27. Driving a DC motor using PWM

This example demonstrates the use of pulse width modulation (PWM) block within a timer to drive a DC motor. The motor is connected to the driver on the Base-board using two wires, its direction of rotation and rotation speed can be controlled by changing the duty cycle of the PWM signals.

27.1. The theory

A motor is a linear load – higher voltage U_{mot} over a motor results with faster rotation (providing that the loading of the motor stays the same). The current through a motor increases with the applied voltage and the loading (breaking force) of the motor. A microcontroller cannot provide sufficient voltage and current to drive a motor, therefore a driver must be used to increase the voltage and current levels provided by a microcontroller as required by the motor.

The power supply used to cover voltage and current requirements of the motor must provide a constant voltage U_{max} which is the maximum voltage the motor might need. The motor driver then reduces this voltage as directed by the microcontroller to U_{mot} in order to rotate the motor with the required speed. Since current is flowing through both motor and the driver, heat is dissipated (power wasted) on the driver. In order to avoid the heating of the driver a pulse width modulation is usually employed.

Consider the periodical signal *X* at Figure 27.1. It remains at 10 V for 100 ms, then returns to 0 V for 300 ms. It's average value is $\langle x \rangle = 10\,V\ 100\,ms/(100\,ms + 300\,ms) = 2.5\,V$. The signal with amplitude of 10 V and duty cycle of 25 % therefore has the average value of 2.5 V. If the duty cycle is changed the average value of this signal also changes. It is therefore possible to change the average voltage by changing the duty cycle of a periodic square signal.

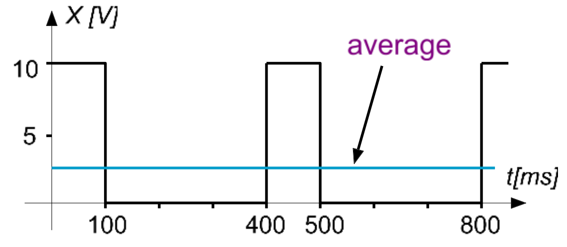


Figure 27.1: The average value of a periodical signal

Making a periodic signal by a microcontroller is simple: we only need to convert logic low to 0 V and logic high to, say, 10 V. This can be done by a switch, and switch does not dissipate power: when ON the voltage over the switch is zero and therefore the dissipation on the switch is null; when OFF the current is zero so the dissipation is again null. What we need is a digitally controlled (by signal X_{SW}) switch as in Figure 27.2. The diode must be added since the current through the motor cannot stop when the switch is OFF: when switch is ON the current flows from the power supply through the switch and motor, and when

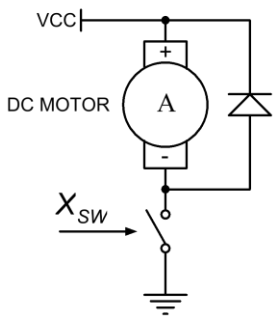
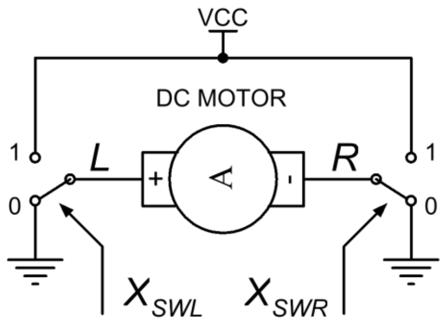


Figure 27.2: Basic connection of a motor using PWM driving

the switch is OFF the voltage induced in the motor due to its rotation forces approximately the same current to flow through the diode.

This way we can run a motor with the required rotational speed without dissipating power on a driver. The rotational speed of the motor can be adjusted by changing the duty cycle of the periodic control signal X_{sw} for the switch. Motor stops when duty cyle is 0%, and runs faster with higher duty cycle. The same technique can be used for any load that requires higher voltages and currents: light bulbs, heaters, etc. The only prerequisite is that the period of the control signal X_{sw} is short compared to the response time of the load. For motor a period of a millisecond would probably suffice, but this would put the driving signal into audible range at about 1 kHz and we would hear the sound from mechanics of the motor; it is better to use frequencies out of audible range, say 20 kHz. On the other hand higher frequencies put more stress on switch, so it might not be good to run the switch at 1 MHz.

