**PWM based Permanent Magnet DC Motor Speed Control**

**1. Aim:**

To generate the Pulse Width Modulated (PWM) signal for different duty cycles to control the DC motor.

**2. Introduction:**

Pulse width modulation (PWM) is a simple digital technique to control the value of an analog variable. PWM uses a rectangular waveform to quickly switch a voltage source on and off to produce a desired average voltage output. Although the output is binary at any time instant, the average output over a time span can be any value between 0 and the maximum voltage.

Specifically, the percentage of time in the on state within one period is proportional to the mean value of the voltage output. Consequently, when software changes the duration of the on state, the output voltage is adjusted accordingly to emulate an analog signal.

PWM has been widely used in a variety of applications, such as motor speed and torque control, digital encoding in telecommunications, DC-to-DC power conversion, and audio amplification. We should select the PWM switching frequency carefully to avoid serious negative impacts on applications. For example, the PWM switching frequency of an LED light must be at least 120 Hz to prevent the flickering effects that humans can see.

**2.1 DC Motor Control:**

The objective of this exercise is to control the rotational speed of a DC motor. The 5 V DC motor has a resistance of 10Ω. Your control system will apply a variable power to the motor by varying the duty cycle of a 50 Hz signal, as shown below.

A picture containing diagram, line, text, plan

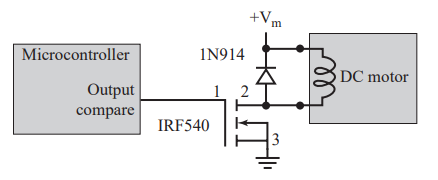
Description automatically generated

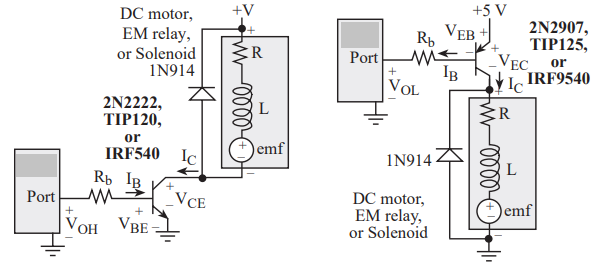
**2.2 DC Motor Interface using a High-Current MOSFET:**

The circuits in this section can be used to interface solenoids, EM relays, and DC motors because they all have electromagnets that behave like Figure. To interface an electromagnet, we consider voltage, current, and inductance. First, we need a power supply at the desired voltage requirement of the coil.

It is important to realize that many devices cannot be connected directly up to the microcontroller. In the specific case of motors, we need an interface that can handle the voltage and current required by the motor.

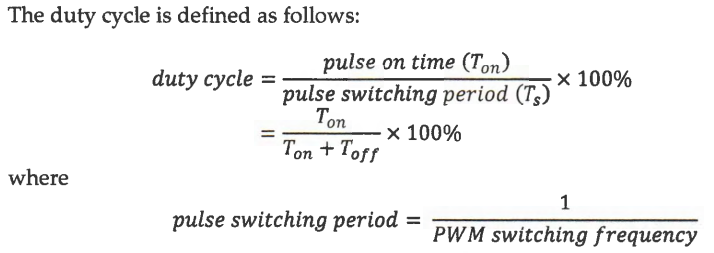
If the only available power supply is larger than the desired coil voltage, we use a voltage regulator (rather than a resistor divider) to create the desired voltage. We connect the power supply to the positive terminal of the coil, shown as V in Figure. We will use an NPN transistor to drive the negative side of the coil to ground. The other way to interface the coil is to use a PNP transistor to drive the positive side of the coil to V. The computer can turn the current on and off by controlling the base of the transistor.

