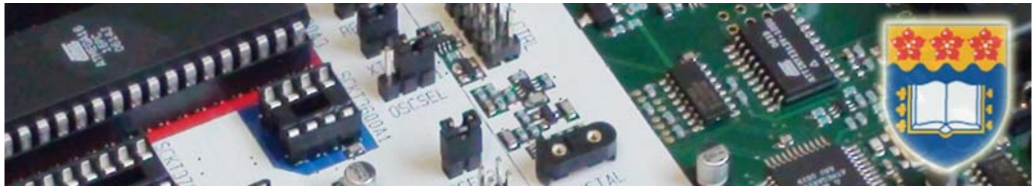


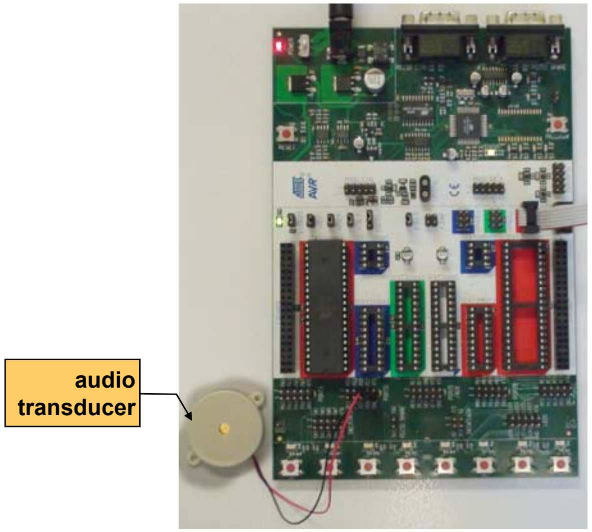
ECTE333
Lecture 10 - Pulse Width Modulator

School of Electrical, Computer and Telecommunications Engineering
University of Wollongong
Australia

ECTE333's schedule

Week	Lecture (2h)	Tutorial (1h)	Lab (2h)
1	L7: C programming for the ATMEL AVR		
2		Tutorial 7	Lab 7
3	L8: Serial communication		
4		Tutorial 8	Lab 8
5	L9: Timers		
6		Tutorial 9	Lab 9
7	L10: Pulse width modulator		
8		Tutorial 10	Lab 10
9	L11: Analogue-to-digital converter		
10		Tutorial 11	Lab 11
11	L12: Revision lecture		
12			Lab 12
13	L13: Self-study guide (no lecture)		
Final exam (25%), Practical exam (20%), Labs (5%)			

Lecture 10's demo: Generating music using PWM



Demo developed by John Mu and Lam Phung.
Video: [avr]/ecte333/lab10_task3.mp4

Lecture 10's sequence

10.1 Introduction

10.2 Output Compare Unit of Timer 1

10.3 Example application of PWM

[Lab10: Task 1 | Task 2 | Task 3]

10.1 Introduction

- In Lecture 9, we learnt two features of a timer:
  - overflow interrupt, and
  - input capture.

- **Overflow interrupt:**
  - triggered when timer reaches its top limit;
  - for measuring time that is longer than one timer cycle.
  - for finding the elapse time, creating a time delay, etc.

- **Input capture:**
  - an interrupt is triggered when there's a change in pin ICP1.
  - value of Timer 1 is automatically stored in register ICR1.
  - for finding period, frequency, pulse width of a signal.

Output Compare

- In this lecture, we'll study another important functionality of a timer: output compare.

- **Output compare** allows **custom processing** to be done when the timer reaches a **preset target value**.

- Examples of custom processing:
  - clearing timer,
  - changing values of dedicated pins,
  - triggering an interrupt.

- Output compare can be used to
  - generate signals of various shapes,
  - perform actions (e.g. ADC) at specific time instants.

An analogy with ECTE333 schedule

target timer value

custom processing

Week	Lecture (2h)	Tutorial (1h)	Lab (2h)
1	L7: C programming for the ATMEL AVR		
2		Tutorial 7	Lab 7
3	L8: Serial communication		
4		Tutorial 8	Lab 8
5	L9: Timers		
6		Tutorial 9	Lab 9
7	L10: Pulse width modulator		
8		Tutorial 10	Lab 10
9	L11: Analogue-to-digital converter		
10		Tutorial 11	Lab 11
11	L12: Revision lecture		
12			Lab 12
13	L13: Self-study guide (no lecture)		
	Final exam (25%), Practical exam (20%), Labs (5%)		

Output Compare: Common elements

- **Output compare registers**: To store the target timer values.

- **Output compare pins**: These dedicated pins can be automatically changed (set, reset, toggled) when there is an output compare match.

