Med dette dokument vises, hvordan man kan bruge Timer1 i Arduinoens ATMEGA328P til at generere et PWM-signal på en pin.

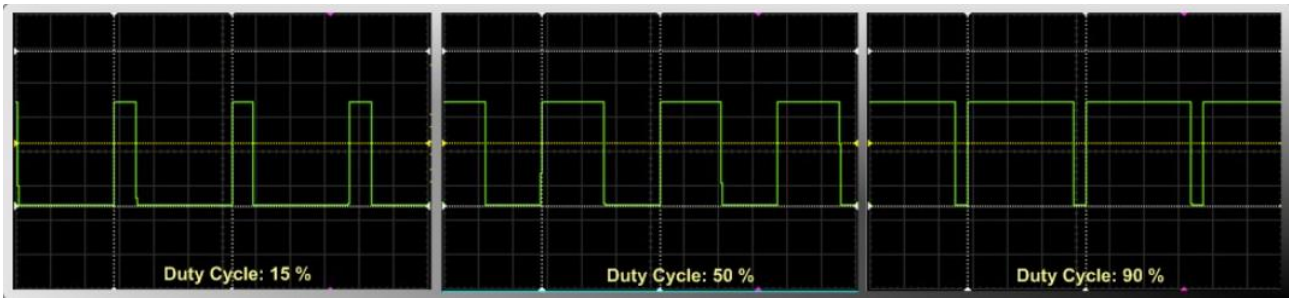
De umiddelbart indbyggede PWM-muligheder på de pins, der er mærket med en bølgestreg, - Pin: 3, 5, 6, 9, 10 & 11, kan godt bruges til fx at fade lysdioder. Det sker vha. funktionen analogWrite()

Men frekvensen de giver (og dermed også timebasen) er alt for høj til at styre store belastninger.

Frekvensen er 490 Hz, dog 980 Hz på pin 5 & 6

Det er OK hvis man bare skal fade en LED.

Her er vist princippet i Puls Bredde Modulering. Den energi, der afsættes i belastningen, er proportional med pulsens dutycycle.



$$DutyCycle = \frac{T_{on}}{T_{on} + T_{off}} \cdot 100\%$$

Men hvis man skal bruge en PWM-funktion til fx at styre en skruemaskine, bør frekvensen ikke være for stor.

Det er i skiftene, hvor MOSFET – transistoren skal gå ON eller OFF, der afsættes effekt.
Det er fordi, det er svært at få til at foregå ”momentant”. Transistorens Gate virker som en – godt nok lille – kondensator i forhold til både Drain og Source. Og det betyder, at ved hver skift skal denne kapacitet enten lades op, - eller aflades, og dertil kræves et antal ladninger, dvs. en strøm.

Det vil derfor altid tage ”lidt tid” for transistoren at skifte. Og på dens vej, fx fra at være off til on, er den jo på et tidspunkt kun halvt ”on”. Derfor vil der være et spændingsfald over den, - og da der går en strøm, vil der være effektafsætning – læs varmeafsætning.

$$P = U \cdot I = I^2 \cdot R \; [Watt]$$

Derfor bør timebasen være så lang som muligt, og dermed frekvensen så lav som muligt.

Vi har hidtil arbejdet med Atmega 328P's indbyggede timerfunktion med tilhørende interrupt. Her brugte vi timeren til "blot" at tælle krystallets pulser. - Og give et interrupt på bestemte tidspunkter.

Vi brugte mode 4 af de 15 modes, processoren kan opsættes til.

Det fremgår af skemaet her:

Timer1 mode select:

Ud fra dette skema ses de forskellige modes:

Langt de fleste modes kan bruges til forskellige former for PWM.

I det følgende gennemgås PWM mode 15.

Mode	WGM13	WGM12 (CTC1)	WGM11 (PWM11)	WGM10 (PWM10)	Timer/Counter Mode of Operation	TOP	Update of OCR1x at	TOV 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

PWM MODE 15

Mode 15 kaldes også Fast PWM. Ved Fast PWM er princippet, at en tæller tælles op indtil en bestemt værdi, hvorefter den starter forfra igen.

