PRÁCTICA 4

Índice

1. Objetivos

- Ilustrar los distintos usos de la modulación PWM para controlar actuadores mecánicos: servos y motores de DC.

- Implementar un controlador de un vehículo de radio control en base a la inclinación del controlador.

- Medir la inclinación usando un acelerómetro de 2 ejes.

- Controlar la posición del servo de la dirección proporcionalmente a la inclinación de la placa.

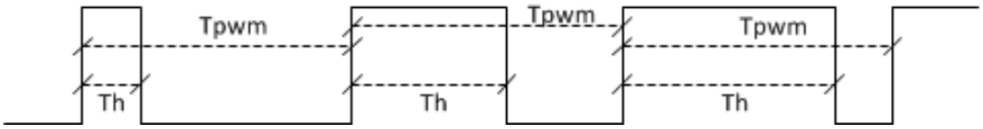
- Establecer la velocidad y sentido un motor de DC proporcionalmente a la inclinación de la placa.

Usaremos la modulación PWM de distintas formas para controlar tanto el servo como el motor de DC.

2. Introducción

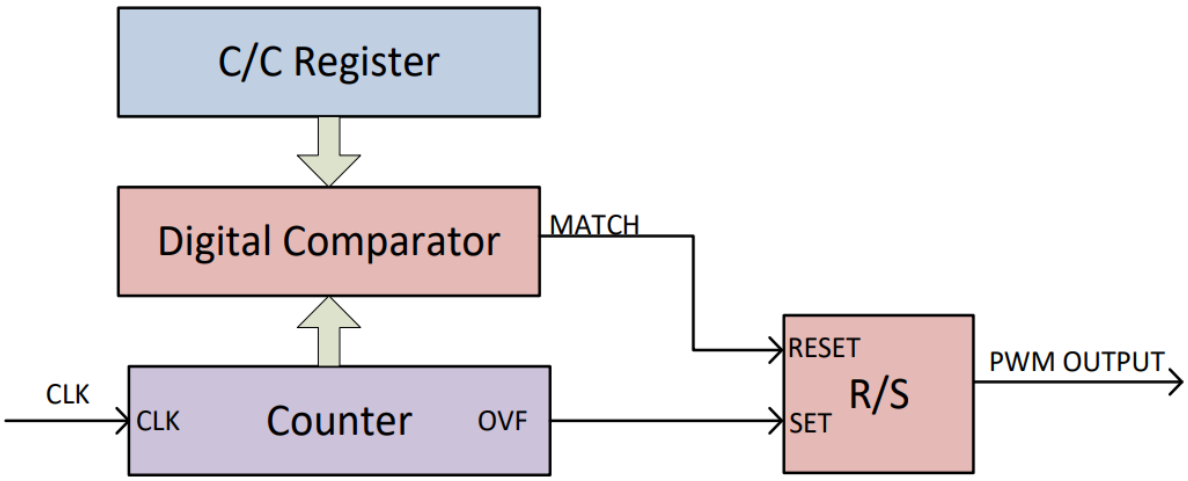
Esta práctica nos introduce a las señales PWM y a los sensores I2C. La modulación PWM se caracteriza por tener una señal con un periodo constante y un tiempo en alto, o Dutty-Cicle, variable. Esto permite codificar la información de dos maneras diferentes:

- Tiempo en alto: Lo crucial es Th, y Tpwm no es tan relevante.
- Dutty-Cicle: Lo importante es la relación entre Th y Tpwm.



