

– Lab 4: Pedestrian Detection Feature (Timers/WDT) –

Objetivo:

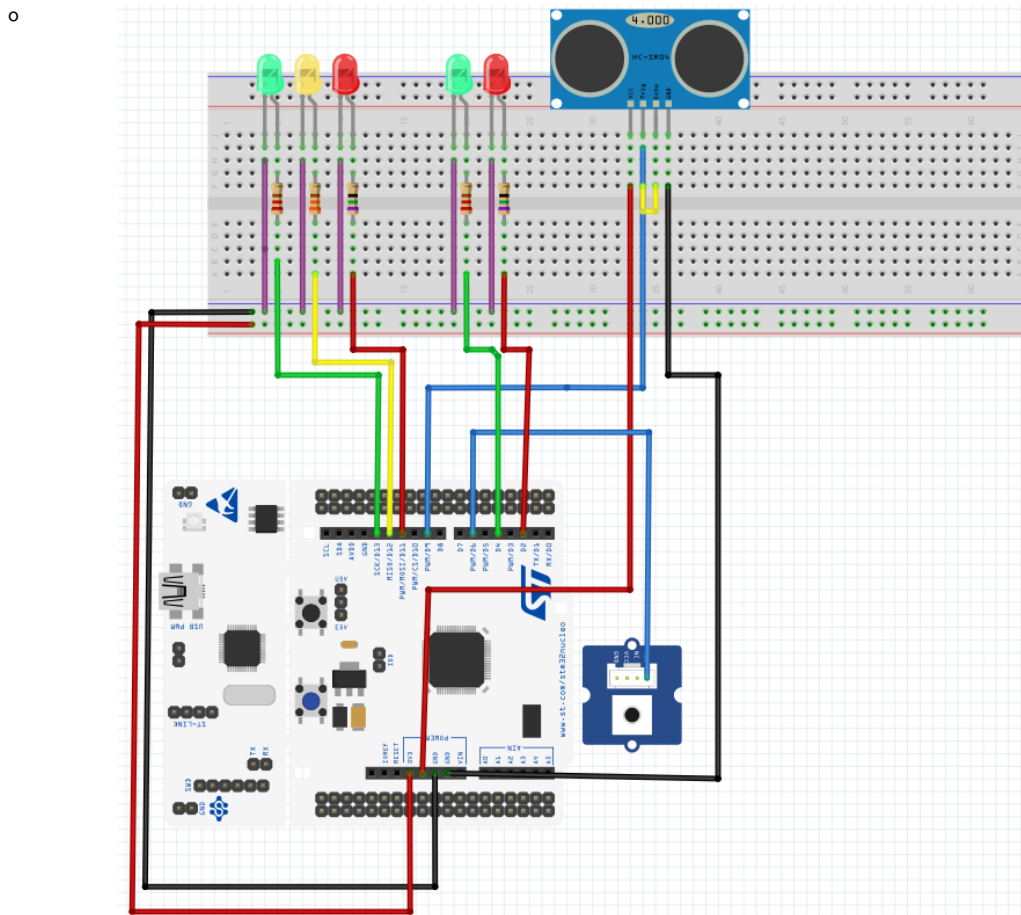
El objetivo de la práctica es el siguiente:

Debido a una pandemia y atendiendo a las recomendaciones sanitarias, nuestro cliente ha decidido añadir un sistema sin contacto, que permita realizar la misma operación que el botón. Para ello, se hará uso de un sensor de ultrasonidos y de un timer de la placa de desarrollo STM32-F411RE. Se considerará la presencia de un peatón si se detecta un obstáculo a menos de 200 mm.

Además, nuestro cliente ha decidido que no quiere interrupciones anidadas ni tampoco que una vez finalizado el ciclo del semáforo se inicie inmediatamente un nuevo ciclo, se deberá esperar al menos 5 segundos entre una petición y otra.