(I de "langsomme" – dvs. ikke Fast modes - tæller tælleren op til en værdi og derefter tælles ned til 0 igen

Krystallets frekvens sendes via prescaleren til Timer1, der har 16-bit.

Tællerens værdi bliver hele tiden sammenlignet med værdien i to 16-bit Compare-registre, A og B.

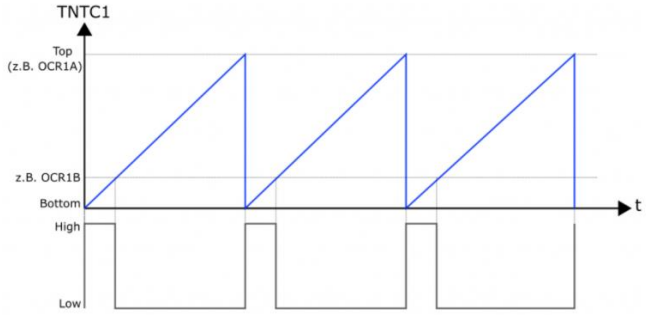
Når tællerens værdi er lig med værdien i Compare-reg-B, resettes udgang pin 10.

Når tælleren når op til værdien i Compare-reg-A, resettes den – og pin 10 bliver høj.

Fast PWM, timer1, mode 15.

Når timerens værdi når op til den værdi, der er sat ind i compare-register OCR1A, resettes den, - og output-pin 10 settes.

Og, - hvis timerens værdi er lig med værdien sat ind i compare-register OCR1B, resettes udgangen på pin 10.



Hvordan pin 10 reagerer, kan dog ændres. Det sker ved at indstille to bit – bit 4 & 5 i SFR TCCR1A. Se senere !!

Ved nu at vælge den rigtige prescaler, - og ved at indsætte en værdi i register OCR1A – (Output Compare Register Timer1 # A) kan man vælge en passende Timebase for den funktion, man skal kontrollere.

Og ved at indsætte en værdi i Register OCR1B - (Output Compare Register Timer1 # B) bestemmer man dutycycle på Arduens pin 10.

Et program kan så løbende – mens timerfunktionen kører, - ændre værdien i OCR1B-registeret, og derved ændre på pin-10-signalets pulsbredde (dutyCycle).

Det kan fx ske efter læsning af et potmeter.

Det kræver selvfølgelig noget opsætning – og udregninger.

Valg af Timebase

Eks: **Prescaler = 8:**

Krystallets frekvens er 16 MHz. Vælges fx en **prescaler på 8**, får tælleren 2 MHz.

Maks værdi, der kan tælles vha. 16 bit er 65.535

Tælleren giver derfor overflow efter 65535 / 2E6 = 0,033 sek. = 33 mS.

Det giver en timebase på max 33 mS. Og derfor mindst: 1/33mS = 30,5 PWM-pulser pr sekund.

Vælges top-compare-værdien så lavere end 65.535 bliver timebasen jo tilsvarende kortere – og frekvensen så højere.

Eks: **Prescaler = 256:**

Ønskes en længere timebase må prescaleren være større.

Med 256 får tælleren: 16E6/256 = 62.500 pulser pr sekund.

Regnestykket bliver: Max Timebase 65535 / (16E6 / prescaler)

Ved prescaler 256 fås: Det tager 1,05 sekunder at nå til max.

På skemaform

På skemaform ser det sådan ud:

Prescaler	Frekvens til tæller Hz	Max Timebase 65535 / Frekv. [S]	Minimum PWM-frekvens [Hz]
1	16 M	4,09 m	244,1
8	2 M	32,7 m	30,5
64	250 K	0,262	3,81
256	62,5 K	1,05	0,95
1024	15,625 K	4,19	0,238

Nu gælder ovenstående jo kun hvis tælleren skal tælle helt op til 0xFFFF = 65.535

Men det behøves ikke. I mode 15 resettes tælleren, når den når op på den samme værdi, der er indsat i Compare-A-registeret.

Det gør jo, at man kan vælge sin timebase selv.

