MICROCONTROLADORES

Práctica No. 18. Generar tonos con modulo PWM.

1. Objetivo

- Uso del módulo PWM para generar señales de diferentes frecuencias.

2. Material y Equipo.

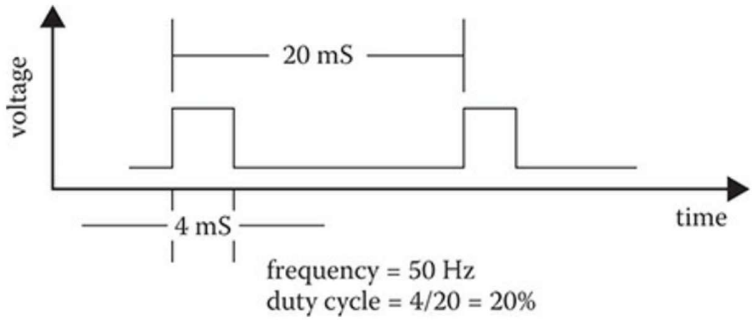
- Computador o laptop con STM32CubeIDE.
- Un Buzzer pasivo.

3. Marco de Referencia.

Los Timers tiene varios modos de funcionamiento aparte del modo normal uno de los más comúnmente usado es el modo PWM el cual nos permite hacer lo siguiente.

- Controlar el giro de un servomotor.
- Variar la luminosidad de un led.
- Controlar la velocidad de un motor de corriente continua.
- Generar diferentes tipos de señales como señales senoidales, triangulares o cuadradas.
- Desarrollar un generador de tonos acústicos.

La técnica de modulación por ancho de pulso (PWM) consiste en generar una señal con una frecuencia constante, pero poder variar el ciclo de trabajo de dicha señal esto es poder variar el tiempo que la señal permanece en alto sin modificar la frecuencia como se muestra en la siguiente figura.



En la figura anterior tenemos una señal PWM con un periodo de 20 ms (50 Hz) y tenemos un ciclo de trabajo de 4 ms en alto y 16 ms en bajo la idea es poder variar el ciclo de trabajo de la señal en alto mantenido el mismo periodo.

El STM32F103 todos sus timers cuenta con canales PWM para este fin. Un timer es configurado en modo PWM con la función HAL_TIM_PWM_ConfigChannel() y una instancia de la estructura C TIM_OC_InitTypeDef la cual está definida de la siguiente forma.

```
typedef struct {
  uint32_t OCMode;       /* Specifies the TIM mode. */
  uint32_t Pulse;        /* Specifies the pulse value to be loaded
                            into the Capture Compare Register. */
  uint32_t OCPolarity;   /* Specifies the output polarity. */
  uint32_t OCNPolarity;  /* Specifies the complementary output polarity.*/
  uint32_t OCFastMode;   /* Specifies the Fast mode state. */
  uint32_t OCIdleState;  /* Specifies the TIM Output Compare pin state during Idle state.*/
  uint32_t OCNIdleState; /* Specifies the complementary  TIM Output Compare pin
                            state during Idle state. */
} TIM_OC_InitTypeDef;
```

El primer elemento de la función define el modo de trabajo del timer en este caso es modo PWM definida por la constante TIM_OCMODE_PWM1, el segundo elemento se carga el valor del ciclo de trabajo que puede ir de 0 a 65535 y este valor. Para determinar el periodo de la señal PWM se usa la siguiente formula.

