Department of Computer Engineering
Faculty of Engineering, University of Peradeniya

CO321 Embedded Systems - 2019

Lab 4 - Pulse Width Modulation (PWM)

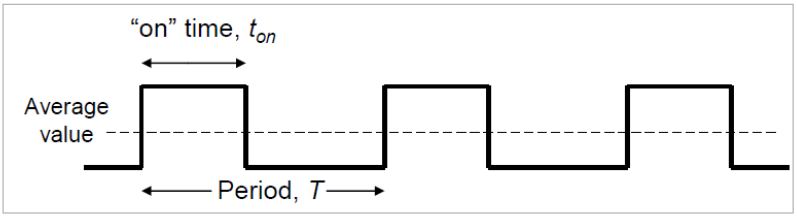
AVR PWM Programming in C Language

Pulse Width Modulation (PWM)

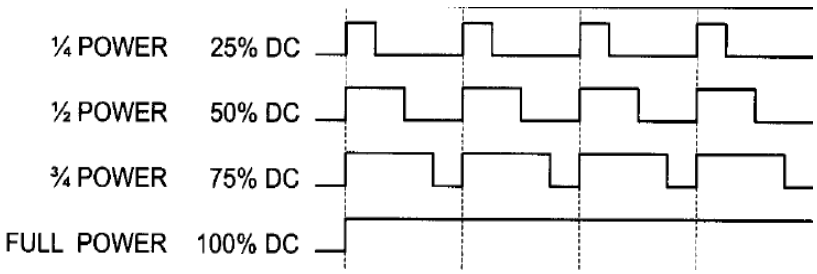
Pulse width modulation (PWM) is a powerful technique for controlling analog circuits with a microprocessor's digital output. It is a simple method of using a rectangular digital waveform to control an analog variable. PWM control is used in a variety of applications, ranging from communications to automatic control.

Period *T* is normally kept constant. Pulse width, or "on" time, is varied to get the desired output. The Duty Cycle is the proportion of time that the pulse is 'ON' or 'high', and is expressed as a percentage:

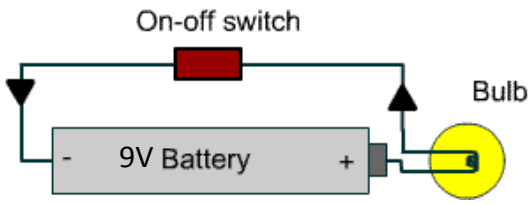
Duty cycle = (pulse ON time / pulse period *T*) * 100%



Whatever duty cycle a PWM stream has, there is an average output voltage, as indicated by the dotted line. If the "on" time is small, the average output is low; if the "on" time is large, the average output is high. **By controlling the duty cycle, we control this average output voltage.**



The figure below shows a simple circuit that could be driven using PWM. In the figure, a 9V battery power an incandescent light bulb.



If we closed the switch connecting the battery and lamp for 50ms, the bulb would receive 9V during that interval and light up. If we then opened the switch for the next 50ms, the bulb would receive 0V. If we repeat this cycle 10 times (a full second), the bulb would appear to be lit as though it were connected to a 4.5V battery (50% of 9V). We say that the duty cycle is 50% and the modulating frequency is 10Hz.

Most loads require a much higher modulating frequency than 10Hz. Imagine that our bulb was switched on for five seconds, then off for five seconds, then on again. The duty cycle would still be 50%, but the bulb would appear brightly lit for the first five seconds and off for the next. In order for the bulb to see an average voltage of 4.5 volts, the cycle period must be short relative to the load's response time to a change with the switch state. To achieve the desired effect of a dimmer-lamp (but always lit), it is necessary to increase the modulating frequency. The same is true in other applications of PWM. Common modulating frequencies range from 1kHz to 200kHz.