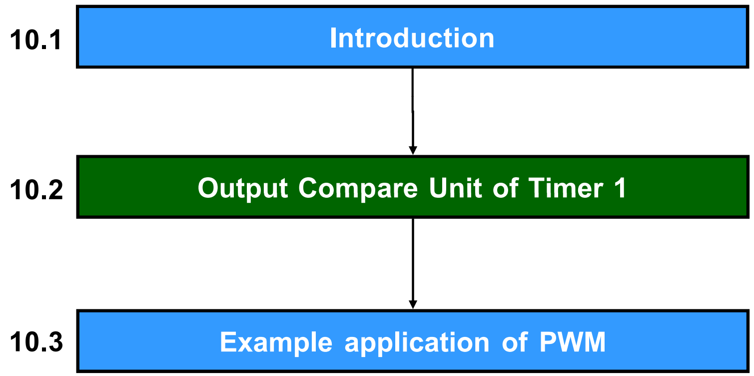
- **Configuration registers**: To configure the operations of timer.

- **Output compare interrupt**: ISR contains code for custom processing on an output compare match.

AVR Demo: Segway

Atmega16, gyroscope, accelerometer, ADC, PWM motors
(by Aaron Ticehurst, UOW).

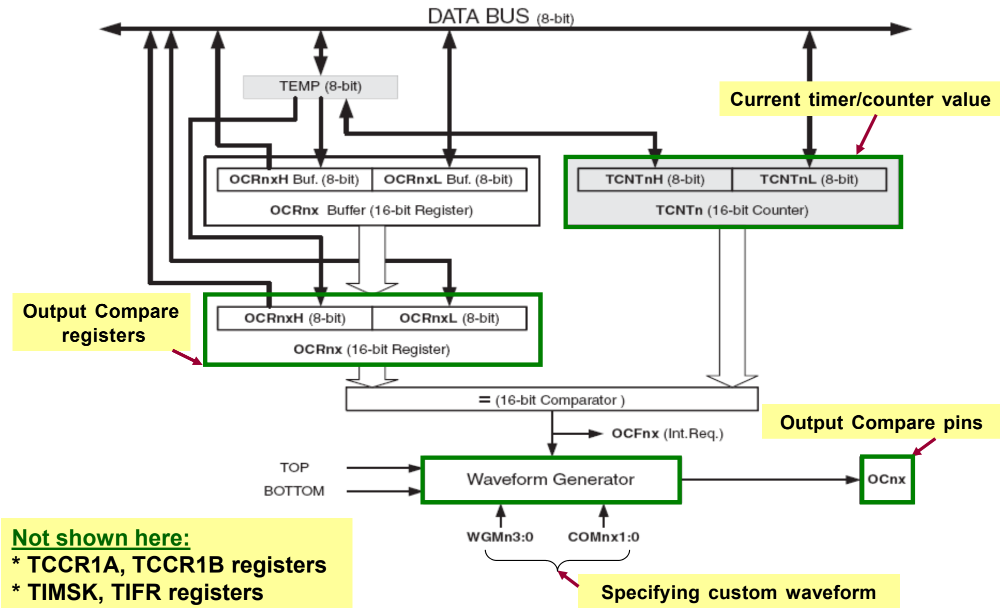
Lecture 10's sequence



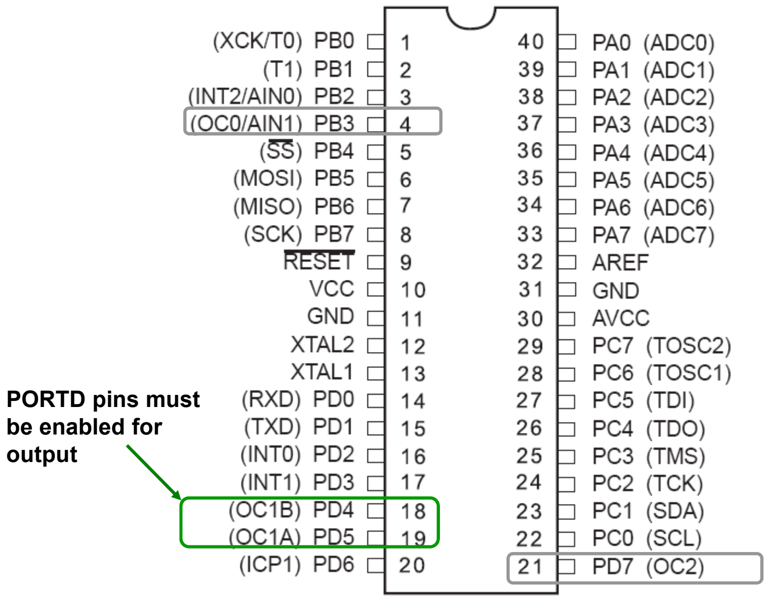
10.2 Output Compare Unit in Timer 1

- Timer 1 has two output compare channels: A and B.
- Timer 1 is continuously compared to OCR1A, OCR1B, or a fixed limit.
- When a match occurs, flag OCF1x is set, where x = 'A' or 'B'.
- When a match occurs, Timer 1 can
  - trigger an output compare interrupt.
  - change output compare pins OC1x.

