TRABAJO DE SISTEMAS ELECTRÓNICOS DIGITALES
CURSO 2022/23

APLICACIÓN DOMÓTICA CON EL MICROCONTROLADOR: CONTROL DE PERSIANAS

APELLIDOS, NOMBRES: ANTOLÍN PULIDO, LUCÍA (55125)
GÓMEZ BARAHONA, EVA (55267)
SEISDEDOS BARRIENTOS, MARTA (55468)

FECHA: 09/01/2023

GRUPO: LUNES (A404)

PROFESOR: LUIS CASTEDO

ÍNDICE

INTRODUCCIÓN

El desarrollo de este proyecto se basa en la realización de una aplicación domótica con un microcontrolador ARM, en nuestro caso, el control de persianas en modo manual y automático. La placa de desarrollo utilizada es la STM32F4 Discovery, en concreto, la STM32F411. Además de dicha placa, hemos visto oportuno añadir ciertos dispositivos adicionales necesarios para el correcto funcionamiento que se describen a continuación.

COMPONENTES

Como hemos mencionado anteriormente, hemos utilizado varios dispositivos para el correcto funcionamiento del control de persianas.

1. **STM32**: El dispositivo elemental es la placa de desarrollo STM32F411, microcontrolador de tipo ARM. Esta placa cuenta con una serie de puertos en los cuales se han ido conectando los dispositivos adicionales, y las entradas y salidas de nuestro proyecto.

    - Botones: utilizados como entradas. Hemos empleado el puerto PA0 como botón para el modo manual. Cada vez que pulsemos este botón (estando en modo manual), la persiana procederá a bajar o subir, en función de su posición y su acción.

    - Leds: utilizados como salidas. Hemos activado una serie de leds para indicar si la persiana está subiendo o bajando. En caso de estar subiendo, se encenderá el led verde que pertenece al pin 12, por el contrario, si la persiana está bajando, se encenderá el led azul perteneciente al puerto del pin 15. Si la persiana está parada, no se encenderá ningún led.

2. **Botón auxiliar**: debido a los escasos botones de la placa de desarrollo hemos visto necesario conectar un botón adicional al puerto PA2 para el funcionamiento automático de la persiana. Cuando se pulse este botón se pasará de modo manual a modo automático y viceversa.

3. **Sensor de luminosidad**: empleado para el funcionamiento automático. Este sensor se emplea como conversor analógico digital, y se encarga de medir la luz incidida sobre nuestra maqueta. En función del nivel de dicha luz, la persiana bajará o subirá.

4. **Otros elementos**: para la conexión de todos estos dispositivos, también hemos utilizado elementos como cables, resistencias, y una placa *protoboard*.

FUNCIONAMIENTO

Se han programado dos modos de funcionamiento principales para la persiana. Estos son modo manual y automático.

MODO MANUAL

Este modo será el que esté por defecto en el dispositivo. En un primer momento, la persiana se encontrará bajada totalmente. En el momento que se pulse el botón de la placa STM32F411 (asignado al pin GPIO_PIN_0) se producirá el movimiento de la persiana hacia arriba. Esto se podrá ver claramente indicado ya que se encenderá un LED verde (asignado al pin GPIO_PIN_12) hasta que esta se haya subido completamente. Se ha establecido un contador para garantizar que cuando la persiana se ha recogido completamente el motor pare. Si se vuelve a pulsar el mismo botón la persiana bajará, indicándose así con un LED azul (asignado al pin GPIO_PIN_15).

La implementación de un contador cuando la persiana está en movimiento permite que, si se pulsa el botón una vez la persiana está en movimiento, esta modifique su trayectoria y la invierta. Esto permite un funcionamiento rápido y eficiente del dispositivo.

MODO AUTOMÁTICO

En este segundo modo se implementarán fotorresistores, ya que se quiere conseguir que la persiana funcione atendiendo a la luz solar que hay en el exterior. Se ha programado de tal forma que, si existe luz solar (es de día) la persiana suba. De forma análoga, si ha oscurecido y no hay luz suficiente la persiana bajará. Para habilitar este método será necesario pulsar el botón auxiliar implementado en el montaje. Una vez este modo está activado el motor de la persiana se activará según la luz que reciba la fotorresistencia. De igual forma que en el método anterior, se puede ver el movimiento del motor a partir de los LEDs asignados para esto.