The above method is applicable when we need to push the current to the load in one direction only (rotate a motor in one direction, run a heater, incandescent lamp, etc.). However, when we need to push the current through the motor (or other load) in one or the other direction to cause the required rotation speed and one of two rotation directions a different approach is necessary. We shall use a bridge configuration of switches as depicted in Figure 27.3. Two control signals for switches are needed, these are X_{SWL} *and* X_{SWR}*.* There are four possible steady value combinations of the two control signals as given in table at the same figure, where the voltage between nodes *L* and *R* (over the load - motor) is also given. Note that a voltage is present over the load for two value combinations of the driving signal, these are “10” and “01”. Other two combinations “00” and “11” make a short over the load (motor), and allow the motor to push the current in either direction by itself; the diode from Figure 27.2 is not needed (this is true when you use a driver circuit where four diodes are already incorporated, as in the driver at the BaseBoard). The voltage over the motor can have either polarity or can be zero.



X_{SWL}	X_{SWR}	L	R	L-R
0	0	0	0	0
0	1	0	VCC	-VCC
1	0	VCC	0	VCC
1	1	VCC	VCC	0

Figure 27.3: Bridge configuration is used to apply driving in both directions

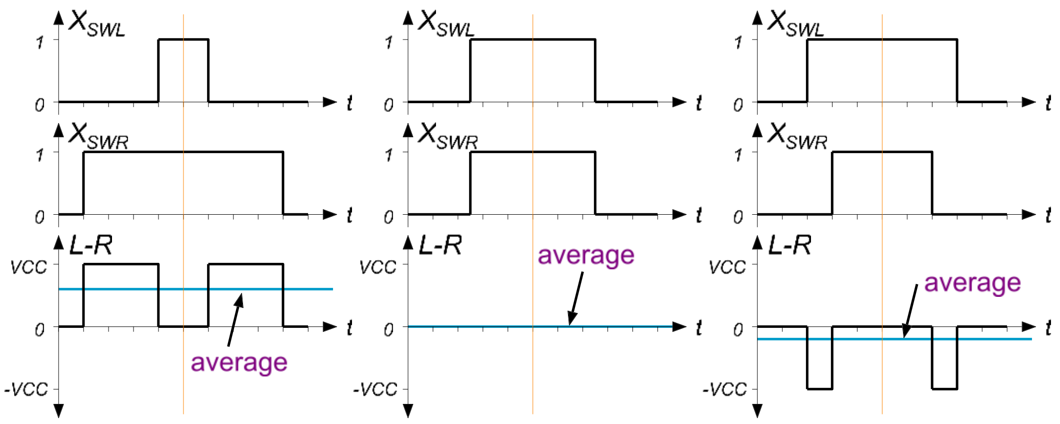


Figure 27.4: Three examples for PWM settings of driving signals

Consider now the implementation of the PWM technique to control signals X_{SWL} and X_{SWR}. Figure 27.4 shows three diverse situations for three different PWM settings (only one period of the otherwise periodic signals X_{SWL} *and* X_{SWR} is shown). In all cases the signals are "center aligned", they are therefore symmetrical to the orange line in the middle of the period.

- The middle diagram shows the situation where both X_{SWL} and X_{SWR} have the same duty cycle of 50 % and. The resulting average voltage over the load is zero, since both switches change simultaneously to the same value.
- The left diagrams show the situation where the X_{SWL} duty cycle is reduced below 50 % (to 20 %), and the X_{SWR} duty cycle is increased for the same amount above 50 % (to 80 %). The resulting average voltage over the load is positive and equal to 60 % of the VCC (2 times 30 %).
- The right diagrams show the situation where X_{SWL} duty cycle is increased above 50 % (to 60 %), and the X_{SWR} duty cycle is reduced for the same amount below 50 % (to 40 %). The resulting average voltage over the load is negative and equal to -20 % of the VCC (-2 times 10 %).

It is therefore possible to set the average voltage over the load to any positive or negative value within ±VCC using the principal schematic from Figure 27.3 and by setting the duty cycle of both driving signals symmetrically in opposite directions.

