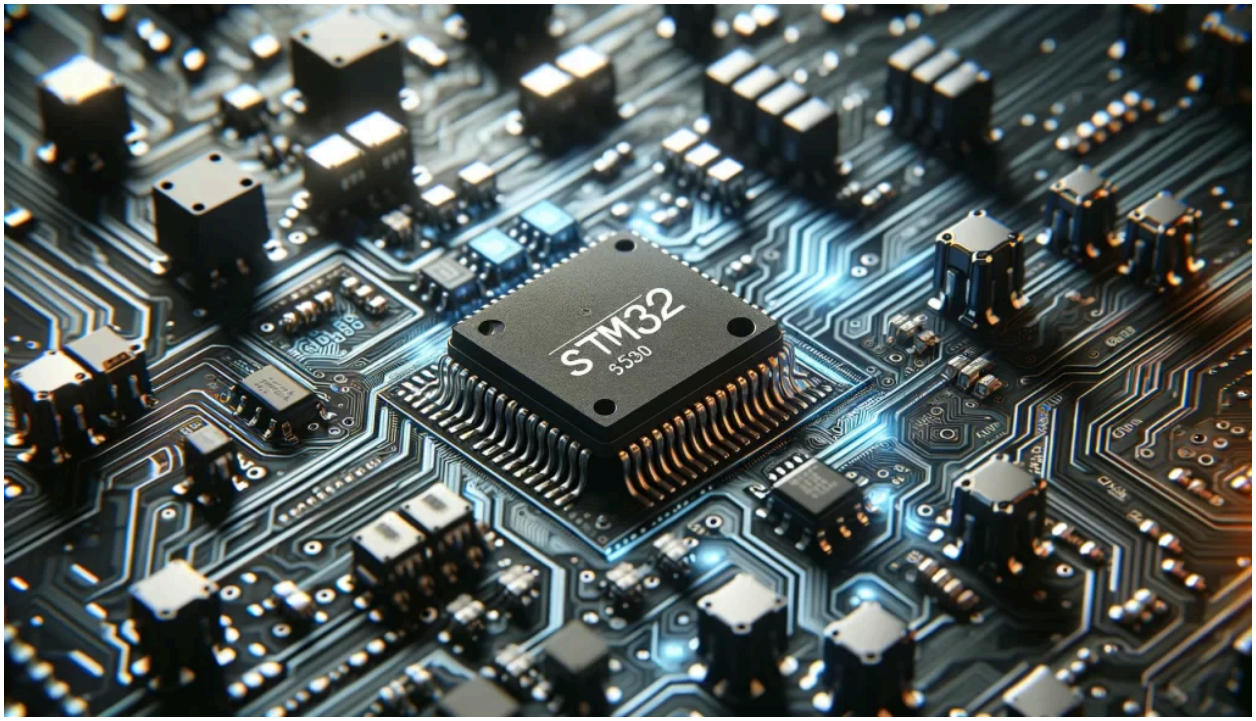
UNDERSTANDING PWM AND CLOCKS IN STM32

ROHITH S - HARDWARE IOT



How a Timer Works and Its Relationship with Clock

In simple terms, an STM32 timer is like a counting device that increments at regular intervals determined by the clock frequency. This timer allows the microcontroller to keep track of time, generate waveforms (like PWM for servos), or trigger events after specific intervals. Here's the basic flow:

1. **Clock Frequency**: A clock with a certain frequency (e.g., 32 MHz) drives the timer. This means that the clock "ticks" 32 million times per second.
2. **Counting with the Timer**: The timer counts these ticks, incrementing its value with each tick.
3. **Generating Timing Intervals**: By counting up to a specific value, the timer can create time intervals. For instance, if it counts up to 1000 and then resets, it creates a short delay.
4. **Impact of Clock Frequency**: The clock frequency affects how fast the timer counts. A higher clock frequency (e.g., 32 MHz) means the timer increments very quickly, while a lower clock frequency (e.g., 8 MHz) means the timer counts more slowly.

Detailed Explanation: How Timers and Clock Work in STM32

1. **Clock and Frequency Basics**
    - The clock frequency in a microcontroller is the rate at which its internal operations occur, measured in Hertz (Hz). For example, a 32 MHz clock ticks 32 million times per second.
    - The unit "Hz" signifies "cycles per second," so 32 MHz means 32 million cycles per second.

2. **How Timers Use Clock Cycles**
    - Timers in STM32 use these clock cycles to increment their internal count. They're typically set up to do one of the following:
        - Count up to a specified value (Auto-Reload Register or ARR) and then reset to zero.
        - Generate specific time intervals or pulse widths by configuring the count limit and reset behavior.
    -
    - Each clock cycle represents a tiny unit of time. For example, with a 32 MHz clock, each tick lasts about:

$$T_{\text{tick}} = \frac{1}{F_{clock}} = \frac{1}{32\,000\,000} \approx 31.25\,\text{ns}$$

Understanding Timer Configuration with PSC and ARR

- The timer's behavior depends on two main values:
  - **Prescaler (PSC)**: Divides the clock frequency, slowing down the timer. For instance, with a PSC of 9, a 32 MHz clock effectively becomes a 3.2 MHz clock for the timer.
  - **Auto-Reload Register (ARR)**: Sets the maximum count value. When the timer count reaches ARR, it resets back to zero and can trigger an event (like toggling a PWM output or triggering an interrupt).
-

Using Timers to Generate a Specific Frequency

- To achieve a particular timing or frequency, the timer clock frequency is calculated as:

$$F_{timer} = \frac{F_{clock}}{(PSC + 1)}$$

  - Then, to set up a timing interval or PWM, we set ARR to get the desired frequency or period.