PWM with Timer0 in ATmega 328P

The ATmega328P comes with 3 timers as we discussed. All these timers can be used for PWM. They support two PWM modes; namely, **Fast PWM** and **Phase Correct PWM**. In this lab we will focus on the Fast PWM mode with `TIMER0`. The advantage of using the built-in PWM feature of the AVR (rather than creating PWM in software) is that it gives us the option of pre-programming the period and duty cycle, therefore relieving the CPU to do other important things.

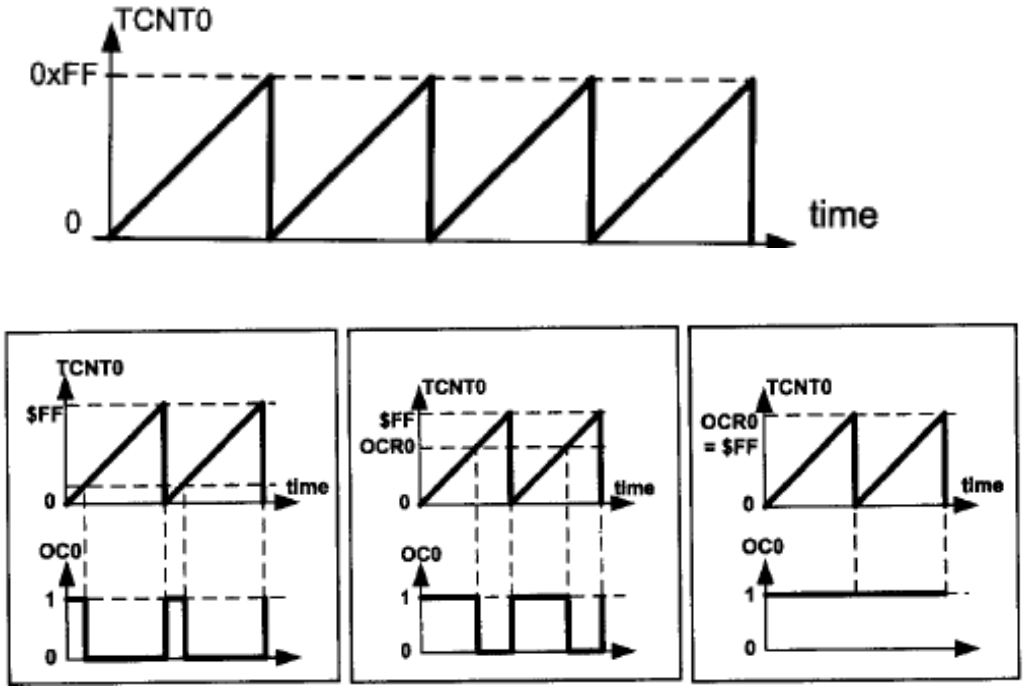
ATmega328 `TIMER0` has 2 outputs, `OC0A` (Timer/Counter 0 Output Compare Match A Output) and `OC0B` (Timer/Counter 0 Output Compare Match B Output). Since both of these outputs run off the same timer and waveform generators both `OC0A` and `OC0B` are synchronized. We will use `OC0A` in this lab.

Registers involved

- `TCNT0` – Timer/Counter Register
- `OCR0A` – Output Compare Register A
- `TCCR0A` – Timer/Counter Control Register A
- `TCCR0B` – Timer/Counter Control Register B

In the fast PWM, the counter (`TCNT0`) counts like it does in the Normal mode. After the time is started, it starts to count up. It counts up until it reaches its limit of `0xFF`. Then it rolls over from `0xFF` to `0x00`.

In the below figure, you can see the reaction of the waveform generator when compares match occurs while the timer is in the fast PWM mode.



In the figure `OC0` is the PWM output pin (Actually, this should be `OC0A` for our microcontroller). `OCR0` is the Output Compare Register (For our microcontroller this should be `OCR0A`). At the beginning (count 0) PWM pin is set to 1. When the counter value in `TCNT0` reaches `OCR0` the PWM pin is set to low.