Output Compare Unit ─ Block diagram



Output Compare Unit ─ Relevant pins



Output Compare Unit ─ Main aspects

10.2.1 What changes can be made to output compare pins OC1x?

10.2.2 What are the available operation modes of timer 1?

10.2.3 Steps to produce a custom waveform?

10.2.4 How to use output compare interrupt?

10.2.1 Changing output compare pins OC1x

- When a timer event (compare match, or timer = 0) occurs, pins OC1x can be automatically updated:
  - toggled,
  - set to 1,
  - cleared to 0, or
  - no change.
- The type of update is controlled by two flags in register TCCR1A : COM1x1 and COM1x0 where x = 'A' or 'B'.

COM1A1	COM1A0	COM1B1	COM1B0	FOC1A	FOC1B	WGM11	WGM10

Register TCCR1A

for pin OC1A for pin OC1B

- The exact change depends also the operation mode of Timer 1.

10.2.2 Operations modes of Timer 1

- Timer 1 has 15 operation modes, divided into 5 groups:
  - Normal
  - Clear Timer on Compare Match
  - Fast PWM
  - Phase correct PWM
  - Phase and Frequency Correct PWM

  Three PWM groups
- The operation mode is selected by 4 bits:

  WGM = {WGM13, WGM12, WGM11, WGM10}
- Each group of operations will be discussed next.

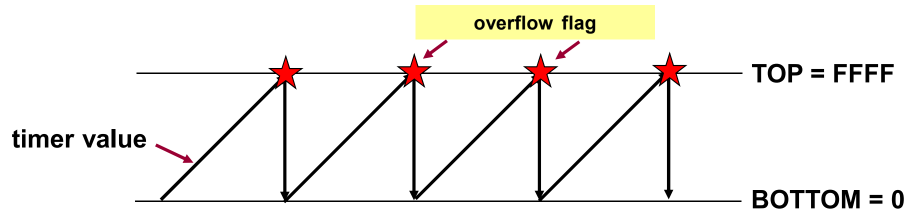
Selecting operation mode of Timer 1

COM1A1	COM1A0	COM1B1	COM1B0	FOC1A	FOC1B	WGM11	WGM10	Register TCCR1A
ICNC1	ICES1	-	WGM13	WGM12	CS12	CS11	CS10	Register TCCR1B

Mode	WGM13	WGM12 (CTC1)	WGM11 (PWM11)	WGM10 (PWM10)	Timer/Counter Mode of Operation	TOP	Update of OCR1X	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	BOTTOM	TOP
6	0	1	1	0	Fast PWM, 9-bit	0x01FF	BOTTOM	TOP
7	0	1	1	1	Fast PWM, 10-bit	0x03FF	BOTTOM	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

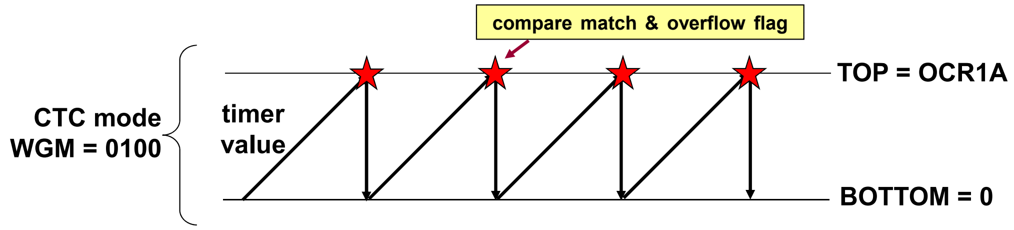
10.2.2a Normal mode

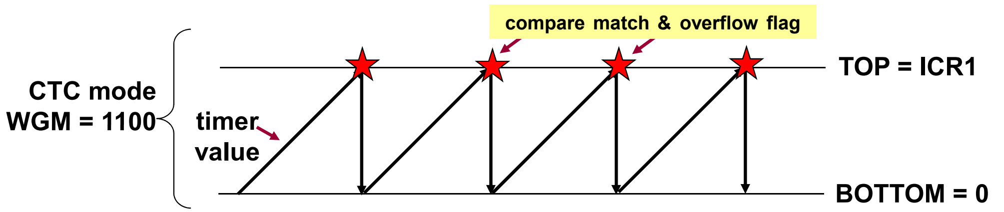
- Timer repeatedly counts from 0 to TOP, where TOP = 0xFFFF.
- Overflow flag TOV1 is set after timer reaches TOP.
- No change is allowed on output compare pins OC1x.
- Discussed in Lecture 9.



10.2.2b CTC modes

- Timer is reset to 0 when it reaches the value in OCR1A or ICR1.