2. **Practical Example for Generating a PWM Signal**
   - Assume:
     - Clock = 32 MHz32MHz
     - PSC = 99 (divides the clock by 10 to make it 3.2 MHz)
     - We want a frequency of 50 Hz50Hz (or a period of 20 ms20ms).
   -
   - Rearranging gives us:

$$ARR = \frac{3.2\,\text{MHz}}{50\,\text{Hz}} - 1 = 63999$$

Summary

- Timers count based on system clock frequency, which directly impacts their counting speed.
- Prescalers divide clocks, slowing down timer counts.

- ARR sets limits for timer counts to generate desired timing intervals or frequencies.
- By adjusting PSC and ARR, you can control frequency and period, allowing timers to fit specific application needs.

Calculation of Prescaler (PSC) and Auto-Reload Register (ARR) for 50Hz PWM Signal

To calculate the **Prescaler (PSC)** and **Auto-Reload Register (ARR)** values for a 50Hz PWM signal on an STM32 board with a 32MHz clock, we can use the following formulas:

Frequency Calculation

- Desired PWM frequency (FPWMFPWM): 50 Hz
- Corresponding period:

$$T_{PWM} = \frac{1}{F_{PWM}} = 20\,\text{ms}$$

Given Parameters

- Timer clock frequency (FtimerFtimer): 32,000,000 Hz (32 MHz)

ARR and PSC Calculation Formulas

1. **Choose a Large ARR Value for Higher Resolution**:
    - Aim for a high PWM resolution by choosing an ARR as large as possible, typically close to 65535.

2. **Calculate PSC Based on ARR and Desired PWM Frequency**:
    - Using the formula:

$$PSC = \frac{F_{timer}}{ARR \times F_{PWM}} - 1$$

- To simplify this process, let's start with $ARR = 64000$:

$$PSC = \frac{32\,000\,000}{64000 \times 50} - 1$$

$$PSC = \frac{32\,000\,000}{3\,200\,000} - 1$$

$$PSC = 10 - 1$$

$$PSC = 9$$

Summary of Values

- **PSC**: 9
- **ARR**: 64000

Implementation

Set these values in CubeMX or in your initialization code to generate a stable 50Hz PWM suitable for servo control with a 32MHz clock.

What is ARR (Auto-Reload Register)?

In simple terms, ARR stands for Auto-Reload Register. It's a value used by timers in microcontrollers like STM32 to define the duration of one full cycle of a PWM signal. It essentially sets the "count limit" for the timer. When the timer's counter reaches this value, it resets to zero and starts counting again.

Simple Explanation

For example, if you want a PWM signal that repeats every 20 milliseconds (for a frequency of 50 Hz), the timer needs to count up to a value that represents this duration. Thus:

- **ARR** defines the period of your PWM cycle.
- A higher ARR results in a longer cycle and thus a lower frequency.

Detailed Explanation: How ARR Works in STM32 Timer

1. **Timer Basics: Counting and ARR Limit**

- The STM32 timer counts up from zero. Each tick represents one increment of the counter based on the timer's clock.
- ARR sets the upper limit: When the counter reaches this value, it resets to zero and starts counting up again, marking the end of one PWM cycle.

2. **PWM Period and Frequency Control**
   - Frequency is determined by how long it takes for the counter to go from 0 to ARR.
   - A higher ARR means that it takes longer for the timer to reach its limit, resulting in lower frequency.
   - The PWM period is defined by:

$$Period = \frac{(PSC + 1) \times (ARR + 1)}{F_{timer}}$$

where:

- PSCPSC is the prescaler that divides input timer frequency.
- ARRARR is auto-reload register value.
- FtimerFtimer is timer’s clock frequency (e.g., 32 MHz).

Duty Cycle Control Using Compare Register (CCR)

- The ARR defines full period while CCR controls duty cycle; e.g., if ARR=1000ARR=1000 and CCR=500CCR=500, then duty cycle will be 50% because signal will be high for half period and low for other half.

Factors Affecting ARR

- Desired Frequency: To lower frequency, increase ARR; to increase frequency, decrease ARR.
- Prescaler (PSC): Affects how quickly counter ticks; indirectly influences necessary ARR values.
- Timer Clock Speed (FtimerFtimer): Higher speeds allow finer control but may need higher PSC values.