Esquema:

El esquema de nuestro circuito realizado en Fritzing es el siguiente:

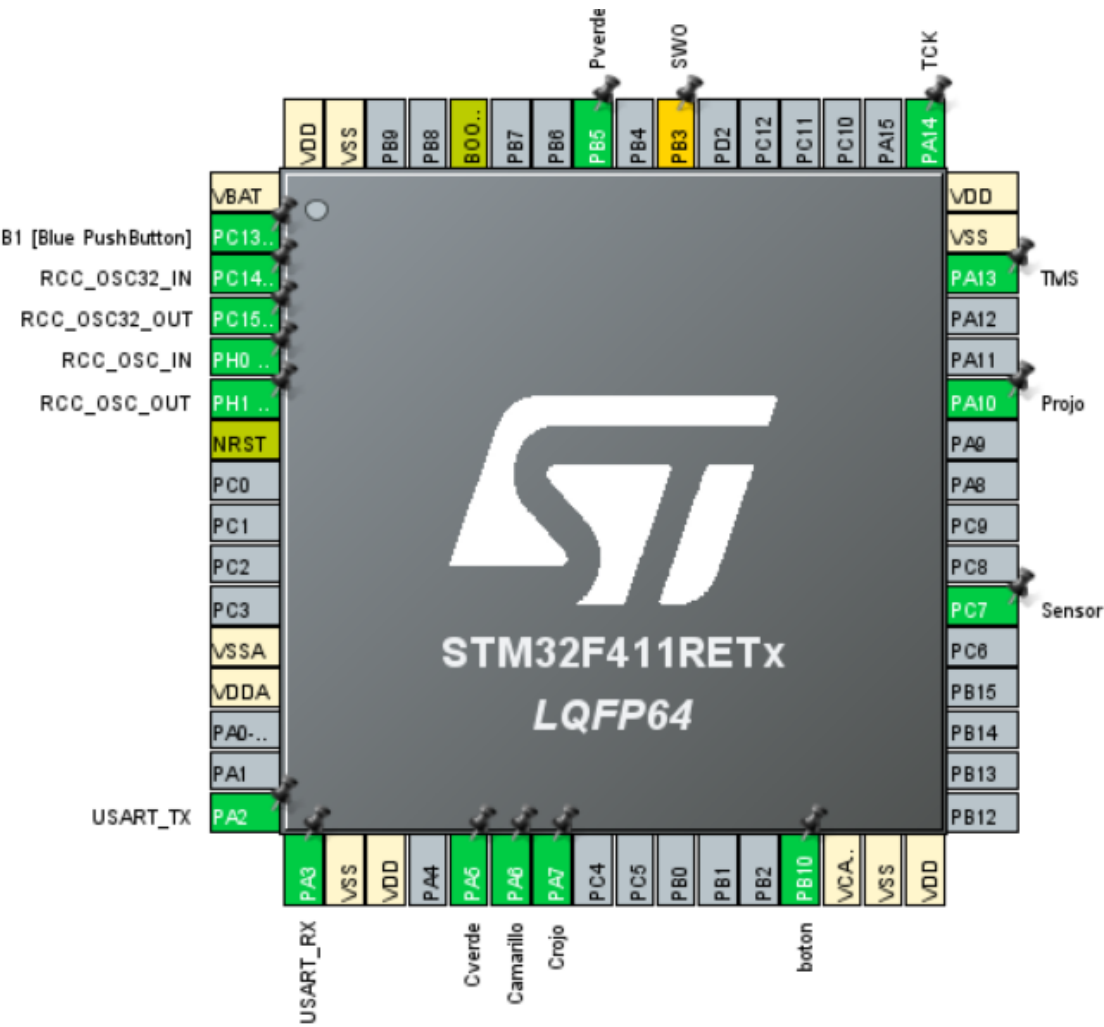


Componentes:

- Placa STM32
- Placa Grove
- Botón para cable de cuatro canales
- Ultrasonidos HC-SR04
- 2 LED rojos, 2 LED verdes, LED Amarillo
- Respectivas resistencias (resistencia explicada en prácticas anteriores)

Esquema STM32 CubeMX

El archivo STM32 Cube MX está adjuntado junto a esta memoria, por lo que no entraremos en más detalles.