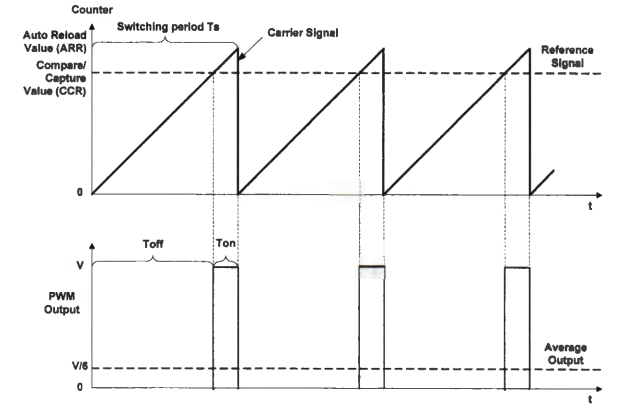




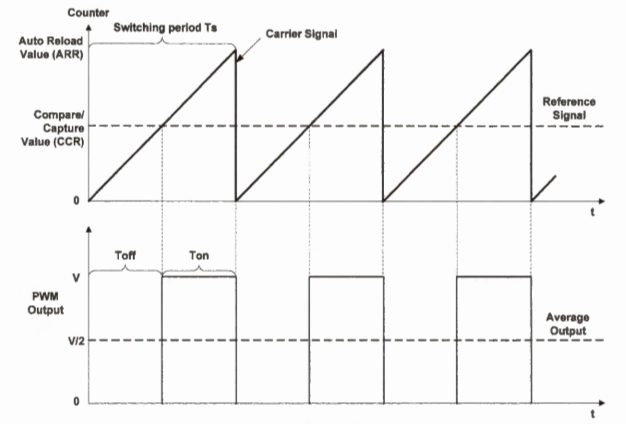
**3. Microcontroller Implementation:**

The average value of a simple PWM output based on a sawtooth carrier signal and a constant reference, as illustrated in Figure.



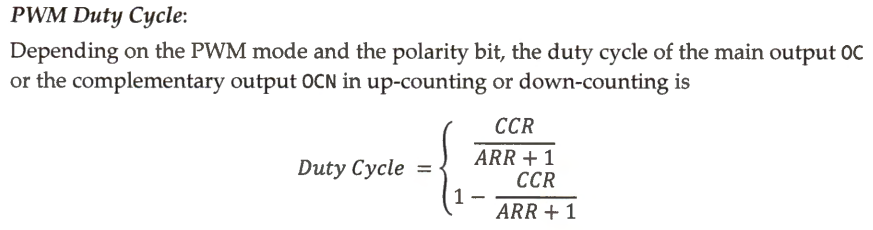


**Example of simple PWM when duty cycle is 1/6**



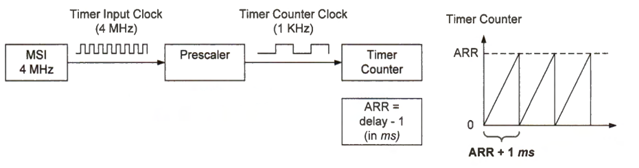
**Example of simple PWM when duty cycle is 1/2**

The PWM output signal is determined by three factors: comparison between the timer counter (CNT) and the given reference valued stored in the compare and capture register (CCR), the PWM output mode, and the polarity bit.



**3.1 Determination of ARR and CCR Values:**

Assume that one need to generate a PWM signal of 100Hz, with 60% duty cycle. If the given microcontroller is running at 4MHz, find the prescaler, ARR and CCR values.

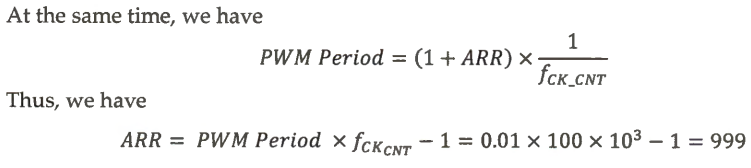


In this example, the prescaler factor is set as 39, thus the frequency at which the counter increments is

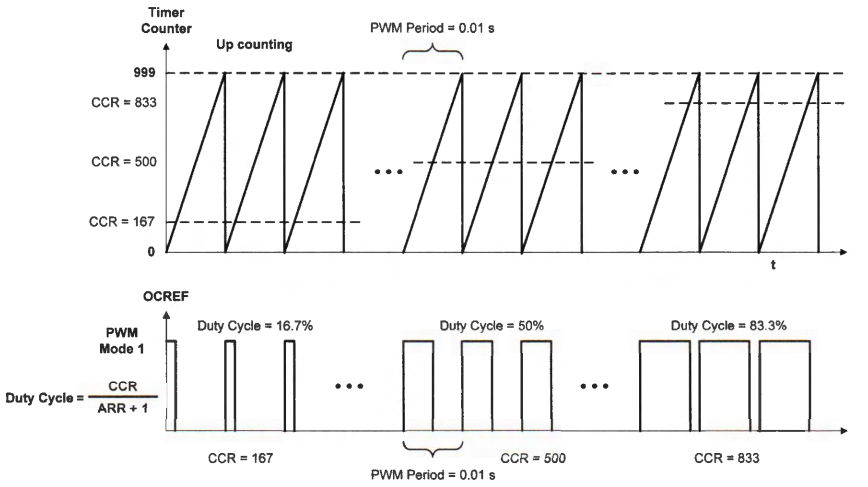


For a given clock frequency, ARR determines the PWM period. Assume the counting mode is up-counting, and the PWM frequency is 100 Hz. Thus, the PWM period should be





When ARR is kept at a fixed value, CCR controls the duty cycle. Assume timer 1 uses PWM mode 1, in which the reference output (OCRE F) is high for up-counting if the timer counter is less than CCR, and otherwise, OCREF is low. CCR controls the duty cycle of PWM output if ARR is fixed.