Example Calculation

Assuming:

- Timer clock Ftimer=32 MHzFtimer=32MHz
- Prescaler PSC=9PSC=9
- ARR=64000ARR=64000

Then:

$$Period = \frac{(9+1) \times (64000+1)}{32\,000\,000} \approx 0.02\,seconds = 20\,ms$$

This configuration yields a PWM frequency of **50 Hz**.

Summary

ARR defines timer's counting limit directly affecting PWM signal's frequency. Changing ARR alters this frequency impacting applications like servo control where specific timing is crucial.

What is PSC (Prescaler)?

In simple terms, PSC stands for Prescaler. It divides input clock frequency of a timer to slow it down. This division allows timer to count more slowly which is often necessary to achieve lower frequencies for applications like PWM.

Detailed Explanation: How PSC Works in STM32 Timers

1. **Timer Clock and PSC Division**
    - The STM32 timer counts at rate of its clock derived from system clock; prescaler slows down this counting speed.
    - For example, if your timer clock is 32 MHz32MHz and PSCPSC is set to 99: Timer\Count\Rate=\frac{32\,MHz}{10}=3.2\,MHz

2. **Frequency Reduction**

- A higher PSC means slower counting producing lower frequencies; essential when you need precise slower timing as it helps adjust timer practical application-specific frequencies.
- With PSC of 9, timer only "ticks" once for every ten input clock cycles.

3. **Formula for Frequency and Period with PSC**
   The period of PWM depends on both PSC and ARR:

$$Period = \frac{(PSC + 1) \times (ARR + 1)}{F_{timer}}$$

4. **Using PSC for Different Frequencies**
   By setting PSC appropriately you can achieve wide range output frequencies from single input clock.
5. **Factors Affecting PSC Choice**
   - Desired Frequency or Period: Higher PSC chosen for lower frequencies; lower values used for higher frequencies.
   - System Clock: Faster clocks may require more adjustment of PSC.
   - Timer Resolution Needs: Lower PSC values give finer control over timing because increments faster giving more points per cycle.

Example Calculation Using PSC

Assuming:

- Timer clock F_{timer}=32MHz
- Desired PWM frequency F_{PWM}=50Hz
- ARR=64000

To find PSC:

$$PSC = \frac{F_{timer}}{ARR \times F_{PWM}} - 1$$

Substituting values gives:

$$PSC = \frac{32,000,000}{64000 \times 50} - 1$$

Effect of CCR on PWM Signal for Servo Control

Introduction to CCR in PWM

In STM32 microcontrollers, the **Compare Capture Register (CCR)** controls the duty cycle in a PWM signal. The duty cycle determines the time a signal remains high during each cycle, controlling the pulse width for applications like servo motors.

Formula to Calculate CCR for a Given Duty Cycle

If you have a timer with a configured **Auto-Reload Register (ARR)** and want to set a PWM duty cycle, the CCR value can be calculated as:

$$\text{CCR} = \left(\frac{\text{Duty Cycle}}{100}\right) \times \text{ARR}$$

Where:

- **Duty Cycle** is the desired percentage of the "on" time (e.g., 50% for half of the time period).
- **ARR** (Auto-Reload Register) sets the maximum count value for the timer, which determines the period of the PWM signal.

Example

If:

- The timer's **ARR** is set to 1000 (for example, a PWM period of 20 ms),
- And the desired duty cycle is **25%**,

Then:

$$\text{CCR} = \left(\frac{25}{100}\right) \times 1000 = 250$$

In this case, setting the **CCR** to 250 would result in a 25% duty cycle PWM signal.

CCR's Role in Servo Control

For servos, the CCR value sets the pulse width:

- **0 degrees**: A lower CCR value.
- **180 degrees**: A higher CCR value.

For your setup:

- **0 degrees**: Calculate a lower CCR value.
- **180 degrees**: Set CCR to **450** for the desired pulse width.

Implementation Example

To move the servo, use the following:

- **For 0°**: Calculate the CCR for a 1ms pulse.

$$\text{CCR}\,(0^\circ) = \frac{1\ \text{ms}}{20\ \text{ms}} \times 1250 \approx 62$$

- **For 90°**: Calculate the CCR for a 1.5ms pulse.

$$\text{CCR}\,(90^\circ) = \frac{1.5\ \text{ms}}{20\ \text{ms}} \times 1250 \approx 94$$

- **For 180°**: Calculate the CCR for a 2ms pulse.

$$\text{CCR}\,(180^\circ) = \frac{2\ \text{ms}}{20\ \text{ms}} \times 1250 \approx 125$$

STM32CubeIDE Settings

Click Timer → Click TIM2 →

Clock Source set to Internal Clock

Channel3 set to PWM Generation CH3

Configuration → Parameter Settings →

Prescaler set to 180-1 (APB1Timer*2-1)

Counter Period 1000-1

Summary

PSC divides timer clock making it run at lower frequencies; used along with ARR adjusts period and frequency of PWM or other applications. This comprehensive formatting ensures all mathematical expressions are correctly presented while maintaining logical flow from basic concepts through detailed explanations related to timers in STM32 microcontrollers without altering any original content or meaning.

Refer:

https://deepbluembedded.com/stm32-servo-motor-control-with-pwm-servo-library-examples-code/