PWM MODES IN ALL 3 TIMERS

PWM Timer modes & waveforms: *(pay attention to OCRnx registers controlling the 'pulse width' and corresponding OCnx pins (physical pin) at which output is observed)*

Timer0 – mode 3:

Period = 256 * (N/16Mhz)
Pulse width = (OCR0A+1) * (N/16Mhz)
Output on OC0A

OR

Period = 256 * (N/16Mhz)
Pulse width = (OCR0B +1) * (N/16Mhz)
Output on OC0B

Timer0 – mode 7:

Period = (OCR0A + 1) * (N/16Mhz)
Pulse width = (OCR0B+1) * (N/16Mhz)
Output on OC0B

Timer2 – mode 3:

Period = 256 * (N/16Mhz)
Pulse width = (OCR2A+1) * (N/16Mhz)
Output on OC2A

OR

Period = 256 * (N/16Mhz)
Pulse width = (OCR2B+1) * (N/16Mhz)
Output on OC2B

Timer2 – mode 7:

Period = (OCR2A+1) * (N/16Mhz)
Pulse width = (OCR2B+1) * (N/16Mhz)
Output on OC2B

PWM MODES IN ALL 3 TIMERS

Timer1 – mode 5:

Period = 256 * (N/16Mhz)
Pulse width = (OCR1A+1) * (N/16M
Output on OC1A

OR

Period = 256 * (N/16Mhz)
Pulse width = (OCR1B+1) * (N/16Mhz)
Output on OC1B

Timer1 – mode 6:

Period = 512 * (N/16Mhz)
Pulse width = (OCR1A+1) * (N/16M
Output on OC1A

OR

Period = 512 * (N/16Mhz)
Pulse width = (OCR1B+1) * (N/16Mhz)
Output on OC1B

Timer1 – mode 7:

Period = 1024 * (N/16Mhz)
Pulse width = (OCR1A+1) * (N/16Mhz)
Output on OC1A

OR

Period = 1024 * (N/16Mhz)
Pulse width = (OCR1B+1) * (N/16Mhz)
Output on OC1B

Timer1 – mode 14:

Period = ICR1 * (N/16Mhz)
Pulse width = (OCR1A+1) * (N/16Mhz)
Output on OC1A

OR

Period = ICR1 * (N/16Mhz)
Pulse width = (OCR1B+1) * (N/16Mhz)
Output on OC1B

PWM MODES IN ALL 3 TIMERS

Timer1 – mode 15:

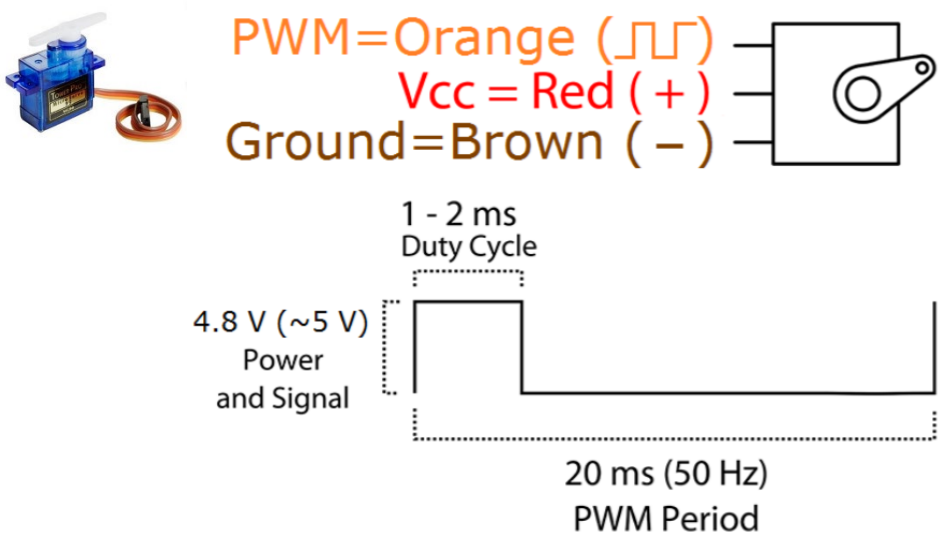
Period = (OCR1A+1) * (N/16Mhz)
Pulse width = (OCR1B+1) * (N/16Mhz)
Output on OC1B

Output Pins:

OC0A – PD6 : TCCR0A = 100000xx
OC0B – PD5 : TCCR0A = 001000xx
OC2A – PB3 : TCCR2A = 100000xx
OC2B – PD3 : TCCR2A = 001000xx
OC1A – PB1 : TCCR1A = 100000xx
OC1B – PB2 : TCCR1A = 001000xx

'xx' bit positions will vary based on mode configured (modes of PWM)

Servo motor datasheet



Position "0" (1.5 ms pulse) is middle, "90" (~2 ms pulse) is all the way to the right, "-90" (~1 ms pulse) is all the way to the left.