27.2. The Hardware

A microcontroller incorporates several counter/timers (TIM) with a capability to generate PWM signals. TIMs differ by the number of PWM outputs and flexibility (the programming options); TIMs 1, 8 and 2 to 5 have the richest selection of options. We will use TIM3 in this experiment since its outputs are readily available at the BaseBoard through a power driver consisting of L293D and connected as depicted in figure 27.3. The driver L293D consists of two H-bridges, one of them will be used in this experiment as depicted in Figure 27.5. We arbitrary select lines PC06 and PC07 to connect a motor through a driver chip; the chip must be turned on before use by setting PC11 (signal MOT_EN) high.

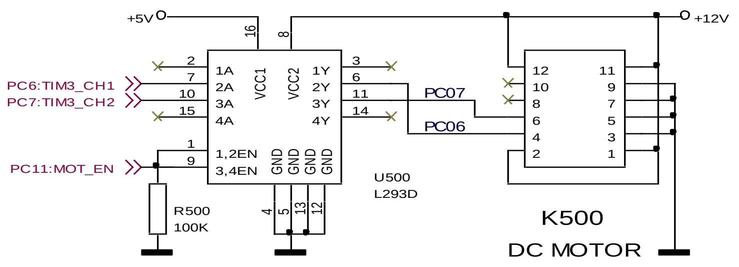


Figure 27.5: Bridge configuration for this experiment

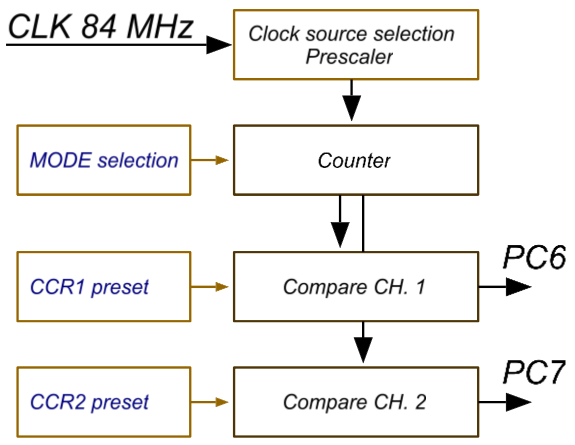


Figure 27.6: Simplified TIM3 block diagram

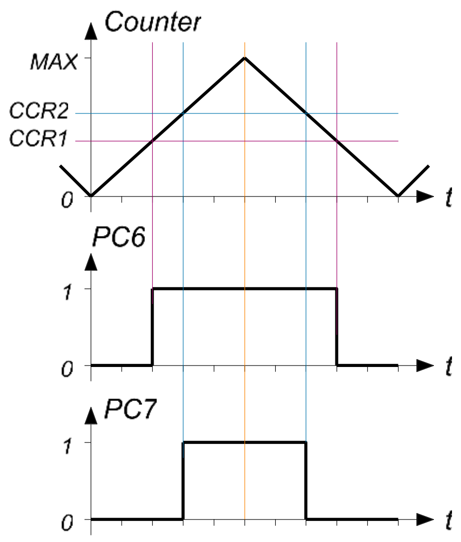


Figure 27.7: Timing diagram for "Compare" mode

A simplified TIM3 block diagram is given at Figure 27.6, while the complete block diagram is given in RM0090, chapter 17.2. The heart of TIM is a Counter which counts periodical clock pulses, appropriately divided clock of 84 MHz in this case. Some configuration is needed to connect the 84 MHz clock to the Counter input, and the output of the Counter is simply an increasing 16-bit number.

Consider now the timing diagram at Figure 27.7. The counter is configured to start counting from zero up to a predefined number MAX (say 1023 is the biggest number), then the counting direction reverses until zero is reached. At zero the counting direction reverses again… The content of the counter is shown in the diagram at Figure 27.7, top graph. The consecutive Counter output values form a triangular shape in the diagram, and the corresponding counting mode of the counter is called “up-down auto-reload counting” at STM.

