Pulse Width Modulation (PWM)

The mbed expansion board that is connected to the STM32 microcontroller has a piezo buzzer on it. This means it is possible to add sound effects to your programs! The buzzer is connected to GPIO pin PB10 which, looking at the datasheet, is attached to output channel 3 of timer 2. This means that we can use pulse width modulation to create the sounds.

There are two possible ways of using PWM:

1. By simply adjusting the frequency of the timer it is possible to create square waves of various pitches. This results in Gameboy-like sounds.
2. By having a fixed PWM frequency and varying the duty cycle it is possible to simulate sine waves. This will result in more pleasant noises, but the implementation is a bit more involved.

This document will describe method 1.

Configuring the Timer

Because only timer 2 is connected to the buzzer is it necessary to use another timer for time keeping in your program. We will use timer 15. The configuration is very much similar to timer 2, however, the ARR register is only 16-bit instead of 32-bit, so the maximum usable reload value is 65535. This means that you will need to use the prescaler to get to the lower frequencies. Here is an example that sets timer 15 to 100 Hz:

```
RCC->APB2ENR |= RCC_APB2Periph_TIM15; // Enable clock line to timer 2;
TIM15->CR1   = 0x0000;
TIM15->ARR   = 63999;  // Set auto reload value
TIM15->PSC   = 9;      // Set pre-scaler value
TIM15->DIER |= 0x0001; // Enable timer interrupt

NVIC_SetPriority(TIM1_BRK_TIM15_IRQn, 0);
NVIC_EnableIRQ(TIM1_BRK_TIM15_IRQn);

TIM15->CR1  |= 0x0001; // Enable timer
```

Notice that you will need to use the `TIM1_BRK_TIM15_IRQHandler ()` function for your interrupt logic.

To setup timer 2 for PWM output we start by setting it up as before, e.g.:

```
RCC->APB1ENR |= 0x00000001; // Enable clock line to timer 2;
TIM2->CR1   = 0x0000; // Disable timer
TIM2->ARR   = 1000; // Set auto reload value
TIM2->PSC   = PRESCALER_VALUE; // Set pre-scaler value
TIM2->CR1  |= 0x0001; // Enable timer
```

Then we configure the counter compare registers of the timer:

```
TIM2->CCER &= ~TIM_CCER_CC3P; // Clear CCER register
TIM2->CCER |= 0x00000001 << 8; // Enable OC3 output

TIM2->CCMR2 &= ~TIM_CCMR2_OC3M; // Clear CCMR2 register
TIM2->CCMR2 &= ~TIM_CCMR2_CC3S;
TIM2->CCMR2 |= TIM_OCMode_PWM1; // Set output mode to PWM1
```

```
    TIM2->CCMR2 &= ~TIM_CCMR2_OC3PE;
    TIM2->CCMR2 |=  TIM_OCPreload_Enable;

    TIM2->CCR3 = 500; // Set duty cycle to 50 %
```

This tells the timer to compare the value stored in CNT with the value stored in CCR3. If CNT is smaller than CCR3 it will send a logic high to channel 3 output and otherwise it will send a logic low.

Now we need to connect pin PB10 to the timer. First we configure it for alternate function:

```
    RCC->AHBENR |= RCC_AHBENR_GPIOBEN; // Enable clock line for GPIO bank B
    GPIOB->MODER   &= ~(0x00000003 << (10 * 2));    // Clear mode register
GPIOB->MODER   |=  (0x00000002 << (10 * 2));    // Set mode register
```

And then specify alternate function 1, i.e., PWM output:

```
    GPIO_PinAFConfig(GPIOB, GPIO_PinSource10, GPIO_AF_1);
```

Running this code should make the buzzer emit a constant tone. To change the tone we can simply write a different value to CCR3 and ARR:

```
    void setFreq(uint16_t freq) {
        uint32_t reload = 64e6 / freq / (PRESCALER_VALUE + 1) - 1;

        TIM2->ARR  = reload; // Set auto reload value
        TIM2->CCR3 = reload/2; // Set compare register

        TIM2->EGR |= 0x01;
   }
```

This function writes to the ARR register to set the frequency according to the input value. Then the CCR3 register is set to half the reload value resulting in a duty cycle of 50 %, and finally we write to the EGR register to inform the timer that the values have been changed.

That’s basically it. Using this code you should be able add music and sounds to your game!