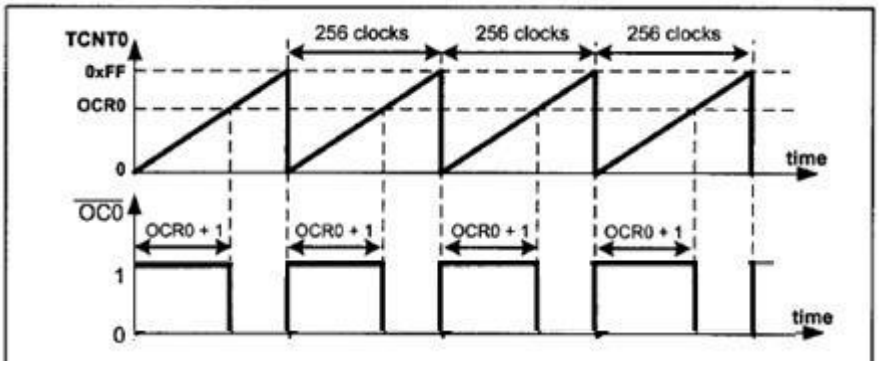
Frequency of the generated wave in Fast PWM mode

The timer counts from 0 to `TOP` and then rolls over. So, the frequency of the generated wave is 1/256 of the frequency of timer clock. The frequency of the timer clock can be selected using the prescaler. So, in 8-bit timers the frequency of the generated wave can be calculated as follows (N is determined by the prescaler.)

$$F_{generated\ wave} = \frac{F_{timer\ clock}}{256}$$

$$F_{timer\ clock} = \frac{F_{oscillator}}{N}$$

$$\Rightarrow F_{generated\ wave} = \frac{F_{oscillator}}{256 \times N}$$



Duty Cycle of the generated wave in Fast PWM mode

The duty cycle can be determined using the `OCR0A` register. The bigger `OCR0` value results in a bigger duty cycle; When `OCR0`=255, the `OC0` is high 256 clocks out of 256 clocks. The Duty cycle can be calculated using the following formula.

$$\text{Duty Cycle} = \frac{OCR0 + 1}{256} \times 100$$

Note that in ATmega 328P this waveform is called the non-inverted mode. There is another mode called inverted mode where the PWM waveform would be inverted. In this lab we will only deal with the non-inverted mode.

Steps in programming

- Configure the corresponding PWM pin as an output pin.
- Find the counts required for the given duty cycle and write it to `OCR0A`.
- Configure the PWM settings.
  - ➢ Select the timer mode. In lab 4 we used the normal mode. But today we should use Fast PWM mode. (See `WGM01:0` bits in `TCCR0A` and `WGM02` in `TCCR0B`)
  - ➢ We are using a non inverting mode. So select that. (See `COM0A1` and `COM0A0` bits in `TCCR0A`. Use the correct table for Fast PWM since we are using that)
- Select the clock source. (See `CS02:0` in `TCC0B`) You should use the correct prescaler to match the required period for the generated PWM. Note that when you select the clock source, the timer starts running and hence the PWM waveform should appear on the corresponding PWM pin.
- Now PWM is running. You can use the CPU for something else. If not, remember to put a `while(1)` loop to prevent the program from ending.

Exercises

1. Write a program that generates a PWM waveform with a frequency of approximately 976.56Hz and a duty cycle of 50%. Connect one LED to the PWM output pin and another to the 5V output pin (LEDs and the resistors you use should be identical) and check if the intensity of the bulb connected to PWM is half of the other.

2. Now extend the program in part 1 above, to **slowly** fade an LED from off->on and then on->off repeatedly.

3. The frequency of the sound made by a piezo buzzer depends on the frequency of the PWM waveform applied to it. In this case the *duty cycle* affects the *volume* of the sound. Write a program that repeatedly generates 4 separate sound frequencies in the increasing order one after the other. (each frequency should be heard for a notable time before switching to the next frequency, so that it can be heard properly). Keep in mind that humans can hear frequencies in the range 20Hz - 20,000Hz only.