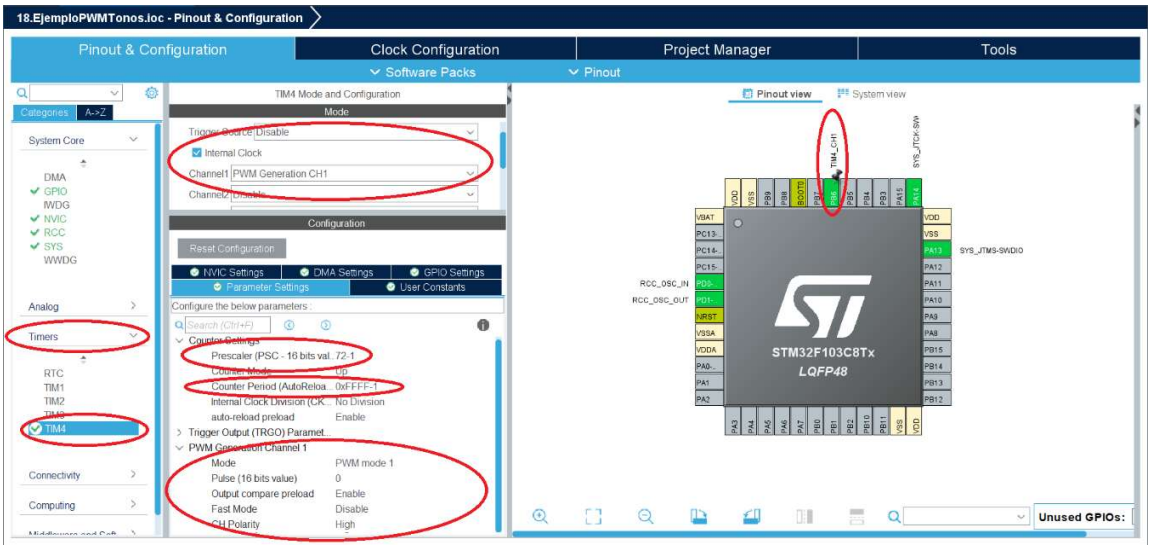
$$TIMERPeriod = \frac{TimeClock}{FREC\_Event}$$

Para el ciclo de trabajo usamos la siguiente formula.

$$TIMPulse = \left(\frac{(TIMPeriod + 1) * PWMDutyCycle}{100}\right) - 1$$

4. Desarrollo y Procedimiento.

Se creará un proyecto en el STM32CubeIDE como se indicó anteriormente.



El código de la práctica es el siguiente en el archivo main.c.