Så en formel for timebasen er: $Timebase = \frac{(TOP) \cdot Prescaler}{16E6}$

Med TOP menes der den værdi, tælleren skal tælle op til. Hvis valgte MODE tæller op til værdien i register OCR1A, fås:

$$Timebase = \frac{(OCR1A) \cdot Prescaler}{16E6}$$

For en given ønsket frekvens findes Compare-A – værdien af:

$$OCR1A = \frac{16E6}{frekvens \cdot Prescaler}$$

OCR1A-værdien må selvfølgelig ikke være mere end 65.535, da det er det største tal, der kan være i 16-bit registeret.

For indstilling af prescaler, se grafik senere, eller se grafikken i dokumentet om timer-interrupt.:

Opsætning af timeren til PWM:

I maven på ATMEGA328P er der flere SFR, - Special Function Registre, der bruges til at indstille de funktioner, man ønsker processoren skal bruge.

De, der er relevante for timer1, ses i følgende grafik:

Vha. de 4 lilla bits vælges timer-mode.

De kaldes ”Bit Wave Generation Mode Bit” - WGM1(0-3)

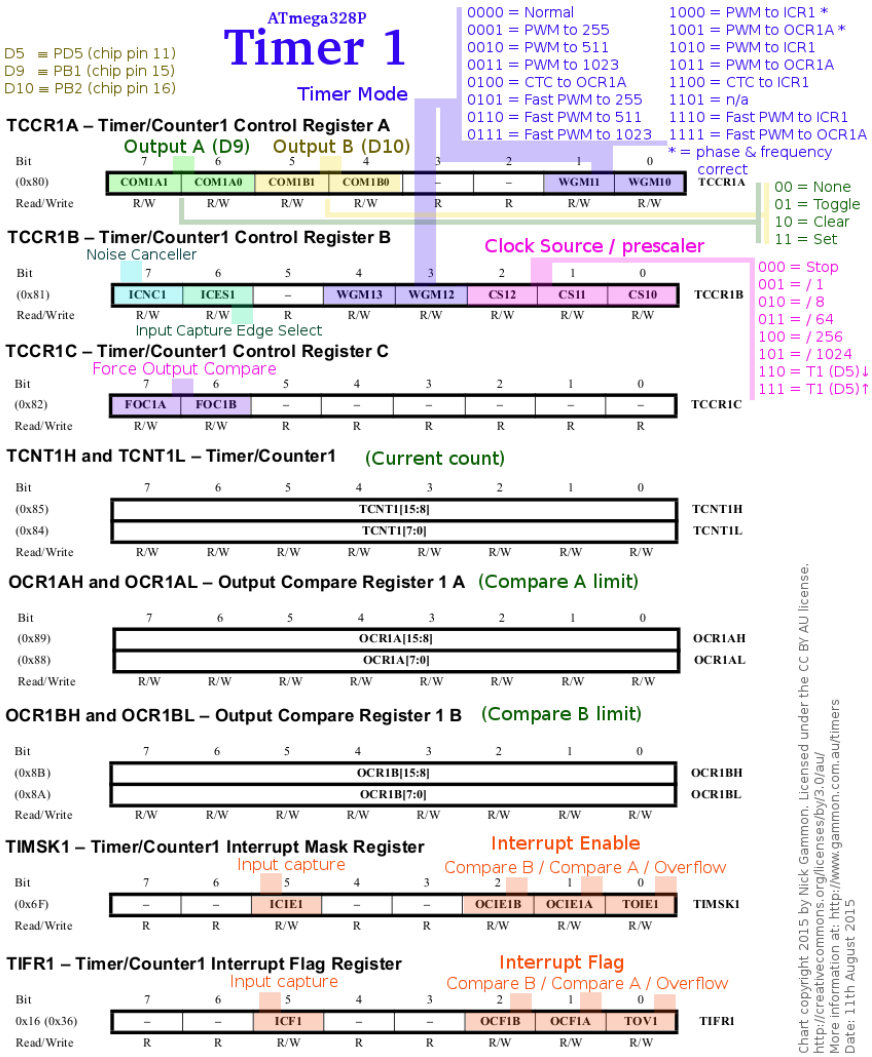
De er placeret i hhv. register TCCR1A og TCCR1B.

Alle 4 bit skal sættes for at vælge mode 15.