CTC modes

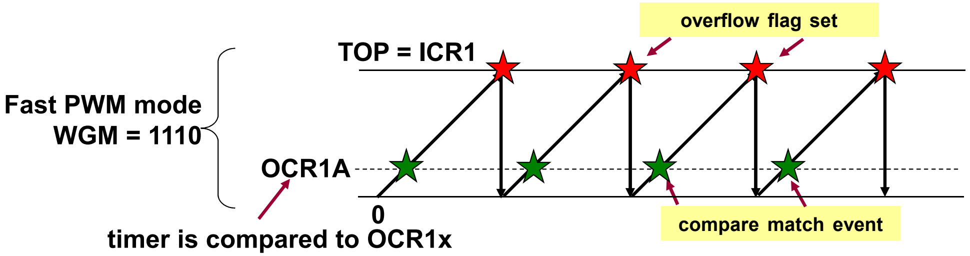
- On compare match, change of pins OC1x is allowed.

COM1A1/ COM1B1	COM1A0/ COM1B0	Description
0	0	Normal port operation. OC1A/OC1B disconnected.
0	1	Toggle OC1A/OC1B on compare match.
1	0	Clear OC1A/OC1B on compare match. (Set output to low level).
1	1	Set OC1A/OC1B on compare match. (Set output to high level).

Changing OC1x in CTC mode

10.2.2c Fast PWM modes

- Timer goes from 0 to TOP, where TOP is equal to
  - 0xFF (for 8-bit mode, WGM = 0101),
  - 0x1FF (for 9-bit mode, WGM = 0110),
  - 0x3FF (for 10-bit mode, WGM = 0111),
  - value in ICR1 (for WGM = 1110),
  - value in OCR1A (for WGM = 1111).
- Compare match occurs when timer = OCR1x register.



Fast PWM modes

- On compare match, change of pins OC1x is allowed.

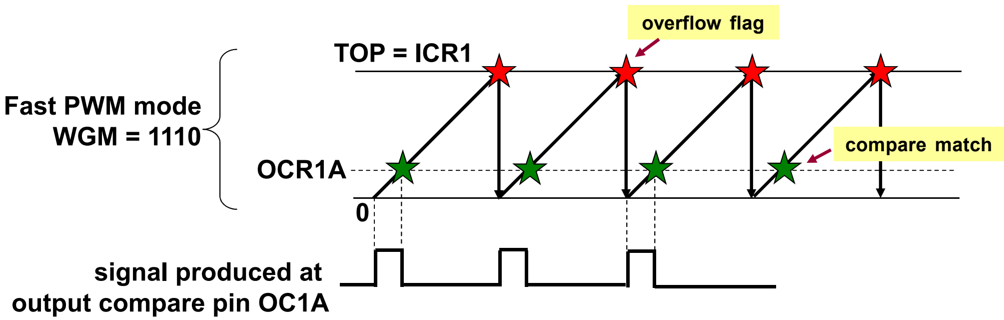
COM1A1/ COM1B1	COM1A0/ COM1B0	Description
0	0	Normal port operation, OC1A/OC1B disconnected.
0	1	- WGM13:0=15: Toggle OC1A on Compare Match, OC1B disconnected. - For other WGM13:0 settings, normal port operation, OCnA/OCnB disconnected.
1	0	Clear OC1A/OC1B on compare match. Set OC1A/OC1B at BOTTOM.
1	1	Set OC1A/OC1B on compare match. Clear OC1A/OC1B at BOTTOM. (inverting mode)

Changing OC1x in fast PWM mode
(Note that BOTTOM = 0)

Fast PWM modes

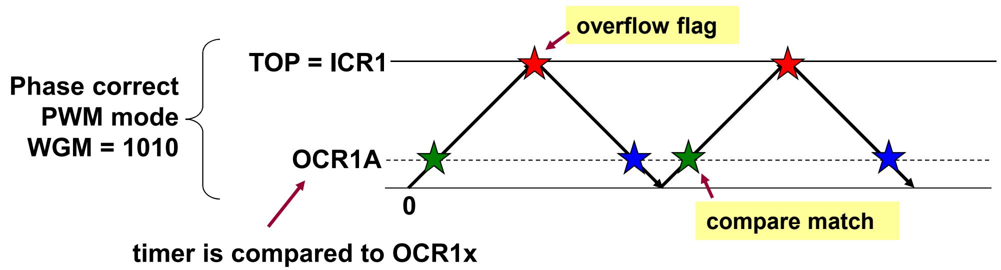
COM1A1/ COM1B1	COM1A0/ COM1B0	Description
0	0	-
0	1	-
1	0	Clear OC1A/OC1B on compare match Set OC1A/OC1B at BOTTOM
1	1	-

Used in the example below (row 1 0)



10.2.2d Phase Correct PWM modes

- Timer counts <u>up and down</u> between 0 and TOP, where TOP is equal to
  - 0xFF (for 8-bit mode, WGM = 1000)
  - 0x1FF (for 9-bit mode, WGM = 0010)
  - 0x3FF (for 10-bit mode, WGM = 0011)
  - value in ICR1 (for WGM = 1010)
  - value in OCR1A (for WGM = 1011)
- Compare match occurs when timer = OCR1x register.