NOTE: *A servo motor needs a 20 ms period. Timer0 and Timer2 can only produce PWM with a max period of ~16ms (255*(1024/16MHz)). Use timer1 in modes 14 or 15 when controlling a servo motor.*

PWM MODES IN ALL 3 TIMERS

Setting PWM modes:

Setting up the timer(s) with proper mode is done using 'WGMxn' bits found in TCCRxA and TCCRxB registers (Both registers need to set-up, to properly configure the timer)

Timer0 and Timer2:

15.9.1 TCCR0A – Timer/Counter Control Register A

Bit	7	6	5	4	3	2	1	0	
0x24 (0x44)	COM0A1	COM0A0	COM0B1	COM0B0	–	–	WGM01	WGM00	TCCR0A
Read/Write	R/W	R/W	R/W	R/W	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

15.9.2 TCCR0B – Timer/Counter Control Register B

Bit	7	6	5	4	3	2	1	0	
0x25 (0x45)	FOC0A	FOC0B	–	–	WGM02	CS02	C	CS00	TCCR0B
Read/Write	W	W	R	R	R/W	R/W		W	
Initial Value	0	0	0	0	0	0			

on

Mode	WGM02	WGM01	WGM00	nter ation	TOP	Upd OC	TOV Flag Set on(1)(2)
0	0	0	0	Normal	0xFF	Imm	MAX
1	0	0	1	PWM, Phase Correct	0xFF	T	BOTTOM
2	0	1	0	CTC	OCRA	Imm	MAX
3	0	1	1	Fast PWM	0xFF	BO	MAX
4	1	0	0	Reserved			–
5	1	0	1	PWM, Correct			BOTTOM
6	1	1	0	Reserved	–	–	–
7	1	1	1	Fast PWM	OCRA	BOTTOM	TOP

PWM MODES IN ALL 3 TIMERS

Timer1:

16.11.1 TCCR1A – Timer/Counter1 Control Register A

Bit	7	6	5	4	3	2	1	0	
(0x80)	COM1A1	COM1A0	COM1B1	COM1B0	–	–	WGM11	WGM10	TCCR1A
Read/Write	R/W	R/W	R/W	R/W	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

16.11.2 TCCR1B – Timer/Counter1 Control Register B

Bit	7	6	5	4	3	2	1	0	
(0x81)	ICNC1	ICES1	–	WGM13	WGM12	CS12	CS11	CS10	TCCR1B
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Table

Mode	WGM13	WGM12 (CTC1)	WGM11 (PWM11)	WGM10 (PWM10)	Timer/Counter Mode of Operation	TOP	Update of OCR1x at	TOV1 Flag Set on
0	0	0	0	0	Normal	0xFFFF	Immediate	MAX
1	0	0	0	1	PWM, Phase Correct, 8-bit	0x00FF	TOP	BOTTOM
2	0	0	1	0	PWM, Phase Correct, 9-bit	0x01FF	TOP	BOTTOM
3	0	0	1	1	PWM, Phase Correct, 10-bit	0x03FF	TOP	BOTTOM
4	0	1	0	0	CTC	OCR1A	Immediate	MAX
5	0	1	0	1	Fast PWM, 8-bit	0x00FF	TOP	TOP
6	0	1	1	0	Fast PWM, 9-bit	0x01FF	TOP	TOP
7	0	1	1	1	Fast PWM, 10-bit	0x03FF	TOP	TOP
8	1	0	0	0	PWM, Phase and Frequency Correct	ICR1	BOTTOM	BOTTOM
9	1	0	0	1	PWM, Phase and Frequency Correct	OCR1A	BOTTOM	BOTTOM
10	1	0	1	0	PWM, Phase Correct	ICR1	TOP	BOTTOM
11	1	0	1	1	PWM, Phase Correct	OCR1A	TOP	BOTTOM
12	1	1	0	0	CTC	ICR1	Immediate	MAX
13	1	1	0	1	(Reserved)	–	–	–
14	1	1	1	0	Fast PWM	ICR1	BOTTOM	TOP
15	1	1	1	1	Fast PWM	OCR1A	BOTTOM	TOP

Setting up a Timer: step-by-step guide

Before you get to program a timer module, based on your requirements (projects/HW/Lab activity), you would need the following information before hand:

1. Which Timer should I use?
2. What mode do I want to run the timer?
3. Which prescaler value will enable me to generate desired signal?
4. For the provided objective, what do I want my 'Compare Output (OCnx)' pins to do?