Bit 5 og 4 i register TCCR1A (gule) bestemmer hvad der skal ske når værdien i tælleren (TCNT1 L&H) bliver lig med værdien i register OCR1B (CompareB limit).

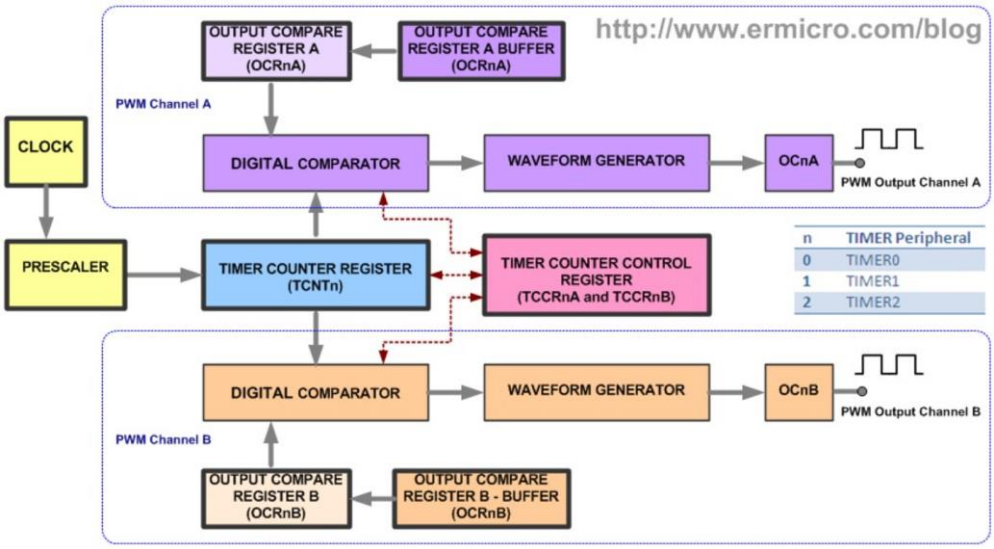
De sættes til Clear, = 10, som betyder at output Pin 10 cleares når tællerens værdi er lig med compareværdien.