Phase Correct PWM modes

- On compare match, change of pins OC1x is allowed.

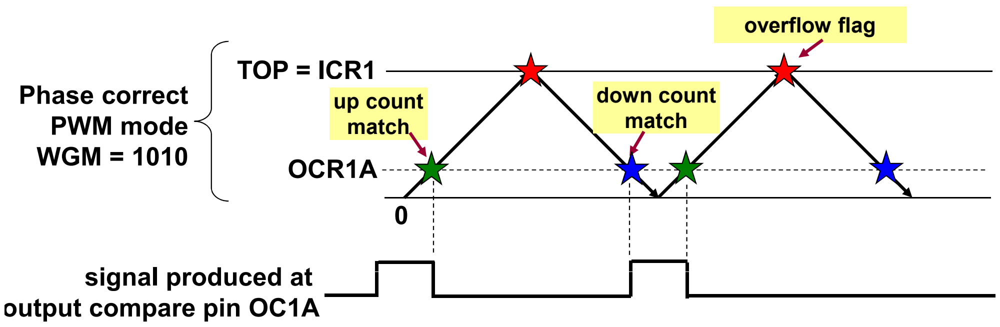
COM1A1/ COM1B1	COM1A0/ COM1B0	Description
0	0	Normal port operation, OC1A/OC1B disconnected.
0	1	- WGM13:0= 9 or 14: Toggle OCnA on Compare Match, OCnB disconnected. - For other WGM13:0 settings, normal port operation, OC1A/OC1B disconnected.
1	0	Clear OC1A/OC1B on compare match when up-counting. Set OC1A/OC1B on compare match when down-counting.
1	1	Set OC1A/OC1B on compare match when up-counting. Clear OC1A/OC1B on compare match when down-counting.

Changing OC1x in Phase Correct PWM mode

Phase Correct PWM modes

Used in the example below

COM1A1/ COM1B1	COM1A0/ COM1B0	Description
0	0	-
0	1	-
1	0	Clear OC1A/OC1B on compare match when up-counting. Set OC1A/OC1B on compare match when down-counting.
1	1	-



10.2.2e Phase and Frequency Correct PWM modes

- Timer counts up and down between 0 and TOP, where TOP is equal to
  - value in ICR1 (for WGM = 1000) or
  - value in OCR1A (for WGM = 1001)
- Compare match occurs when timer = OCR1x register.
- On compare match, changing pins OC1x is done similarly in Phase Correct PWM modes.

10.2.3 Producing a custom waveform

Steps to produce a custom waveform on an output compare pin OC1x

- Select the operation mode of Timer 1: CTC, fast PWM, or phase correct PWM, …
- Select how output compare pin will be updated on compare match event.
- Configure timer 1: clock source, prescaler, …

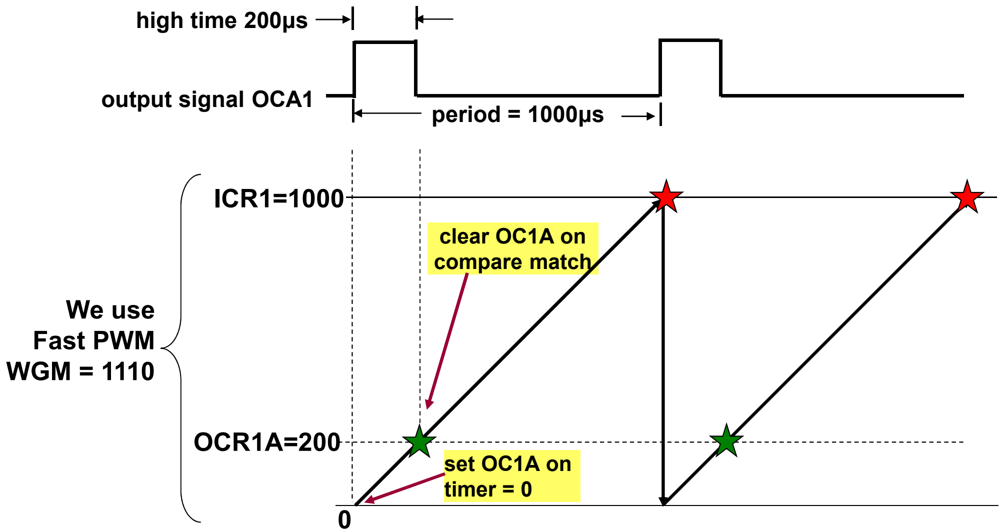
set registers TCCR1A and TCCR1B

- Put correct values in the output compare registers.

set register OCR1A or ICR1

Example 10.1: Producing a custom waveform

Use Timer 1 to create a signal with period = 1000µs, high time = 200µs.



