denominada hasRecording. Esta variable permite determinar si se debe reproducir la grabación de ejemplo o una grabación personalizada. Su declaración es la siguiente:

```
int hasRecording= 0;
```

La lógica de los LEDs también cambia: en vez de encender todos los LEDs al final, se apagan primero con LEDClear() y luego se encienden según la variable position.

Finalmente, se realizaron modificaciones en las funciones responsables de controlar la dirección del gatillo. En particular:

- Se ha asignado el valor 1 a hasRecording en la función correspondiente al gatillo en posición UP.

Enlace Drive:
https://drive.google.com/drive/folders/1lMF1V71jjUim9aYuIrR2yOyz7tXNDZ9Y?usp=drive_link

Bibliografía

Cama, J. M. (s.f.). *Sistemas operativos empotrados.* Barcelona: FUOC.

Gallego, F. V. (2022). *Libro de sistemas integrados.* Barcelona: FUOC.

Guillén, I. v. (s.f.). *Introducción a los sistemas empotrados.* Barcelona: FUOC.

López Vicario, J., & Morell Pérez, A. (s.f.). *El hardware: arquitectura y componentes.* Barcelona: FUOC.

Prades García, J. D., & Hernández Ramírez, F. (s.f.). *El software.* Barcelona: FUOC.