Pin ...	Signal ...	GPIO o...	GPIO ...	GPIO ...	Maxim...	User L...	Modified
PA5	n/a	Low	Output...	No pull...	Low	Cverde	✓
PA6	n/a	Low	Output...	No pull...	Low	Camarillo	✓
PA7	n/a	Low	Output...	No pull...	Low	Crojo	✓
PA10	n/a	Low	Output...	No pull...	Low	Projo	✓
PB5	n/a	Low	Output...	No pull...	Low	Pverde	✓
PB10	n/a	n/a	Extern...	No pull...	n/a	boton	✓

Explicación de la solución:

En la línea 42 definimos las variables que usaremos para medir la distancia y el tiempo.

```
39 //Declaramos los posibles estados del programa
40 static enum {ST_Inicial, ST_Camarillo, ST_Pverde, ST_PARPADEO} next_state = ST_Inicial;
41 //Declaramos las variables que usaremos para medir la distancia
42 uint32_t tiempoTIM,tinicio,tfinal,pulso,distancia;
43
44 /* USER CODE END PM */
45
```

Entre las líneas 63 y 66 declaramos las funciones que utilizamos a lo largo de nuestro programa, para satisfacer la lógica deseada. Son las siguientes:

```
62 //Declaramos las funciones con las que contará el programa
63 void secuencia(int new_state);
64 void task_parpadeo(void);
65 void AEcho(void);
66 void ATrigger(void);
67 /* USER CODE END PFP */
68
```

Nuestro método main únicamente se compone de una llamada a la función secuencia, tal y como ideamos en la práctica anterior:

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

    secuencia(next_state);

    /* USER CODE BEGIN 3 */
}
```

En dicha función es donde reside toda la lógica del programa. Entraremos solo en detalle en aquellas partes que varíen y hayamos implementado respecto a las prácticas anteriores.

```
void secuencia(int new_state){
    switch (new_state)
    {
        case ST_Inicial:
            GPIOA -> ODR = GPIO_ODR_OD5_Msk | GPIO_ODR_OD10_Msk; //
            calcularDistancia();
            distancia = (pulso * 10/58);
            printf("Distancia: %d cm\r\n", distancia);

            if(distancia < 20){
                next_state = ST_Camarillo;
            }
            break;
        case ST_Camarillo:
            HAL_Delay(3000); //Esperamos los 3 segundos que dura el
            GPIOA->ODR = GPIO_ODR_OD6_Msk | GPIO_ODR_OD10_Msk; //Co
            HAL_Delay(3000);
            next_state=ST_PVerde;
            break;
        case ST_PVerde:
            GPIOA->ODR = GPIO_ODR_OD7_Msk; //Coche rojo activo
            GPIOB -> ODR = GPIO_ODR_OD5_Msk; //Peaton verde activo
            HAL_Delay(5000);
            next_state=ST_PARPADEO;
            break;
        case ST_PARPADEO:
            task_parpadeo();
            next_state= ST_Inicial;
            break;
    }
}
```

En el caso del estado inicial, lo que hacemos en esta práctica es calcular la distancia llamando a la función *calcularDistancia()*, que se encargará de

generar un pulso de 10 us y esperar recibir el echo, como observamos en la siguiente imagen:

```
void calcularDistancia(void)
{
    tiempoTIM=0;

    HAL_GPIO_WritePin(Sensor_GPIO_Port, Sensor_Pin, GPIO_PIN_RESET);
    HAL_Delay(4);
    HAL_GPIO_WritePin(Sensor_GPIO_Port, Sensor_Pin, GPIO_PIN_SET);
    HAL_Delay(10); //pulso de 10us
    HAL_GPIO_WritePin(Sensor_GPIO_Port, Sensor_Pin, GPIO_PIN_RESET);

    AEcho();
    while (!(HAL_GPIO_ReadPin(Sensor_GPIO_Port, Sensor_Pin)));

    tinicio=tiempoTIM;
    while (HAL_GPIO_ReadPin(Sensor_GPIO_Port, Sensor_Pin));

