Processor: STM32L021D4

IDE: Atollic TrueStudio

I am cycling the PWM frequency from 3.2KHz to 3.7KHz in 15mS steps and then repeating but I am hearing short pauses from the piezo circuit connected to the output. I have trapped the pwm output on the logic analyzer which shows periods that the pwm is not functional. The error always happens after updating the frequency but it is random. The times that the PWM stops is almost always 32mS in length. When the PWM output stops, the pin?s output state is dependent on the values in the OC3M bits of the TIM2->CCMR2 register which leads me to this excerpt from the reference manual?

> *110: PWM mode 1 - In upcounting, channel 1 is active as **long as TIMx_CNT<TIMx_CCR1 else inactive**. In downcounting, channel 1 is inactive (OC1REF=?0) as long as TIMx_CNT>TIMx_CCR1 else active (OC1REF=1).*
> *111: PWM mode 2 - In upcounting, channel 1 is inactive as **long as TIMx_CNT<TIMx_CCR1 else active.** In downcounting, channel 1 is active as long as TIMx_CNT>TIMx_CCR1 else inactive.*

As a test, I cleared the TIM2 counter register when updating the ARR register and the gaps went away but at a cost of adding other sound issues because I am interrupting the pwm signal, similar to stopping and starting the timer module.

I tried a constant output and it is good so it is definitely related to switch frequencies. The source code has been condensed to a couple sample functions to eliminate other interference. In the capture below, the DBG line is toggled every 15mS when changing frequencies which shows the program is running as normal.

I am loading new values into the ARR and CCR3 registers which contains a shadow copy that is not updated until the next PWM cycle. I was originally trying to turn off the clock, update the values, and then enable the clock but this causes interruptions of the PWM waveform which creates a poor tone on the piezo.

Code

```
void piezo_test(void)

{

    RCC->APB1ENR |= RCC_APB1ENR_TIM2EN;

    uint32_t i = 0;



    const uint32_t freq_list2[][2] =                    // Precalculate the timer values to rule out any math errors.

    {

            // Freq     DC @50%

            {625,  312 },      // 2Mhz / 3.2Khz

            {620,  310 },

            {615,  307 },

            {610,  305 },

            {606,  303 },
```

```
                {601,  300 },
                {597,  298 },
                {592,  296 },
                {588,  294 },
                {583,  291 },
                {579,  289 },
                {575,  287 },
                {571,  285 },
                {567,  283 },
                {563,  281 },
                {559,  279 },
                {555,  277 },
                {551,  275 },
                {547,  273 },
                {544,  272 }        // 2MHz @ 3.675 KHz



        };
        enable_piezo_output();                                  // Set IO for alt function on A10, TIM2_CH3
        startPWM_Test( freq_list2[0][0], freq_list2[0][1]);                 // kick off first frequency
        while(1)
        {
            if ( scheduler_timeSince( tone_timer_ms ) >= 15 )               // wait 15mS
            {
                tone_timer_ms = scheduler_getTick();                    // update timer
                i = (i + 1 ) % 16;                                  // Increment freq index
                startPWM_Test( freq_list2[i][0], freq_list2[i][1]);             // Setup new freq and duty cycle


                dbg_on(1);                                          // Toggle debug line to show when freq update
loaded
                dbg_off(1);                                         //
            }
        }
    }
```

```
void startPWM_Test( uint32_t clk, uint32_t dc )

{

    TIM2->ARR = clk;                                    // Load ARR with the rollover value

    TIM2->CCR3 = dc;                                    // Set to half of ARR for 50% duty cycle

    //TIM2->CNT = 0;


    if(( TIM2->CR1 & TIM_CR1_CEN ) == 0)                // first pass turn on timer and cfg,
otherwise just load CCR2 and ARR to shadow registers

    {

        TIM2->CCER &= ~TIM_CCER_CC3E;

        TIM2->PSC = 0;                                  // Set prescaler to 0, so APBCLK/1 i.e 1MHz

        TIM2->CCMR1 = 0;                                // No input capture

        TIM2->CCMR2 = TIM_CCMR2_OC3M_2 | TIM_CCMR2_OC3M_1 | TIM_CCMR2_OC3M_0
|TIM_CCMR2_OC3PE;// Set OC3 to PWM

        TIM2->CCER |= TIM_CCER_CC3E;                    // Enable the output on OC3
(CC3E = 1)e

        TIM2->CR1 |= TIM_CR1_CEN;                       // Enable counter (CEN = 1)

    }

}
```