AVR PWM Pulse Width Modulation – Tutorial #12

T.K. HAREENDRAN

AVR tutorial

Share this:

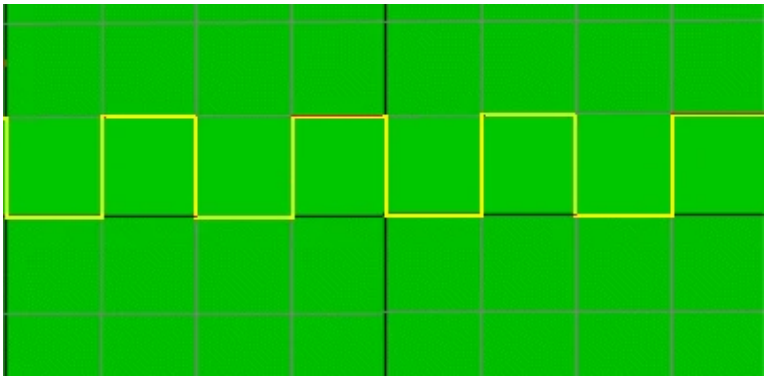
Share More

Pulse Width Modulation (PWM) is a comparatively recent power switching technique for providing intermediate amounts of electrical power between fully on and fully off levels. Usually, digital pulses have same on and off time period, but in some situations we need the digital pulse to have more/less on time/offtime. In PWM technique, we create digital pulses with unequal amount of on and off state to get required intermediate voltage values.

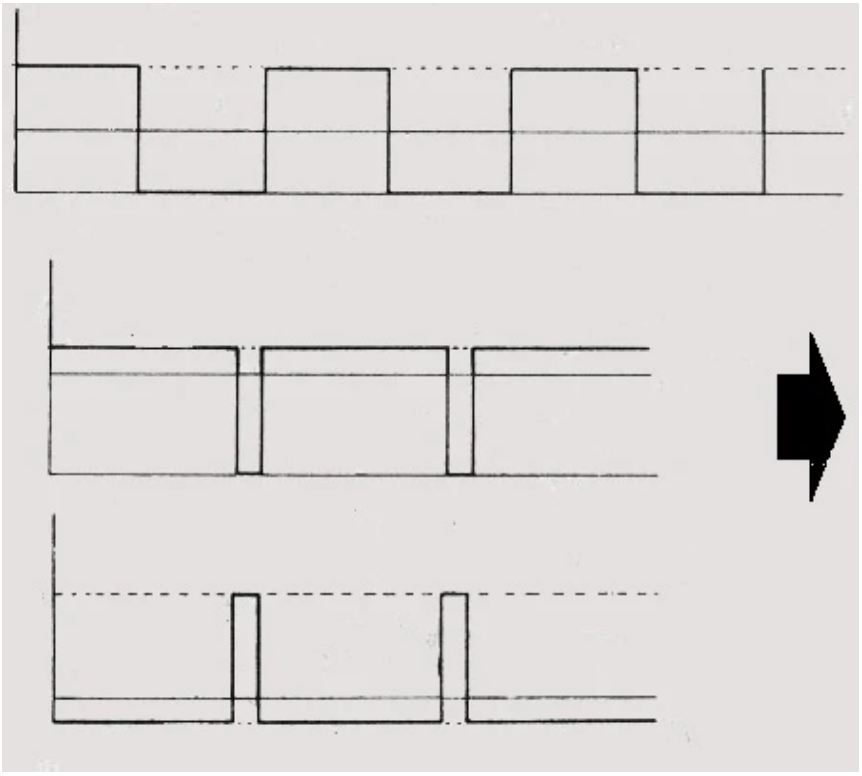
What is duty cycle? Duty cycle is defined by the percentage of high voltage duration in a complete digital pulse. It can be calculated by:

% of Duty cycle = T on /T (total time) x 100

If the duty cycle is 50%, then it will remain on for exact half the duration of the total time period of the digital pulse.