5. Which method am I using to check the status of my timer – Polling or Interrupts?
    a. If using Interrupts, be sure to set corresponding 'Interrupt Enable' bit to '1'
    b. If using Polling, make sure to clear corresponding 'flag', by writing 'logic 1' to its bit position.
6. If using Interrupts, what does my ISR do?
    a. ISRs are functions, that will never be called in 'void loop()' section of code
    b. ISRs are a function that do not take in arguments, nor return arguments
    c. NO 'Serial.println()', NO 'delay()' inside your ISR – remember 'Get IN & Get OUT'
7. Based on the mode selected, what 'count' values (to be loaded onto OCRnx registers) will provide me with the desired signal?
8. When do I start my timer (by loading appropriate 'count' value to corresponding register - based on the mode selected), that would result in obtaining desired signal/timing?

Programming a Timer: (follows the previous step-by-step guide)
Now that we have all the necessary information, let see how to translate the information into values to be loaded onto corresponding registers:

a. Once you have decided on which timer (0/1/2) to use, your next step is to set-up the timer for appropriate mode
b. Setting mode: Across all timers, the bit fileds '{WGMxn}' control the operational mode selection. For 8-bit Timers (0/2), mode selection is controlled by {WGMx2, WGMx1, WGMx0} bits found in TCCRxA and TCCRxB register. {WGM13, WGM12, WGM11, WGM10} bits found in TCCR1A and TCCR1B registers control the mode selection for Timer 1.
    i. Record the appropriate values to be set at 'WGMxn' bits to enable the desired mode (record these values on a paper or a document)
c. Based on the prescaler value selected, bits {CSx2, CSx1, CSx0} will need to be set to appropriate values. Bits {CSx2, CSx1, CSx0} are part of TCCRxB register across all Timers (0/1/2).
    i. Record the values to be set at 'CSxn' bits for appropriate prescaler value.
d. Based on your requirement, you would need to use {COMxA1, COMxA0, COMxB1, COMxB0} bits found in TCCRxA register, to configure how do you want the output signal to be represented (HIGH during 'Duty Cycle' or LOW during 'Duty Cycle') on Channels A/B of OCnx pins
    i. You would only set the bits for the channel that you expect the output from (Channel A or Channel B or both channels)
    ii. Record the values to be set at {COMxA1, COMxA0, COMxB1, COMxB0} bits

NOTE: If you have stepped through the above steps successfully, Kudos! – You have now set-up a timer for your application (but it hasn't started yet). Setting the mode, prescaler value, and signal representation at output pin, is done in TCCRxA and TCCRxB register for all timers (0/1/2)

e. If you have decided to use Interrupts for your application, you would need to enable the corresponding mode's 'Interrupt Enable' bit found in 'TIMSKx' register.
    i. Based on which channel (A/B) you expect the output, its corresponding 'Interrupt Enable' bit – 'OCIExA/OCIExB' should be set to '1'
    ii. Before you enable the interrupts, make sure to disable global interrupts, and re-enable global interrupts immediately.

f. Writing ISR: ISRs are functions that gets automatically executed when an event (internal/external) triggers the corresponding interrupt. Though ISRs are functions, they do not take 'input arguments' nor 'return arguments'.
   i. Variables updated from within the ISR, and accessed in 'void loop()' will need to be declared as a global variable – IS THIS SUFFICIENT?
   ii. Your code within ISR should be an absolute minimum code that does a very specific task (turn OFF LEDs, read SENSOR data, update a flag variable). Do not have print statements and delay() functions within your ISR

NOTE: At this point, your timer is ready to be started. Also at this point you would have recorded the values to be set at each individual bit positions (for mode, prescaler, output representation, interrupt enable) for TCCRxA, TCCRxB, and TIMSKx (if needed).