DESCRIPCIÓN CÓDIGO

```
/* USER CODE BEGIN PV */
    volatile int estado=0; //0 esta bajada, 1 esta subida, 2 en movimiento
    volatile int accion=0;//0 parado, 1 subiendo, 2 bajando
    volatile int position=1; //motor
    volatile int cuenta=0;
    volatile int aux=0;
    volatile int botonautomatic=0;
    __IO uint16_t luz=0;
/* USER CODE END PV */
```

Inicializamos todas las variables que vamos a utilizar durante el programa

```
/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_TIM2_Init(void);
static void MX_ADC1_Init(void);
/* USER CODE BEGIN PFP */
```

Inicializamos todas las funciones INIT necesarias

```
/* Private user code ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin){

    if (GPIO_Pin==GPIO_PIN_0){      //pulsacion de boton
        if(accion==0){              //persiana parada
            if(estado==0)           //persiana bajada
                accion=1;           //subiendo
            else if(estado==1)      //persiana subida
                accion=2;           //bajando
        }
        else if(accion==1)          //persiana subiendo
            accion=2;


        else if(accion==2)          //persiana bajando
            accion=1;
    }

    else if (GPIO_Pin==GPIO_PIN_2){//modo automatico
        if (botonautomatic==0) //activa modo
            botonautomatic=1;

        else if(botonautomatic==1) //desactiva modo
            botonautomatic=0;
    }
}
```

Hacemos uso de esta función *Callback* para generar una interrupción cada vez que presionemos el botón de la placa, en este caso conectado al pin **PA0**, para subir o bajar las persianas manualmente. En cambio, si accionamos el botón instalado en la *protoboard* (conectado con el pin **PA2**) se activará el modo automático.

Seguidamente, creamos la siguiente función:

```
int antirrebotes(volatile int* button_int, GPIO_TypeDef* GPIO_Port, uint16_t GPIO_number){

    uint8_t button_count=0;
    int counter=0;
    const uint8_t periodo_comprobacion=25; //ms

    if (HAL_GPIO_ReadPin(GPIO_Port, GPIO_number)){
        counter=HAL_GetTick();

        while (button_count<4){
            if ((HAL_GetTick()-counter) >= periodo_comprobacion){
                if (HAL_GPIO_ReadPin(GPIO_Port, GPIO_number)!=1){
                    button_count=0;
                }else{
                    button_count++;
                }
                if (button_count==4){
                    return 1;
                    button_count=0;
                }
            counter=HAL_GetTick();
            }
        }

    }
    button_count=0;
    return 0;

}
```

Implementamos la función antirrebotes con el objetivo de evitar el bloqueo del botón.

```c
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_TIM2_Init();
  MX_ADC1_Init();
  /* USER CODE BEGIN 2 */
  HAL_TIM_Base_Start_IT(&htim2);

  while (1)
  {
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
```

Configuración de todos los periféricos utilizados.
Después del *USER BEGIN 2,* definimos el comienzo del temporizador **TIM2.**

```c
    HAL_ADC_Start(&hadc1);
    if(HAL_ADC_PollForConversion(&hadc1,100000)==HAL_OK){
        luz=HAL_ADC_GetValue(&hadc1);
    }
```

Inicializamos el conversor ADC y asignamos el valor convertido de analógico a digital a la variable **luz**.

Justo a continuación, se inicia el modo automático:

```c
    if (antirrebotes(&botonautomatic,GPIOA,GPIO_PIN_2)){
        if(botonautomatic){
            if(luz>1500) //persiana se sube
                accion=2;

            else
                accion=1;

        }
    }


    if(cuenta>499 && accion==1){
        accion=0;
        estado=1; //persiana subida completamente

    }


    if(cuenta<1 && accion==2){
            accion=0;
            estado=0; //persiana bajada completamente
    }
```