Example 10.1: Determining registers

COM1A1	COM1A0	COM1B1	COM1B0	FOC1A	FOC1B	WGM11	WGM10	
1	0	0	0	0	0	1	0	Register TCCR1A

ICNC1	ICES1	-	WGM13	WGM12	CS12	CS11	CS10	
0	0	0	1	1	0	0	1	Register TCCR1B

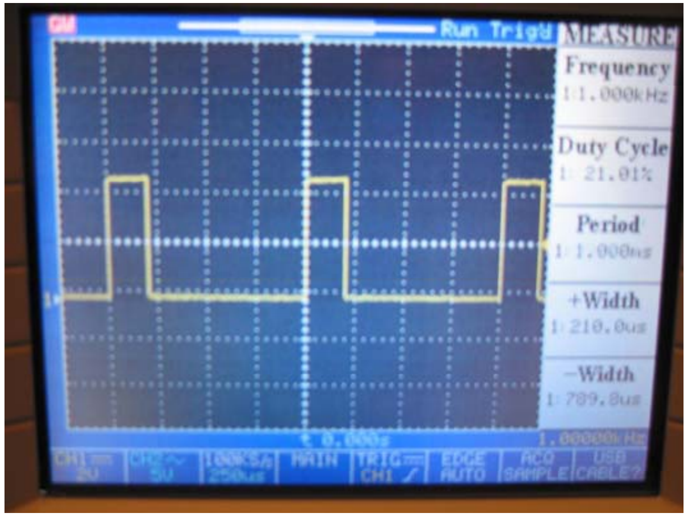
- ICR1 = 1000 → period of output signal
- OCR1A = 200 → pulse width of output signal
- WGM3:0 = 1110 → Fast PWM mode where TOP = ICR1.
- CS12:0 = 001 → Internal clock, no prescaler
- COM1A1:0 = 10 → set OC1A when timer = 0
  clear OC1A when compare match

Example 10.1: Program make_pwm.c

```
#include <avr\io.h>

int main(void) {

   DDRD=0b00100000; // set port D for output (D.5 is OC1A)

   // Set register TCCR1A
   // WGM11:WGM10    = 10: with WGM13-WGM12 to select timer mode 1110
   //                      Fast PWM, timer 1 runs from 0 to ICR1
   // COM1A1:COM1A0 = 10: clear OC1A when compare match, set OC1A when 0
   //                      compare match occurs when timer = OCR1A
     TCCR1A = 0b10000010;

   // Set register TCCR1B
   // WGM13:WGM12    = 11
   // CS12:CS0       = 001: internal clock 1MHz, no prescaler
   TCCR1B = 0b00011001;

   ICR1  = 1000;      // period of output signal
   OCR1A = 200;       // pulse width of output signal

   while(1){;}
}
```


Example 10.1: Testing

- Download program make_pwm.hex to STK500 board.
- Use oscilloscope to measure signal on pin OC1A (D.5).



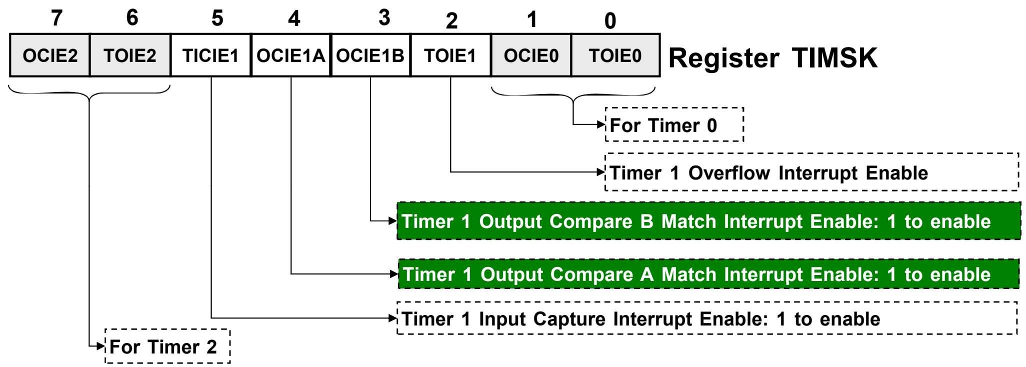
10.2.4 Output Compare Interrupt

- We've learnt to produce PWM signals on dedicated output compare pins OC1x.

- <u>What if we need to</u>
  - perform custom operations at predefined time instants, or
  - produce signals on an arbitrary output pin?

- <u>A possible approach is to</u>
  - trigger an output compare interrupt at correct time instants.
  - write an ISR that performs the custom operations.