Note that duty cycle, duty factor and pulse repetition rate are parameters of all rectangular waves, and are very important in digital circuitry. Duty cycle is the ratio of pulse width to the signal period expressed as a percentage. Duty factor is the same thing as duty cycle except it is expressed as a decimal,and not as a percentage. If duty cycle is 50%, the duty factor is 0.5. A repetition rate describes how often a pulse train occurs in a second, and is often used to describe some waveforms.



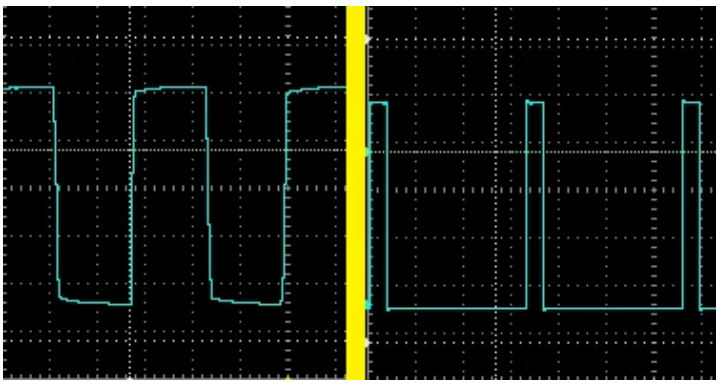
To understand the PWM principle, look at the example. Here a square wave of equal mark space ratio is available as an output. If this squarewave is fed to the base terminal of a transistor in common-emitter configuration, the transistor is in saturation or cut off for equal periods, so the average voltage at the collector will be half the supply voltage. If the mark to space ratio is increased then the average voltage will rise. And the output voltage will fall when the mark to space ratio is decreased.

To recap, a pulse width modulator is basically a square wave oscillator whose output mark/space ratio can be altered by an external voltage. For implementing PWM technique, a square wave with an easily adjustable mark space ratio is necessary.

PWM & AVR

For making PWM, AVR contains separate hardware! By using this, the CPU instructs the hardware to produce PWM of a particular duty cycle. The ATmega8 has 3 PWM outputs, 2 are located on timer/counter1 (16bit) and 1 is located on timer/counter2 (8bit). Timer/Counter2 is the simplest PWM device on the ATmega8. Timer/Counter2 is capable of running on 2 modes the Fast PWM mode and the Phase Corrected PWM mode; each of these modes can be inverted or non-inverted. Also note that there are three methods by which you can make PWM from AVR TIMER 1.

- Fast PWM
- Phase and Frequency Corrected PWM
- Phase Corrected PWM



Here is a sample ATmega8 code to setup TIMER 1 for a 4KHz, 10bit, Phase Corrected PWM at 16MHz Clock:

```
#include
int main(void)
{
   DDRB |= (1 << DDB1);
   // PB1 as output
   OCR1A = 0x01FF;
   // set PWM for 50% duty cycle at 10bit
   TCCR1A |= (1 << COM1A1);
   // set non-inverting mode
   TCCR1A |= (1 << WGM11) | (1 << WGM10);
   // set 10bit phase corrected PWM Mode
   TCCR1B |= (1 << CS11);
   // set prescaler to 8 and starts PWM
   while (1)
   {
   }
}
```

Yes, we watched that PWM can be generated from 16-bit Timer/Counter1 or 8-bit Timer/Counter2 . Here is an example code of 8-bit Timer2 Fast PWM Mode (8KHz) at 16MHz clock:

```
#include
int main(void)
{
   DDRB |= (1 << DDB3);
   // PB3 as output
   OCR2 = 128;
   // set PWM for 50% duty cycle
   TCCR2 |= (1 << COM21);
   // set non-inverting mode
   TCCR2 |= (1 << WGM21) | (1 << WGM20);
   // set fast PWM Mode
   TCCR2 |= (1 << CS21);
   // set prescaler to 8 and starts PWM
   while (1)
   {
   }
}
```

→ Part 13: **AVR Analog to Digital Conversion (ADC)**