**4.1 Program to Generate a PWM to Control the Brightness of a LED at 100Hz:**

#include "stm32f4xx.h"

void TIM2\_Init(void);

void configureLED(void);

void TIM4\_Init (void)

{

//1. Enable TIM@ and GPIO clock

RCC->APB1ENR |= (4UL<<0); // enable TIM4 clock

RCC->CFGR |= 0<<10; // set APB1 = 16 MHz

//2. Timer pre-scalar and period configuration

TIM4->CR1 &= ~(0x0010); //Set the mode to Count up

TIM4->PSC = 160-1; //Set the Prescalar

TIM4->ARR = 1000-1; //Set period (Auto reload) to 400

//TIM2->SR &= ~(0x0001); //Clear Update interrupt flag

}

void configureLED(void)

{

RCC->AHB1ENR |=(1UL<<3); //Enable GPIOD clock

GPIOD->MODER |= (2UL<<24); //Alternate function

GPIOD->AFR[1] |= (2UL<<16); //Timer4\_Channel1

}

void configurePWM(void)

{

TIM4->CCMR1 |=(6UL<<4); //PWM mode

//TIM4->CCMR1 |=(1UL<<3);

TIM4->CCER |=(0x1<<0);

}

int main ()

{

int i,brightness,stepSize;

brightness=1;

stepSize=1;

TIM4\_Init ();

configurePWM();

configureLED();

TIM4->CCR1=500; //Initial duty cycle 50%

TIM4->CR1 |= 1UL; //Start the timer

while(1)

{

if(brightness>=999)

brightness=1;

brightness+=stepSize;

TIM4->CCR1=brightness;

for(i=0;i<5000;i++);

}

}

**4.2 Program to Generate a Multichannel PWM Signals:**

#include "stm32f4xx.h"

void GPIO\_Config(void);

void TIM\_Config(void);

int main(void)

{

GPIO\_Config();

TIM\_Config();

//Start Timer2

TIM4->CR1 |= 1UL;

while(1)

{

}

}

void GPIO\_Config(void)

{

//GPIOD clock enable

RCC->AHB1ENR |= (1UL << 3);

//Set GPIOD pins 12 - 15 to alternate mode

GPIOD->MODER |= (2UL << 12\*2) | (2UL << 13\*2) | (2UL << 14\*2) | (2UL << 15\*2);

//Set alternate function to AF2 (TIM4)

GPIOD->AFR[1] |= (2UL << 4\*4) | (2UL << 5\*4) | (2UL << 6\*4) | (2UL << 7\*4);

}

void TIM\_Config(void)

{

// Basic timer configuration

RCC->APB1ENR |= (1UL << 2); //Enable TIM4 clock

TIM4->CR1 &= ~(0x0010); //Set the mode to Count up

TIM4->PSC = 160-1; //Set the Prescalar

TIM4->ARR = 1000-1; //Set period (Auto reload) 10ms duration of total pulse

//Channel 1 config

TIM4->CCMR1 |= (6UL << 4); //Set OutputCompare mode to PWM

TIM4->CCMR1 &= ~(3UL << 0); //CC to output

TIM4->CCER &= ~(1UL << 1); //Output Compare polarity to active high

TIM4->CCER |= (1UL << 0); //Capture compare output enable

//Set the Pulse width

TIM4->CCR1 = 100;//10% duty cycle

//Channel 2 config

TIM4->CCMR1 |= (6UL << (4+8)); //Set OutputCompare mode to PWM

TIM4->CCMR1 &= ~(3UL << 8); //CC to output

TIM4->CCER &= ~(1UL << (1+4)); //Output Compare polarity to active high

TIM4->CCER |= (1UL << 4); //Capture compare output enable

//Set the Pulse width

TIM4->CCR2 = 200;//20% duty cycle

//Channel 3 config

TIM4->CCMR2 |= (6UL << 4); //Set OutputCompare mode to PWM

TIM4->CCMR2 &= ~(3UL << 0); //CC to output

TIM4->CCER &= ~(1UL << 9); //Output Compare polarity to active high

TIM4->CCER |= (1UL << 8); //Capture compare output enable

//Set the Pulse width

TIM4->CCR3 = 300;//30% duty cycle

//Channel 4 config

TIM4->CCMR2 |= (6UL << (4+8)); //Set OutputCompare mode to PWM

TIM4->CCMR2 &= ~(3UL << 8); //CC to output

TIM4->CCER &= ~(1UL << 13); //Output Compare polarity to active high

TIM4->CCER |= (1UL << 12); //Capture compare output enable

//Set the Pulse width

TIM4->CCR4 = 400;//40% duty cycle

}

**5. Conclusion:**

In this experiment, generation of PWM signals were learnt, which can be applied to various applications, like control of DC motor.