Llamamos a la función antirrebotes.
Si se activa la variable **botonautomatic**, es decir, pulsamos el botón auxiliar de la *protoboard*, y, además, el valor de la variable **luz** es superior a 500, lo que quiere decir que la iluminación es escasa, **la persiana baja**, activándose los *flags* correspondientes.
Sin embargo, si el valor de la variable **luz** es inferior a 500, hay más iluminación de lo normal, **la persiana sube**, activándose los *flags* correspondientes.

```
    if(accion==1 && aux==1){ //subiendo
        do{
        estado=2; //en movimiento
        HAL_GPIO_WritePin(GPIOD, GPIO_PIN_12, GPIO_PIN_SET);//led VERDE activado
        HAL_GPIO_WritePin(GPIOD, GPIO_PIN_13, GPIO_PIN_RESET);
        HAL_GPIO_WritePin(GPIOD, GPIO_PIN_14, GPIO_PIN_RESET);
        HAL_GPIO_WritePin(GPIOD, GPIO_PIN_15, GPIO_PIN_RESET);

        position++;
        HAL_Delay(10);

        if(position>4){
            position=1;
            cuenta++;
        }

        setPosition(position);

        }while (cuenta<499 && (accion==1||aux==1)); //morgan

    }
```

En este caso, vamos a tratar el **modo manual.**
La persiana se encuentra en reposo, por lo que, si pulsamos el botón de la placa, se enciende el **led verde**, lo que significa que la persiana está **subiendo**.

Una vez se activa el estado subiendo, la variable **position** (variable de la configuración del motor, cuya función se llama dentro de este *do while* para que exista movimiento en nuestro motor) empieza a sumar, una vez sea **position** mayor que 4, incrementa la variable **cuenta**. Se repite este proceso hasta que **cuenta** sea mayor a 499.

```
    else if(accion==2 && aux==2){ //bajando
        do{
        estado=2; //en movimiento
        HAL_GPIO_WritePin(GPIOD, GPIO_PIN_12, GPIO_PIN_RESET);
        HAL_GPIO_WritePin(GPIOD, GPIO_PIN_15, GPIO_PIN_SET); //led azul activad
        HAL_GPIO_WritePin(GPIOD, GPIO_PIN_13, GPIO_PIN_RESET);
        HAL_GPIO_WritePin(GPIOD, GPIO_PIN_14, GPIO_PIN_RESET);


        position--;
        HAL_Delay(10);

        if(position< 1){
              position=4;
              cuenta--;
        }

        setPosition(position);

        } while (cuenta>1 && (accion==2||aux==2));
    }
    else{
        HAL_GPIO_WritePin(GPIOD, GPIO_PIN_12, GPIO_PIN_RESET);
        HAL_GPIO_WritePin(GPIOD, GPIO_PIN_13, GPIO_PIN_RESET);
        HAL_GPIO_WritePin(GPIOD, GPIO_PIN_14, GPIO_PIN_RESET);
        HAL_GPIO_WritePin(GPIOD, GPIO_PIN_15, GPIO_PIN_RESET);

    }
}
```

Si volvemos a pulsar el botón de la placa, se enciende el **led azul**, lo que quiere decir que la persiana está **bajando**.

En el caso de que no se pulse ningún botón, todos los leds se mantienen apagados.

Al igual, en el caso de bajada, una vez se activa la variable **position**, el motor se empieza a mover, pero en este caso, se mueve en sentido contrario. Ahora, se repite el proceso hasta que **cuenta** sea menor que 1.

```c
/* USER CODE BEGIN 4 */
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim){

    if(htim->Instance==TIM2){

        if(accion==1)
            aux=1;

        else if(accion==2)
            aux=2;


    }

}
```