Output Compare Interrupt

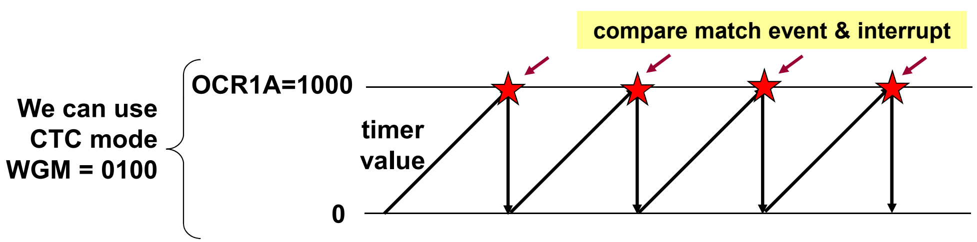


- Output compare interrupt is enabled by OCIE1A and OCIE1B flag for channel A and B, respectively.

- C names for these interrupts: TIMER1_COMPA_vect and TIMER1_COMPB_vect.

Example 10.2: Output Compare Interrupt

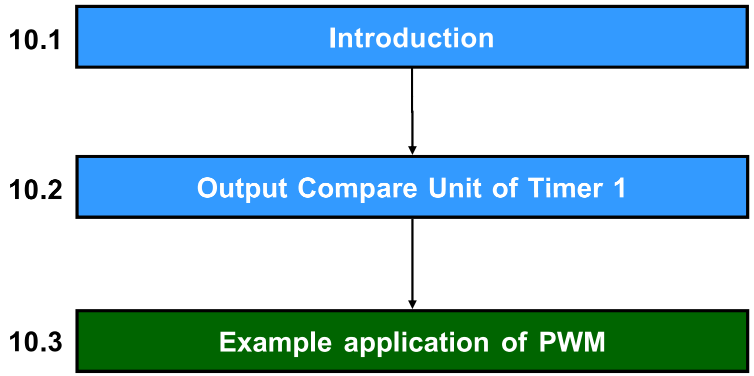
Use Timer 1's output compare interrupt to toggle pin B.1 every 1000μs.



Example 10.2: Program oc_int.c

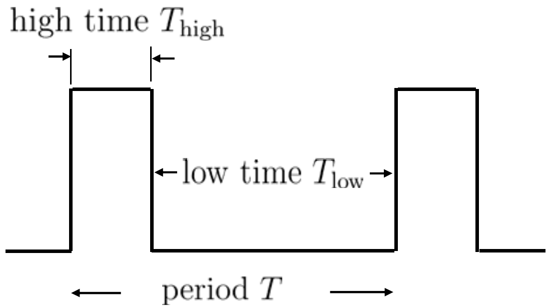
```
#include <avr\io.h>
#include <avr\interrupt.h>
ISR(TIMER1_COMPA_vect){
   PORTB = PORTB ^ 0b00000010;  // toggle B.1 using XOR operator
}

int main(void) {
   DDRB = 0xFF;  // set port B for output
   PORTB = 0xFF; // initial value of port B

   // WGM11:WGM10   = 00: with WGM13-WGM12 to select timer mode 0100
   //                     CTC, timer 1 runs from 0 to OCR1A
   TCCR1A = 0b00000000;

   // WGM13:WGM12   = 01
   // CS12:CS0      = 001: internal clock 1MHz, no prescaler
   TCCR1B = 0b00001001;

   OCR1A = 1000;          // interrupt will be triggered every 1000us

   TIMSK = (1<< OCIE1A); // enable Timer 1 Output Compare A interrupt
   sei();                // enable interrupt subsystem
   while(1){;}
}
```

Lecture 10's sequence



10.3 Example application of PWM

- PWM signals are commonly used in embedded applications: motor control, sound alarm and radio transmission.
- A PWM signal is a periodic, rectangular pulse. The period and the duty cycle can vary.
- Here, we'll generate a PWM signal to control a servo motor.



$$\text{mark space ratio} = \frac{T_{high}}{T_{low}}$$

$$\text{duty cycle} = \frac{T_{high}}{T}$$

Controlling a servo motor

- We use a servo motor S3003.
- It has three wires
  - Black: Ground
  - Red: DC supply between (4.8V, 6V)
  - White: PWM signal
- The frequency of the PWM signal is 50Hz.
- This motor have a rotation range of 180°.
- To keep the motor at a given angle, we must send a PWM signal of a specific duty cycle.
- Range of duty cycle: 1% to 12%.
- See video of motor and PWM waveform.

