Generating Frequencies on the STM32L432KC: A Guide to Timers

For generating frequencies ranging from 220Hz to 1kHz on the STM32L432KC microcontroller, the **general-purpose timers (TIM2, TIM15, TIM16)** are excellent choices. These timers are versatile and can be easily configured to output a Pulse Width Modulated (PWM) signal to a GPIO pin, effectively creating a signal of a specific frequency.

The Best Timer and GPIO Combination

The most straightforward choice for this application, especially when using a NUCLEO-L432KC development board, is **TIM2**. Several of its channels can be routed to easily accessible GPIO pins. For instance, **TIM2_CH1 can be connected to pin PA0**, and **TIM2_CH2 can be connected to pin PA1**. These are readily available on the Arduino Nano-compatible headers of the Nucleo board, making them convenient for prototyping.

Why TIM2 is a great choice:

- **16-bit resolution:** Offers a good balance between frequency range and precision.
- **Multiple Channels:** Provides up to four independent channels for generating different frequencies or PWM signals simultaneously.
- **Ease of Use:** It is a standard general-purpose timer with well-documented features.

Key Formulae for Frequency Generation

The frequency of the output signal is determined by the timer's clock frequency and the values of two key registers: the **Prescaler (PSC)** and the **Auto-Reload Register (ARR)**. The relationship is defined by the following formula:

$$\text{Frequency} = \frac{\text{Timer Clock Frequency}}{(\text{PSC} + 1) \times (\text{ARR} + 1)}$$

- **Timer Clock Frequency:** This is typically derived from the Advanced Peripheral Bus (APB) clock. For the STM32L432KC, the timers on APB1 (like TIM2) often run at the system clock frequency, which defaults to 80 MHz on the NUCLEO-L432KC.
- **PSC (Prescaler):** This 16-bit register divides the timer clock frequency. A value of 0 means no division, 1 means division by 2, and so on.
- **ARR (Auto-Reload Register):** This 16-bit register holds the value at which the timer's counter will reset to 0.

To generate a desired frequency, you need to find a combination of `PSC` and `ARR` values that satisfy the equation. For the 220Hz to 1kHz range, you have a lot of flexibility. A common approach is to choose a `PSC` value that brings the timer's counting frequency into a manageable range, and then calculate the `ARR` value.

Example Calculation for 500Hz:

Assuming the timer clock is 80 MHz:

1. **Choose a Prescaler (PSC) value.** Let's aim for a timer tick frequency of 100 kHz.
    - `PSC = (80,000,000 / 100,000) - 1 = 799`
2. **Calculate the Auto-Reload (ARR) value.**
    - `ARR = (100,000 / 500) - 1 = 199`

With `PSC = 799` and `ARR = 199`, the output frequency will be 500Hz. You can adjust these values to achieve any frequency within your desired range.

Essential Register Configuration

To generate a PWM signal on a GPIO pin, you'll need to configure both the GPIO pin itself and the timer peripheral. Here's a breakdown of the necessary register settings, using **TIM2_CH1 on PA0** as an example:

1. GPIO Configuration

First, you need to enable the clock for the GPIO port and configure the pin to its alternate function mode.

- **Enable GPIOA Clock:** Set the `GPIOAEN` bit in the `RCC_AHB2ENR` register.
    - `RCC->AHB2ENR |= RCC_AHB2ENR_GPIOAEN;`
- **Configure PA0 Mode:** Set the `MODER0` bits in the `GPIOA_MODER` register to '10' for Alternate Function mode.
    - `GPIOA->MODER &= ~GPIO_MODER_MODE0_0;`
    - `GPIOA->MODER |= GPIO_MODER_MODE0_1;`
- **Select Alternate Function:** Set the `AFSEL0` bits in the `GPIOA_AFRL` register to '0001' (AF1) to connect TIM2 to PA0.
    - `GPIOA->AFR[0] |= GPIO_AFRL_AFSEL0_0;`

2. Timer Configuration

Next, configure the timer to generate the PWM signal.

- **Enable TIM2 Clock:** Set the `TIM2EN` bit in the `RCC_APB1ENR1` register.

    - `RCC->APB1ENR1 |= RCC_APB1ENR1_TIM2EN;`
- **Set Prescaler and Auto-Reload Values:** Load the calculated values into the `PSC` and `ARR` registers.
    - `TIM2->PSC = 799;` // For a 100kHz timer clock
    - `TIM2->ARR = 199;` // For a 500Hz output frequency
- **Configure PWM Mode:** In the `TIM2_CCMR1` register, for channel 1, set the `OC1M` bits to '110' for PWM mode 1. Also, enable the output compare 1 preload by setting the `OC1PE` bit.
    - `TIM2->CCMR1 |= TIM_CCMR1_OC1M_2 | TIM_CCMR1_OC1M_1;`
    - `TIM2->CCMR1 |= TIM_CCMR1_OC1PE;`
- **Set Duty Cycle:** The duty cycle is controlled by the Capture/Compare Register (`CCR1`). For a 50% duty cycle, set `CCR1` to half of the `ARR` value.
    - `TIM2->CCR1 = 100;` // 50% duty cycle (100 / 200)
- **Enable Capture/Compare Output:** Set the `CC1E` bit in the `TIM2_CCER` register to enable the output on the corresponding pin.
    - `TIM2->CCER |= TIM_CCER_CC1E;`
- **Enable the Timer:** Finally, set the `CEN` (Counter Enable) bit in the `TIM2_CR1` register to start the timer.
    - `TIM2->CR1 |= TIM_CR1_CEN;`

By following these steps and adjusting the `PSC` and `ARR` registers, you can accurately generate any frequency between 220Hz and 1kHz on your STM32L432KC.

To learn more about PWM and Timers on the STM32, check out this [STM32 Beginners Guide to PWM and Timers](http://googleusercontent.com/youtube_content/0).
http://googleusercontent.com/youtube_content/0