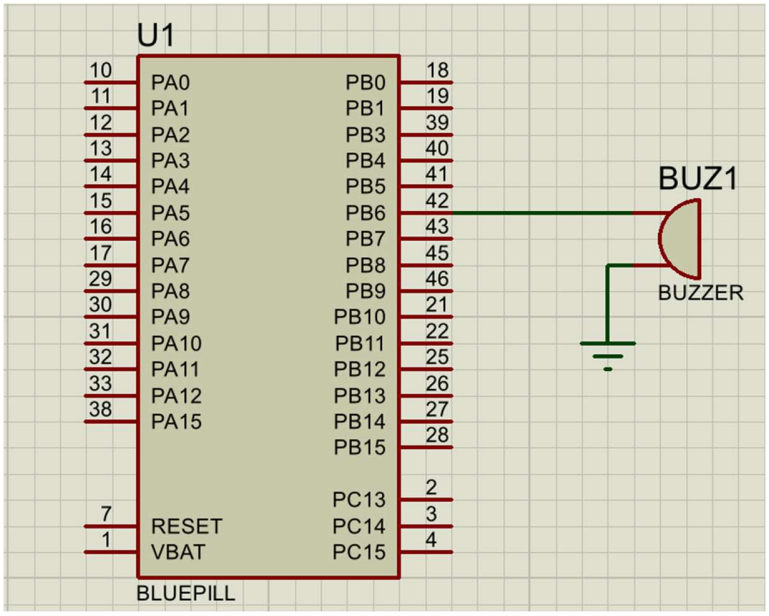
```c
#include "main.h"

#define SYSCLK      72000000
#define PRESCALER   72

#define Do          523
#define Re          587
#define Mi          659
#define Fa          698
#define Sol         784
#define La          880
#define Si          988

#define T1          500
#define T2          100

#define MUSICSIZE   48

TIM_HandleTypeDef htim4;

typedef struct{
    uint16_t freq;
    uint16_t time;
}SoundTypeDef;
const SoundTypeDef Music[MUSICSIZE] = {
    {Do, T1},
    {Do, T1},
    {Sol, T1},
    {Sol, T1},
    {La, T1},
    {La, T1},
    {Sol, T1},
    {0, T2},
    {Fa, T1},
    {Fa, T1},
    {Mi, T1},
    {Mi, T1},
    {Re, T1},
    {Re, T1},
    {Do, T1},
    {0, T2},
    {Do, T1},
    {Do, T1},
    {Sol, T1},
    {Sol, T1},
    {La, T1},
    {La, T1},
    {Sol, T1},
    {0, T2},
    {Fa, T1},
    {Fa, T1},
    {Mi, T1},
    {Mi, T1},
    {Re, T1},
    {Re, T1},
    {Do, T1},
    {0, T2},
    {Sol, T1},
    {Sol, T1},
```

```
    {Fa, T1},
    {Fa, T1},
    {Mi, T1},
    {Mi, T1},
    {Re, T1},
    {0, T2},
    {Sol, T1},
    {Sol, T1},
    {Fa, T1},
    {Fa, T1},
    {Mi, T1},
    {Mi, T1},
    {Re, T1},
    {0, T2},
};

int paso_sonido = 0;
char play_musica = 0;
int timer_sonido;
int sound_counter;

void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_TIM4_Init(void);
void sound(int freq, int time_ms){
    if(freq > 0){
        __HAL_TIM_SET_AUTORELOAD(&htim4, SYSCLK / PRESCALER / freq);
        __HAL_TIM_SET_COMPARE(&htim4, TIM_CHANNEL_1, __HAL_TIM_GET_AUTORELOAD(&htim4) / 2);
    }else{
        __HAL_TIM_SET_AUTORELOAD(&htim4, 1000);
        __HAL_TIM_SET_COMPARE(&htim4, TIM_CHANNEL_1, 0);
    }
    __HAL_TIM_SET_COUNTER(&htim4, 0);
    timer_sonido = ((SYSCLK / PRESCALER / __HAL_TIM_GET_AUTORELOAD(&htim4)) * time_ms) / 1000;
    sound_counter = 0;
    HAL_TIM_PWM_Start_IT(&htim4, TIM_CHANNEL_1);
}

void Sonido_Init(void){
    paso_sonido = 0;
    play_musica = 1;
    sound(Music[paso_sonido].freq, Music[paso_sonido].time);
}
void HAL_TIM_PWM_PulseFinishedCallback(TIM_HandleTypeDef *htim){
     if(__HAL_TIM_GET_FLAG(htim, TIM_FLAG_CC4) != RESET){
         __HAL_TIM_CLEAR_IT(htim, TIM_IT_CC4);
         sound_counter++;

         if(sound_counter > timer_sonido){
             if(play_musica == 0){
                 __HAL_TIM_DISABLE(htim);
             }else{
                 // mientras el indice "paso_sonido" sea menor del total MUSICSIZE
                 // de notas de la partitura
                 if(paso_sonido < MUSICSIZE - 1){
                     if(__HAL_TIM_GET_COMPARE(htim, TIM_CHANNEL_1) == 0){
                         // pasamos a la siguiente nota a reproducir
                         paso_sonido++;
                         // reproducimos la frecuencia de la nota de la partitura
                         sound(Music[paso_sonido].freq, Music[paso_sonido].time);
                     }else{
                         sound(0, 30);   // silencio durante 30 de tiempo
                     }
                 }else{
                     play_musica = 0;
                     __HAL_TIM_DISABLE(htim);
                 }
             }
         }
         if(__HAL_TIM_GET_FLAG(htim, TIM_FLAG_CC4OF) != RESET){
             __HAL_TIM_CLEAR_FLAG(htim, TIM_FLAG_CC4OF);
         }
     }
}
```

```
138 int main(void)
139 {
140     HAL_Init();
141     SystemClock_Config();
142     MX_GPIO_Init();
143     MX_TIM4_Init();
144     Sonido_Init();      // iniciamos los parametros que inician la melodia
145
146     while (1)
147     {
148     }
149 }
```

5. Esquemático del circuito.

El esquemático de la práctica se muestra a continuación.



6. Observaciones.

Esta sección es para que el alumno anote sus observaciones.

7. Conclusiones.

Esta sección es para que el alumno anote sus conclusiones.

8. Importante.

La práctica deberá ser validad en el salón de clases antes de anexar el reporte al manual de prácticas. Una vez validad realizar el reporte de practica como se anteriormente y anexar al manual de prácticas que se entregara a final del curso.