Endelig vælges prescaleren ved at sætte bit 2, 1 og 0 i SFR-register TCCR1B



Kilde: https://www.gammon.com.au/forum/?id=11504&page=2

På næste grafik ses ” opbygningen ” af timer – og compare-systemet i ATMEGA328

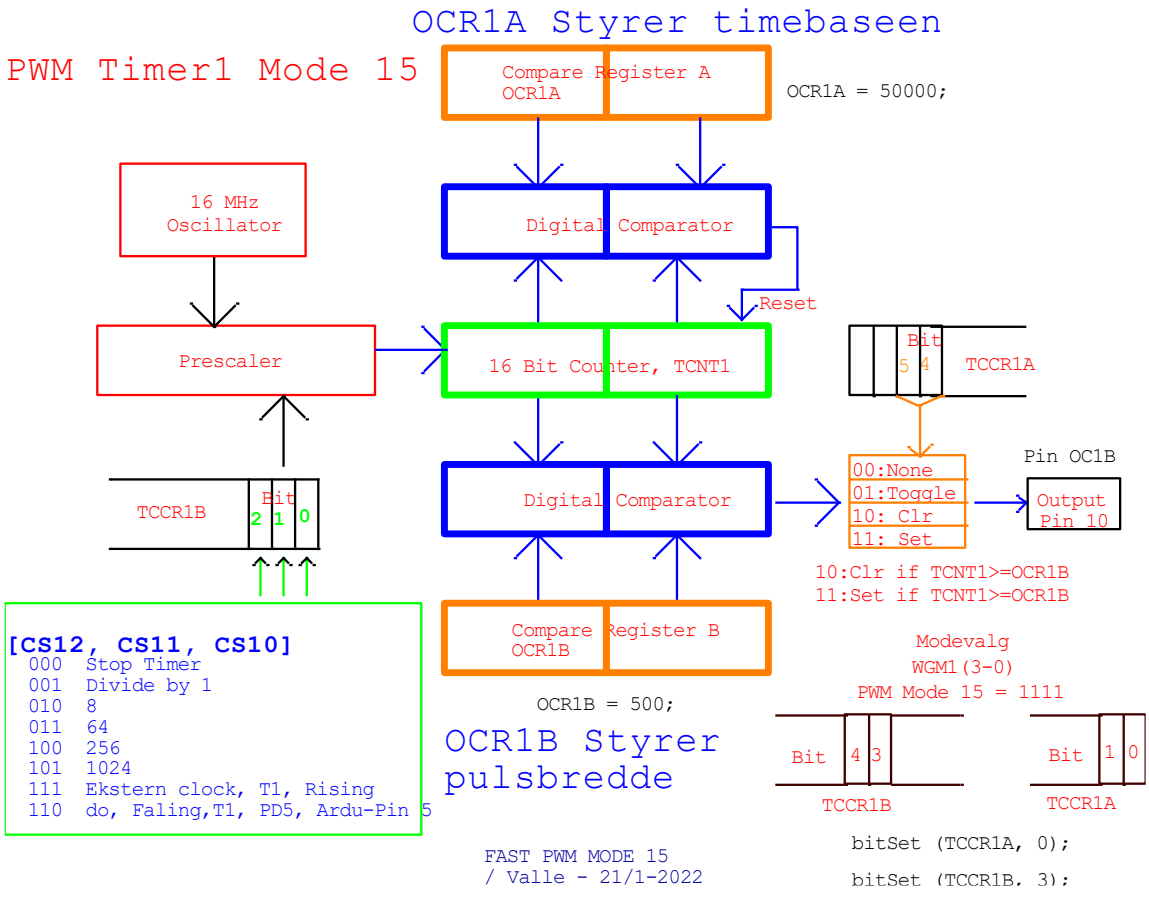


De interne registre kan ses vha. dette generelle blokskema:

Nedenfor ses min egen – tilrettet til mode 15.

Kilde: http://www.ermicro.com/blog/?p=1971

Her ses min tegning: Fast PWM timer1 Mode 15



Ud fra ovenstående skema kan man i de forskellige SFR's indstille den funktion, der ønskes:

Her følger en subrutine, der kan bruges til at indstille PWM i ATMEGA328P.

Funktion til opsætning af Timer1 som PWM generator:

```
void timer1_pwm_init()  // Udkommenter de linjer, der ikke skal bruges
{
  TCCR1A = 0; // clear først alle tællerens styrebit
  TCCR1B = 0;

// Vælg wave generation mode, WGM1(3 - 0) for timer1
  bitSet(TCCR1B, 4);  // Mode 15 = 1111, Fast PWM til OCR1A
  bitSet(TCCR1B, 3);
  bitSet(TCCR1A, 1);
  bitSet(TCCR1A, 0);

  bitSet(TCCR1A, 5);  // Pin 10, Compare-B match, ( OC1B )
  bitSet(TCCR1A, 4);  // 00 = none, 01 = Toggle, 10=Clear, 11=Set
                      // 10=Clear hvis tæller = compareværdi. 11=modsat fase

// vælg Prescaler
   bitSet(TCCR1B,2); // 000=stop, 001=/1, 010 =/8, 011=/64,
   bitSet(TCCR1B,1); // 100=/256, 101=/1024, 110 & 111= ext pin
   bitSet(TCCR1B,0);

// vælg Compare værdi for
  OCR1B = 1000; // Eksempel !!
  OCR1A = 20000; // Eksempel, Top, Timer1-resetværdi i mode 15

// enable pin som output:
  pinMode (10, OUTPUT);  // Compare-B-output
}
```