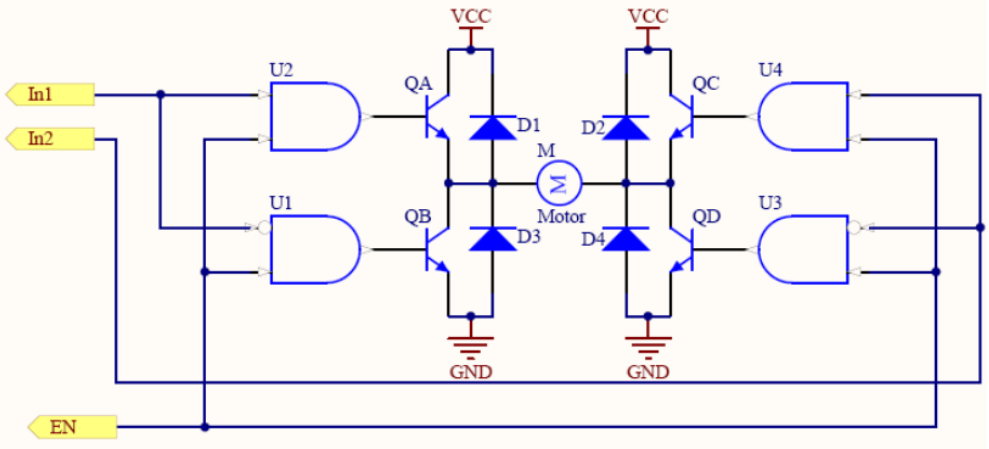
$$Dutty - Cycle \approx \frac{T_h}{T_{PWM}}$$

Generador de PWM



-En cuanto a las *etapas de potencia*, los pines GPIO del microcontrolador están diseñados para generar señales, por lo que no pueden suministrar mucha potencia. Los actuadores suelen operar con tensiones altas y requieren varios amperios. Por ello, es necesario un componente que suministre la potencia adecuada entre el microcontrolador y el actuador.

-Para *controlar motores de corriente continua (DC)*, se utilizan los puentes H, que proporcionan la potencia de manera bipolar. Están formados por 4 transistores de alta corriente conectados al voltaje que alimentará al motor. El microcontrolador solo necesita conmutar los transistores, y estos entregan la potencia necesaria. Gracias a esta topología, podemos controlar la dirección de giro del motor.



Al usar PWM para motores de DC, la velocidad del motor depende del voltaje aplicado en sus bornes:

$$\omega_{DCmotor} \approx K_{Speed} * V_{DCMotor}$$

Una manera de obtener velocidades intermedias es modificando el voltaje desde la perspectiva del motor. El puente H solo conmuta entre encendido y apagado. Para conseguir voltajes intermedios, la idea es encender y apagar el puente H rápidamente usando una señal PWM:

$$V_{DCmotor} \approx \frac{T_h}{T_{PWM}} V_{PowerSupply}$$

El motor actúa como un filtro pasa-bajos, eliminando los armónicos de alta frecuencia y obteniendo una componente continua proporcional al ciclo de trabajo de la señal PWM. En este caso, lo importante es la relación entre Th y Tpwm, y que Tpwm sea lo suficientemente pequeño para que el motor filtre los armónicos.

3. Desarrollo de la práctica

▪ En la fase 1 conectaremos el L298 a la placa de desarrollo siguiendo la imagen orientativa proporcionada en la práctica.

▪ En la fase 2 crearemos un nuevo proyecto STM32, donde seleccionaremos el MCU STM32L475VG.

▪ En la fase 3 configuraremos el reloj y los pines para el modo PWM. Primero, configuramos el reloj "HCLK" a 80MHz y luego los pines del microcontrolador usando la siguiente tabla proporcionada en la práctica:

Dispositivo	Pin	Periférico	Exterior
Motor (In1)	PA2	Timer 2 Channel 3	D9
Motor (In2)	PA15	Timer 2 Channel 1	D10
Pto. Serie	PB6	USART 1 Tx	VCP
	PB7	USART 1 Rx	VCP
IMU (LSM6DSL)	PB10	I2C2 - SCL	-
	PB11	I2C2 - SDA	-
Servo	PB1	Timer 3 Channel 4	D6

Luego, configuraremos los canales 1 y 3 de TIM2 como generadores de PWM, estableciendo el periodo del contador AutoReload Value a 4000, lo cual determinará el periodo de la señal PWM. Además, configuraremos USART1 en modo asíncrono.