www.futaba-rc.com

Controlling a servo motor

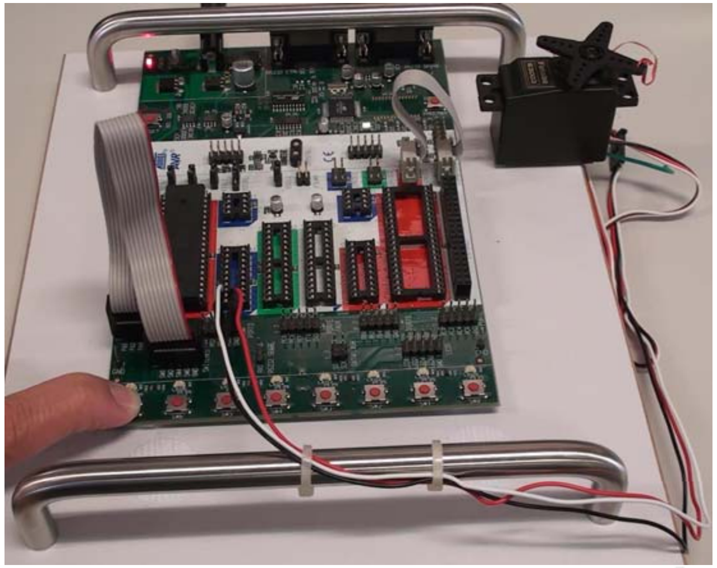
Write C program that lets the user press switches SW6 and SW7 on STK500 board to rotate the motor left and right, respectively.

- The switches can be connected to pins of port A.
- Depending on which switch is pressed, we increment or decrement the duty cycle.
- We then produce a PWM signal on pin OC1A with
  - a period of 20000μs,
  - a specific duty cycle between 1% and 12%.

Controlling a servo motor: motor_control.c [Ex 10.3]

```c
#include <avr\io.h>
int main(void) {
    unsigned int period, duty_cycle, high_time;
    unsigned char button;
    DDRA = 0b00; DDRB = 0xFF; // set port A for input, port B for output
    DDRD = 0b00100000;        // set pin D.5 for output (OC1A)

    // WGM11:WGM10   = 10: with WGM13-WGM12 to select timer mode 1110
    //                     Fast PWM, timer 1 runs from 0 to ICR1
    // COM1A1:COM1A0 = 10: clear OC1A when compare match, set OC1A when 0
    TCCR1A = 0b10000010; // compare match occurs timer = OCR1A
    TCCR1B = 0b00011001; // WGM13:WGM12=11; CS12:CS0=001: internal clock 1MHz, no prescaler

    period = 20000; // PWM frequency = 50Hz, period = 20000us
    duty_cycle = 6; // initial duty cycle
    ICR1  = period;                       // period of output PWM signal
    high_time = (period/100) * duty_cycle; // calculate high time
    OCR1A = high_time;                    // set high time of output PWM signal
    while (1){
          if (button == PINA)             // ignore repeated press
              continue;
          button = PINA; PORTB = button;  // store button press, display on port B
          if ((button & 0b11000000) == 0b11000000) // ignore all except buttons SW6 and SW7
              continue;

          if ((button & 0b10000000) == 0) // Increment duty cycle if switch SW7 is pressed
              duty_cycle = (duty_cycle<12)?duty_cycle+1:duty_cycle;

          if ((button & 0b01000000) == 0) // Increment duty cycle if switch SW6 is pressed
              duty_cycle = (duty_cycle>1)?duty_cycle-1:duty_cycle;
          high_time = (period/100)*duty_cycle;// calculate high time
          OCR1A = high_time;                  // set high time of output signal
    }
}
```


Controlling a servo motor: Testing



Video: [avr]/ecte333/motor_control.mp4

Lecture 10's summary

- What we learnt in this lecture:
  - Output Compare functionality of a timer.
  - Using output compare in Timer 1 to generate signals and execute tasks at specific times.
  - Generating PWM signals for motor control.
- What are next activities?
  - Tutorial 10: 'Pulse Width Modulator' .
  - Lab 10: 'Pulse Width Modulator'
    - Complete the online Pre-lab Quiz for Lab 10.
    - Write programs for Tasks 1 and 2 of Lab 10.
    - See video demos of Lab 10: [avr]/ecte333/lab10_task1.mp4
      [avr]/ecte333/lab10_task2.mp4

Lecture 10's references

- Atmel Corp., 8-bit AVR microcontroller with 16K Bytes In-System Programmable Flash ATmega16/ATmega16L, 2007, [Timers]. Manual
- S. F. Barrett and D. J. Pack, Atmel AVR Microcontroller Primer: Programming and Interfacing, 2008, Morgan & Claypool Publishers, [Chapter 5: Timing Subsystem].

Lecture 10's references

- M. Mazidi, J. Mazidi, R. McKinlay, "The 8051 microcontroller and embedded systems using assembly and C," 2nd ed., Pearson Prentice Hall, 2006, [Chapters 9].

- M. Mazidi and J. Mazidi, "The 8086 IBM PC and compatible computers," 4th ed., Pearson Prentice Hall, 2003, [Chapters 13].

- P. Spasov, "Microcontroller technology the 68HC11," 3rd ed., Prentice Hall, 1999, [Chapters 11].

- H. Huang, "MC68HC12 an introduction: software and hardware interfacing," Thomson Delmar Learning, 2003, [Chapter 8].