Esta función *Callback* viene originada por la definición del comienzo del temporizador **TIM2**. Dentro de esta, señalamos que si el valor de la variable **acción** es 1 (subiendo), el valor de nuestra variable auxiliar **aux** va a ser 1 también. En cambio, si **acción** es 2, **aux** también va a ser 2.
La variable **aux** la tratamos como *flag*.

```c
void setPosition(int pos){

    switch(pos){
    case 1:
        HAL_GPIO_WritePin(GPIOA, GPIO_PIN_4|GPIO_PIN_5|GPIO_PIN_6, GPIO_PIN_RESET);
        HAL_GPIO_WritePin(GPIOA, GPIO_PIN_3, GPIO_PIN_SET);
    break;

    case 2:
        HAL_GPIO_WritePin(GPIOA, GPIO_PIN_3|GPIO_PIN_5|GPIO_PIN_6, GPIO_PIN_RESET);
        HAL_GPIO_WritePin(GPIOA, GPIO_PIN_4, GPIO_PIN_SET);
    break;

    case 3:
        HAL_GPIO_WritePin(GPIOA, GPIO_PIN_3|GPIO_PIN_4|GPIO_PIN_6, GPIO_PIN_RESET);
        HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, GPIO_PIN_SET);
    break;

    case 4:
        HAL_GPIO_WritePin(GPIOA, GPIO_PIN_3|GPIO_PIN_4|GPIO_PIN_5, GPIO_PIN_RESET);
        HAL_GPIO_WritePin(GPIOA, GPIO_PIN_6, GPIO_PIN_SET);
    break;

    }

}
```

Función que configura el movimiento del motor paso a paso.

Tanto en el modo manual como en el automático, la persiana sube y baja una vez se han cumplido el número de vueltas mínimo del motor, dicho valor lo guarda la variable **cuenta**. No obstante, este planteamiento del movimiento permite que, si la persiana está en proceso de subida, si volvemos a pulsar el botón activando la acción de bajada, el valor de la variable **cuenta** se almacena y, por tanto, la persiana procede a bajarse desde el punto en el que se había quedado mientras esta subía.

PROBLEMAS Y MEJORAS DEL CÓDIGO

Como se puede observar en el repositorio adjuntado, el proyecto cuenta con varias versiones donde se ha ido desarrollando el programa y corrigiendo diversos problemas que se hayan podido tener. Los más significativos son los siguientes:

En un primer momento, se inició el proyecto con la programación del modo manual del dispositivo. A la hora de implementar las primeras interrupciones no hubo problemas y el funcionamiento era correcto, pero tuvimos dificultades a la hora de implementar el motor encargado de mover la persiana. Aunque implementamos un temporizador que generaba una interrupción al finalizar su tiempo (tipo *BASE*), a la hora de establecer el código encargado de indicar los pasos del motor (ya que se ha usado un motor paso a paso), debido a los parámetros seleccionados para los pasos de este imposibilitaba que el motor se moviera correctamente, por lo que se tuvieron que hacer diferentes cuentas para adaptarlo al funcionamiento del programa.

Otro de los problemas que surgieron a lo largo del desarrollo del código se dio al tratar la variable cuenta. En un principio, esta variable se estableció para que, en caso de que el usuario

pulsara el botón mientras la ventana estuviera en movimiento, automáticamente cambiara su dirección. Esto pretendía soportar casos reales como podía ser que el usuario hubiese pulsado sin querer o, simplemente hubiera cambiado de opinión. Sin embargo, con lo programado en los primeros códigos, no conseguimos que la persiana parara el movimiento inicial tras una segunda pulsación. Esto se debía a que estábamos planteando de forma incorrecta las condiciones del *do-while* donde se modifica el valor de la posición y se encienden los LEDs necesarios. Tras varios intentos, gracias a las Leyes de Morgan conseguimos determinar una condición válida para esta nueva funcionalidad (se puede observar en la versión 3).

Finalmente, quisimos implementar un código antirrebotes para asegurar un comportamiento uniforme y fluido del código, haciendo hincapié en el modo manual (que fue el que más problemas de inestabilidad dio en las simulaciones del proyecto). Para poder implementarlo de una forma limpia, sin tener que modificar el código completo, se optó por generar una función de tipo *bool* que generara un periodo de comprobación y así poder eliminar las posibles inestabilidades o señales de ruido que puede haber a lo largo de las pulsaciones.

ENLACE AL REPOSITORIO DE GITHUB Y AL VIDEO DEMOSTRATIVO

A continuación, adjunto el link a partir del cual se puede acceder a toda la documentación recogida en nuestro repositorio de Github. También, se adjunta el enlace a una carpeta de Google Drive donde hemos guardado el video demostrativo del funcionamiento de la maqueta de nuestra persiana.

GITHUB:
https://github.com/evagombar/Trabajo-SED-micros

GOOGLE DRIVE:
https://drive.google.com/drive/folders/1mM8AiNvwjf8nL4DhT41M40025g54A7zY?usp=share_link