    ATrigger();
    tfinal=tiempoTIM;
    pulso=tfinal-tinicio;
}
```

El funcionamiento es simple: el trigger envía el pulso y el echo la recibe. El tiempo que tarda en recibirla es la distancia.

Se inicializa la variable tiempoTIM, que nos servirá para calcular el pulso. Posteriormente, con el pin en modo trigger en estado bajo, se pone en alto durante 10 us (pulso) y se vuelve a bajar.

Tenemos las siguientes funciones:

```
void AEcho(void){
    GPIO_InitTypeDef GPIO_InitStruct = {0};
    GPIO_InitStruct.Pin = Sensor_Pin;
    GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
    GPIO_InitStruct.Pull = GPIO_PULLDOWN;
    HAL_GPIO_Init(Sensor_GPIO_Port, &GPIO_InitStruct);
}
```

```
void ATrigger(void){
    GPIO_InitTypeDef GPIO_InitStruct = {0};
    GPIO_InitStruct.Pin = Sensor_Pin;
    GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
    GPIO_InitStruct.Pull = GPIO_NOPULL;
    GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
    HAL_GPIO_Init(Sensor_GPIO_Port, &GPIO_InitStruct);
}
```

Como se puede apreciar, son funciones antagónicas. Respectivamente se encargan de cambiar el modo del pin a Echo o Trigger, cambiando el modo de output a input y viceversa correspondientemente. De tal forma, somos capaces de calcular la distancia del obstáculo mediante el tiempo de respuesta calculado en la función *calcularDistancia()*.

Continuando con la función *secuencia()*, calculamos la distancia mediante la siguiente función (facilitada en el enunciado):

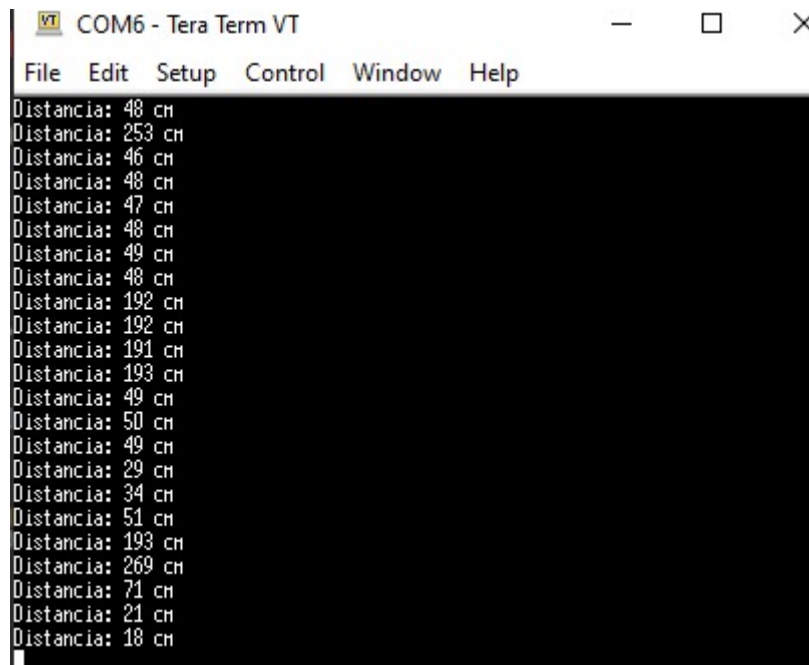
$$distance = pw * \frac{cm}{58\mu s}$$

Debemos comprobar que sea menor que 200mm para iniciar la secuencia y simular el funcionamiento del botón.

Por último, mediante la siguiente función lo que hacemos es incrementar la variable que hemos usado para controlar el tiempo, para posteriormente calcular el incremento respecto al valor inicial. Creemos que no es el método más ortodoxo de realizarlo, pero resulta efectivo:

```
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim){  
    tiempoTIM++;  
}
```

Ejemplo de ejecución (Tera Term)



The screenshot shows a Tera Term VT window titled 'COM6 - Tera Term VT'. The window has a menu bar with 'File', 'Edit', 'Setup', 'Control', 'Window', and 'Help'. The main area displays a list of distance measurements in centimeters (cm) on a black background with white text. The measurements are as follows:

Distance (cm)
48
253
46
48
47
48
49
48
192
192
191
193
49
50
49
29
34
51
193
269
71
21
18