▪ En la fase 4 probaremos el control del motor con un pequeño código. Para ello añadimos en "main.c", en la sección de "/* USER CODE BEGIN 2 */":

```
HAL_TIM_PWM_Start(&htim2, TIM_CHANNEL_1);
HAL_TIM_PWM_Start(&htim2, TIM_CHANNEL_3);
```

Crearemos una función auxiliar para controlar la velocidad del servo, añadiendo el siguiente código en "main.c", en la sección "/ USER CODE BEGIN 4 */":

```
void setMotorSpeed(int16_t speed){

    if (speed > 0 && speed < 4000){

        __HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_1, speed);
        __HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_3, 0);

    } else if (speed > -4000 && speed < 0){

        __HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_1, 0);
        __HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_3, -
speed);
```

```
        }

}
```

Luego, escribiremos un pequeño bucle que utilice la función anterior para mover el servo. Añadiremos el siguiente código en “main.c”, dentro del while(1):

```
int16_t th;

/* USER CODE END WHILE */

/* USER CODE BEGIN 3 */

for (th = 0; th < 4000; th += 100){

        setMotorSpeed(th);
        HAL_Delay(200);

}

for (th = 4000; th > 0; th -= 100){

        setMotorSpeed(th);
        HAL_Delay(200);

}

for (th = 0; th > -4000; th -= 100){

        setMotorSpeed(th);
        HAL_Delay(200);

}

for (th = -4000; th < 0; th += 100){

        setMotorSpeed(th);
        HAL_Delay(200);


}
```

▪ En la fase 5 activaremos el bus I2C. Para iniciarlo vamos a “Pinout & Configuration” → “Connectivity” → “I2C2” → “I2C”. Donde pondremos en el desplegable el valor “I2C”.

▪ En la fase 6, programaremos la función de inicialización y lectura del acelerómetro. Añadiremos el siguiente código en “main.c”, en la sección “/* USER CODE BEGIN 4 */”:

```
void initAccelerometer (void){

	uint8_t buffer[1];
	buffer[0] = 0x40;
	HAL_I2C_Mem_Write(&hi2c2, 0xD4, 0x10, I2C_MEMADD_SIZE_8BIT,
		buffer, 1, 1000);

}
int16_t readAccel(uint8_t axxis){

	uint8_t buffer[2];
	int16_t accel;
	HAL_I2C_Mem_Read(&hi2c2, 0xD4, 0x28 + 2*axxis,
		I2C_MEMADD_SIZE_8BIT, buffer, 2, 1000);
	accel = ((int16_t)(buffer[1]<<8) | buffer[0])*0.061;

	return accel;

}
```

▪ En la fase 7, controlaremos la velocidad y el sentido del motor con la aceleración/inclinación del eje x. Implementaremos los códigos escritos en la fase anterior, añadiendo el siguiente código en “main.c”, dentro del while(1):

```
int16_t accel = readAccel(1);
setMotorSpeed(4 * accel);
HAL_Delay(500);
```

▪ En la fase 8, se dará una breve explicación sobre nuestro servo y su funcionamiento.

▪ En la fase 9, configuraremos el PWM para servos. Primero, conectaremos el servo a la placa de desarrollo según la imagen proporcionada en la práctica. Para configurar el PWM, iremos a “Pinout & Configuration” -> “Timers” -> “Channel4”, y seleccionaremos “PWM Generation CH4” en el desplegable. Luego, en “Parameters Settings”, ajustaremos el “Prescaler” a 79, el “Counter period” a 20000, y el “Pulse” a 500.

▪ En la fase 10 controlaremos la posición del servo mediante la aceleración/inclinación del eje y. Para conseguir esto, añadimos en “main.c”, en la sección de “/* USER CODE BEGIN 4 */”:

```
void setServoPos(float ang){

    float servo = 500 + ((ang * 200) / 18);
    __HAL_TIM_SET_COMPARE(&htim3, TIM_CHANNEL_4, servo);


}
```

Por último, modificamos en “main.c”, dentro del while(1) el siguiente código:

```
int16_t th;
uint16_t servo;

int16_t accel = readAccel(1);
setMotorSpeed(4 * accel);
HAL_Delay(500);

int16_t accelServo = readAccel(0);
float ang = (1000 + accelServo) * 18 / 200;
setServoPos(ang);
```

4. Conclusión

Todas las fases se han completado satisfactoriamente gracias a las explicaciones detalladas en la hoja de la práctica.

En lo personal, la práctica ha sido intensa pero muy útil, ya que hemos aprendido a leer los datos proporcionados por el acelerómetro y a utilizar esa información para mover un servo y un motor.