This value is passed to two comparators in CH1 and CH2 (there are four of them available, but we need only two) having their threshold presets CCR1 and CCR2 set to 40% of 1023 = 409 and 60% of 1023 = 614. The two comparators in channels CH1 and CH2 are configured to respond high when the content of the Counter is bigger than the threshold, and therefore output two PWM signals with the duty cycle of 60% and 40% respectively. Changing the threshold levels of both comparators changes the corresponding duty cycles at comparator outputs. It is therefore possible to generate any symmetrical PWM signals by simply changing the threshold presets of the two comparators, and these outputs are routable to pins PC6 and PC7 of the microcontroller; of course the TIM has to be configured correctly first.

27.3. The Software

Two configuration for TIM3 is implemented in two steps. First pins to route TIM3 comparator outputs to pins of the microcontroller is described, then the actual configuration of TIM 3 to generate two PWM signals is presented. The chapter finishes with the description of the main part of the program.

a) TIM3 – configure pins

TIM3 compare outputs from Channel 1 and Channel 2 can be routed to port GPIOC, pins 6 and 7. Additionally, the driver chip has to be enabled by setting GPIOC, pin 11 high. All this can be configured with the following set of commands:

```
void GPIOCinit_TIM3_PWM (void)   {
GPIO_InitTypeDef        GPIO_InitStructure;
  RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOC, ENABLE);
  GPIO_InitStructure.GPIO_Pin   = GPIO_Pin_6  | GPIO_Pin_7 ;
  GPIO_InitStructure.GPIO_Mode  = GPIO_Mode_AF;
  GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_25MHz;
  GPIO_InitStructure.GPIO_PuPd  = GPIO_PuPd_DOWN;
  GPIO_Init(GPIOC, &GPIO_InitStructure);

  GPIO_InitStructure.GPIO_Pin   = GPIO_Pin_11;
  GPIO_InitStructure.GPIO_Mode  = GPIO_Mode_OUT;
  GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_25MHz;
  GPIO_InitStructure.GPIO_PuPd  = GPIO_PuPd_DOWN;
  GPIO_Init(GPIOC, &GPIO_InitStructure);
  GPIOC->BSRRL = BIT_11;

  GPIO_PinAFConfig(GPIOC, GPIO_PinSource6, GPIO_AF_TIM3);
  GPIO_PinAFConfig(GPIOC, GPIO_PinSource7, GPIO_AF_TIM3);
}
```

The data structure to pass the configuration parameters to the appropriate function is declared in line 2, and the port C is supplied with clock signal (turned ON) by the call in line 3. Next the data structure is stuffed with standard parameters to output a signal using an alternate function at pins 6 and 7, and in line 9 the function to pass these parameters to appropriate registers of port C in called. Next block stuffs the same data structure with new parameters for GPIOC, pin 11, and the appropriate function is called in line 16. The pin 11 is set high in line 17. Lastly in lines 19 and 20 the alternate functions for GPIOC, pins 6 and 7 are configured; these pins are to be used for TIM3.

b) **TIM3 – configure the TIM block for PWM at two outputs**

Timer configuration will be done in two steps: first the Counter, then both comparators. Finally, the TIM3 gets enabled. The listing is given bellow.

```
void TIM3init_PWM_CenterAligned (void)  {
TIM_TimeBaseInitTypeDef TIM_TimeBaseInitStructure;                                            //  2
TIM_OCInitTypeDef       TIM_OCInitStructure;                                                  //  3

  RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM3, ENABLE);                                        //  5
  TIM_TimeBaseInitStructure.TIM_CounterMode = TIM_CounterMode_CenterAligned3;                 //  6
  TIM_TimeBaseInitStructure.TIM_Period = 1024;                                                //  7
  TIM_TimeBaseInitStructure.TIM_Prescaler = 3;                                                //  8
  TIM_TimeBaseInit(TIM3, &TIM_TimeBaseInitStructure);                                         //  9

  TIM_OCInitStructure.TIM_OCMode = TIM_OCMode_PWM1;                                            // 11
  TIM_OCInitStructure.TIM_OCPolarity = TIM_OCPolarity_High;                                   // 12
  TIM_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable;                               // 13
  TIM_OCInitStructure.TIM_Pulse = 409;                                                        // 14
  TIM_OC1Init(TIM3, &TIM_OCInitStructure);                                                    // 15
  TIM_OC1PreloadConfig(TIM3, TIM_OCPreload_Enable);                                           // 16

  TIM_OCInitStructure.TIM_Pulse = 614;                                                        // 18
  TIM_OC2Init(TIM3, &TIM_OCInitStructure);                                                    // 19
  TIM_OC2PreloadConfig(TIM3, TIM_OCPreload_Enable);                                           // 20

  TIM_Cmd(TIM3, ENABLE);                                                                      // 22
}
```