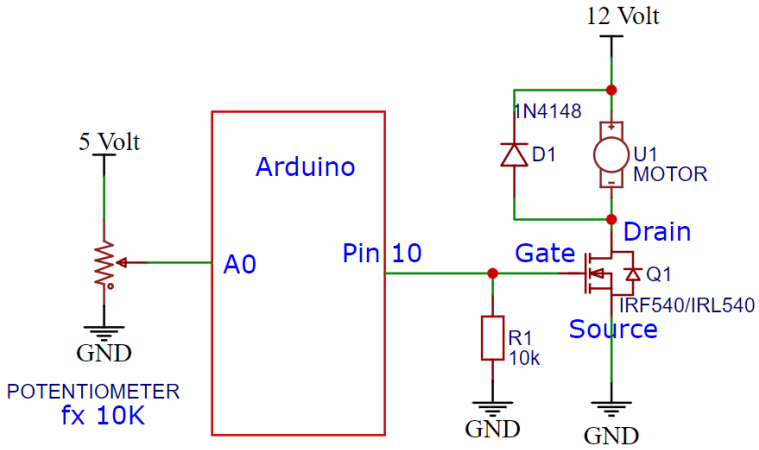
Her er et eksempel på en forsøgsopstilling.

R1 sidder der for at holde Gaten lav hvis ledningen til Arduinoen afbrydes.

Husk altid: **Fælles Stel**

Dvs. at Arduinoens Nul og 12 Volt-forsyningens Nul skal forbindes.

Husk dioden over en induktiv belastning.



Andre modes:

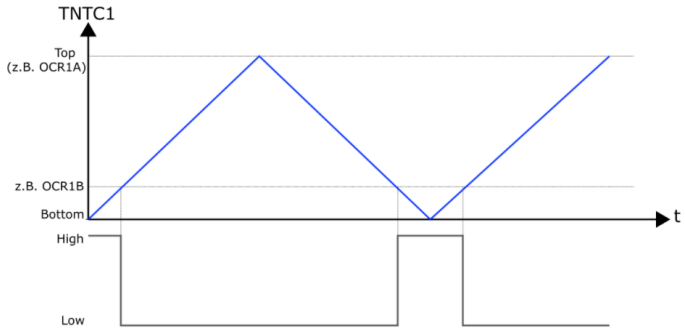
```
1000 = PWM to ICR1 *
1001 = PWM to OCR1A *
1010 = PWM to ICR1
1011 = PWM to OCR1A
1100 = CTC to ICR1
1101 = n/a
1110 = Fast PWM to ICR1
1111 = Fast PWM to OCR1
```

I mode 14 tælles op til værdien i register ICR1 (16 bit), og derfor har man både OCR1A og OCR1B at bruge til PWM-pins.

I mode 8, 9, 10 & 11 tælles op til værdien i ICR1 eller OCR1A, og derefter ned igen til 0..

Derfor bliver **timebasen dobbelt så lang.**

Her er vist princippet i mode 9 og 11, hvor der tælles op til indholdet i OCR1A og derefter ned igen til 0.



Kilde: https://wolles-elektronikkiste.de/timer-und-pwm-teil-2-16-bit-timer1

Fast PWM Mode

The Fast Pulse Width Modulation or Fast PWM modes (modes 5, 6, 7, 14, and 15, WGM[3:0]= 0x5, 0x6, 0x7, 0xE, 0xF) provide a high frequency PWM waveform generation option.

The Fast PWM differs from the other PWM options by its single-slope operation. The counter counts from BOTTOM to TOP then restarts from BOTTOM.

In Fast PWM mode the counter is incremented until the counter value matches either

- One of the fixed values 0x00FF, 0x01FF, or 0x03FF (WGM[3:0] = 0x5, 0x6, or 0x7)
- The value in ICR1 (WGM[3:0]=0xE (mode 14))
- The value in OCR1A (WGM[3:0]=0xF (mode 15)).

The counter is then cleared at the following timer clock cycle.

Note: The Prescale divider values for timer1: (1, 8, 64, 256, or 1024).

Kilde: https://microchipdeveloper.com/8avr:avrtimerover

Kilder:

God: http://www.righto.com/2009/07/secrets-of-arduino-pwm.html

https://maxembedded.wordpress.com/2011/06/29/avr-timers-timer2/

https://wolles-elektronikkiste.de/en/timer-and-pwm-part-2-16-bit-timer1
(Her er også andre PWM-modes forklaret !)

https://www.eprojectszone.com/how-to-modify-the-pwm-frequency-on-the-arduino-part2timer-1-and-phase-correct-pwm-mode/

For andre modes: se: https://embedds.com/programming-16-bit-timer-on-atmega328/