g. Obtaining 'counts': For PWM mode operation of timers, you would need the count values that dictate the 'pulse width' (Duty Cycle) of your output PWM signal. In certain modes (mode-7 for timers 0 and 2, and mode-14,15 for timer 1) you would need the count value for your 'desired period' (PWM period).
   i. Using the formula (ref. timing diagrams above) to obtain the 'count' values to achieve required 'pulse width' and 'period'
   ii. If using mode-3 of 8-bit timers, mode-5,6,7 of 16-bit timers – record the value of 'counts' for 'pulse width'
   iii. If using mode-7 of 8-bit timers, or mode-14,15 of 16-bit timers – record the value of 'counts' for both 'pulse width' and 'period'
h. Starting Timers: At this point in programming, your timer is ready to be started. Loading the 'count' value onto corresponding register will start the timer module's function
   i. To modify your output pulse's duty cycle during run-time, you can re-load the 'OCRxA/B' registers with appropriate count values corresponding to your desired 'pulse width'
   ii. If you choose to modify the 'pulse width' – it is a best practice to disable the timer interrupts (if enabled) before loading the new count values. Remember to enable the timer interrupts again.

Coding with Datasheet:

If you had followed the steps described above, you will have all the bit positions/registers that needs to be updated – recorded with its appropriate value to be set. You will have identified the values for:

1. Mode selection Bits (WGMxx)
2. Prescaler section bits (CSxx)
3. Configuration for Compare Output Pins (COMxA[1:0]/COMxB[1:0])
4. Interrupt Enable bits (if required)
5. 'count' values for 'pulse width'/ 'period'

Armed with these information, your next stop should be your datasheet. Open the datasheet and head to Timers section. Locate the timer you have selected, and land at 'Register Description' section (ref. timer datasheet handouts provided).

a. Start with the first register (TCCRxA). Looking at the datasheet, start populating the bit values (on a scratch sheet) in the appropriate location. Once you have determined the values in all 8 bit positions, load the binary bit pattern to TCCRxA register

PWM MODES IN ALL 3 TIMERS

	7	6	5	4	3	2	1	0	
	COM2A1	COM2A0	COM2B1	COM2B0	–	–	WGM21	WGM20	TCCR2A
	R/W	R/W	R/W	R/W	R	R	R/W	R/W	
values->	0	0	1	0	0	0	1	1	

b. Follow the same process for TCCRxB register. Populate the bit fields with appropriate values and load the binary value to the register

	7	6	5	4	3	2	1	0	
	FOC2A	FOC2B	–	–	WGM22	CS22	CS21	CS20	TCCR2B
	W	W	R	R	R/W	R/W	R/W	R/W	
values->	0	0	0	0	1	0	1	0	

c. Enable interrupts (if necessary). In your datasheet locate the 'TIMSK' register. Populate the bit values for individual field, and load the binary value to the register

	7	6	5	4	3	2	1	0	
	–	–	–	–	–	OCIE2B	OCIE2A	TOIE2	TIMSK2
	R	R	R	R	R	R/W	R/W	R/W	
values->	0	0	0	0	0	1	0	0	

d. Depending on the mode selected, load the determined 'count' values for 'pulse width'/ 'period' into appropriate registers (ref. timing diagram above). This should start your timer.

	7	6	5	4	3	2	1	0	
	OCR2A[7:0]								OCR2A
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
	7	6	5	4	3	2	1	0	
	OCR2B[7:0]								OCR2B
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	

NOTE:

- If using the 'OCnx' pins to generate signal, you should set-up the pin as 'OUTPUT'

PWM MODES IN ALL 3 TIMERS

```
void setup() {
    // SET {WGM21,WGM20} to '1' for mode 7 -  fast PWM
    //SET {COM2B1,COM2B0} to {1,0} for non-inverting PWM signal
    TCCR2A = 0b00100011;
    //SET {CS22,CS21,CS20} to {0,1,0} for prescaler value '8'
    //SET {WGM22} bit to '1' for mode 7 – fast PWM
    TCCR2B = 0b00001010;
    //Load 'OCR2A' with value for total period (0.1 msec) – 'T'
    OCR2A = 200;
    //Load 'OCR2B' with value for 'Ton' period (0.065 msec)
    OCR2B  = 130;
    //Setup pin 'OC2B' as OUTPUT
    DDRD |= 0b00001000;
}
void loop() {
 }
```