The listing commences with two declarations of data structures in lines 2 and 3, one to hold configuration parameters for the Counter (TimeBaseInitStructure), and the other to hold the parameters for comparators (OCInitStructure).

Next the Counter gets configured. A clock for TIM3 is enabled in line 5, and the TIM3 is now ready to be configured. The data structure is stuffed with parameters in lines 6 to 8, and these parameters are utilized by the function call in line 9. Note thet the Counter is put into "CenterAligned3" mode, therefore it is set to count up then down. Other two possible modes are "CountUp" and "CountDown" (there are two more modes, that are not relevant for us here: "CenterAligned1" and "CenterAligned2"). The period of the counter is set to 1024, and the 84MHz clock is pre-divided by 4 (line 8) to achieve 24 MHZ clock for the counter and about 10 kHz repetion of the PWM pulses (24 MHz / 1024 / 2). Since there are two current pulses per period of the PWM signal (see Figure 27.4) this puts the driving of the DC motor out of the audible frequency range.

Next comes the configuration of the compratators. Again the data structure is stuffed with parameters and not all members of the structure as declared in CMSYS are utilized. Line 11 requires the PWM mode for the comparator, and line 12 necessitates output to jump high when the content of the Counter crosses the predefined value CCRx. In line 13 the output of a comparator is enabled, and line 14 sets the initial compare value to 409, which is good for the first comparator in Channel 1. These parameters are used by the function called in line 15, and the call in line 16 sets the same comparator in the simplest mode of operation, where the change of the compare value is allowed at all times. The second comparator in Channel 2 is to be configured using the same parameters, only the initial compare value needs to be changed. This is done in line 18, followed by similar calls for Channel 2.

Finally, the counter gets enabled in line 22.

c) Main program

The listing of the main program is presented below.

```
int main ()  {
int PWMratio = 512, switches;
  GPIOCinit_TIM3_PWM();
  LCD_init();                           // init LCD
  LCD_string("PWM linear motor", 0x00);    // display title string
  LCD_string("%", 0x48);    // display title string
  SWITCHinit ();
  TIM3init_PWM_CenterAligned();

  while (1)     {
    switches = GPIOE->IDR;                                                              // a
    if (switches & S370)  PWMratio++;  // S370                                          // b
    if (switches & S371)  PWMratio--;  // S371                                          // c
    if (PWMratio <    1) PWMratio = 1;                                                  // d
    if (PWMratio > 1023) PWMratio = 1023;                                               // e
    TIM3->CCR1 = PWMratio;                                                              // f
    TIM3->CCR2 = 1023 - PWMratio;                                                       // g
    LCD_sInt3DG((int)((PWMratio-512)/512.0*100),0x44,0x05);  // LCD, signed, 16 bits           // h
    for (int i = 0; i < 1000000; i++)   {};                                             // i
  };
}
```

The main program calls the appropriate functions described above to configure peripherals used (TIM3 pins and counting, LCD display, port GPIOE for switches). Once the configuration is done the main program enters an endless loop.

The initial duty cycle was set during the configuration, but a user should be able to change the spinning speed of the motor. To facilitate this switches are read and checked within the endless loop (line a). It a user presses either of the switches S370 or S371 the duty cycle is increased (line b) or decreased (line c). The duty cycle should stay within limits 0 to 1023, and this is checked in lines d and e. The correct duty cycle is then sent to both comparators in lines f and g; note that the duty cycle is used symmetrically for both comparators. The duty cycle is printed on the LCD and some time gets wasted before the next repetition of the endless loop.