Pulsbreddemodulering

Pulsbreddemodulering bruges mange stedet til at styre energiafsætning i et system. – Eller forstået på en anden måde, til at regulere strømmen til et system.

Fx kan PWM bruges til at styre mængden af energi overført af et induktionskomfur til en gryde.

Det sker ved at tænde og slukke for strømmen til kogepladen, i et bestemt mønster.

Der er behov for en timebase, der fx kan fx være på 5 sekunder.
Det giver mulighed for at tænde kogepladen i fx 50%, - eller 75% - af de 5 sekunder. Og på den måde justere energien, der overføres til gryden.

Det samme kan gøres for at styre omdrejningshastigheden for en skruemaskine. Men her skal timebasen nok være noget kortere.

Timebasen skal afpasses med det system, der skal styres. Bruges det til at dæmpe lyset i lysdioder, skal timebasen være mindre end 1/25 sekund, svarende til 25 Hz. Ellers vil øjet jo se at dioden blinker.

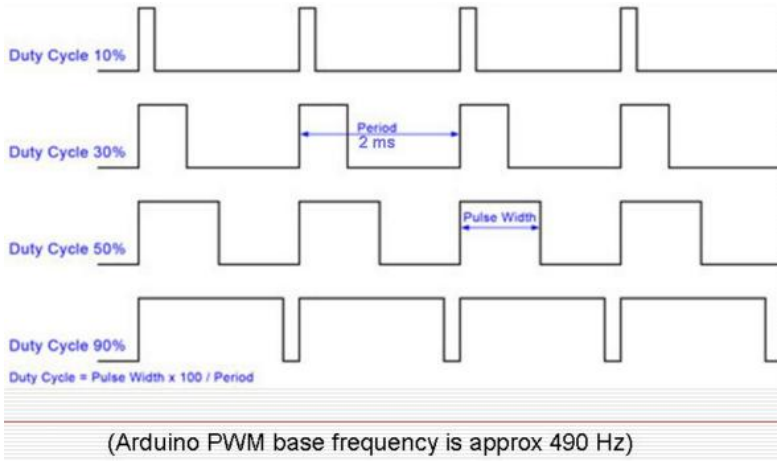
PWM-Princip

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

Begrebet ”Duty Cycle” beskriver i hvor mange procent af en periodetid, et signal er høj.

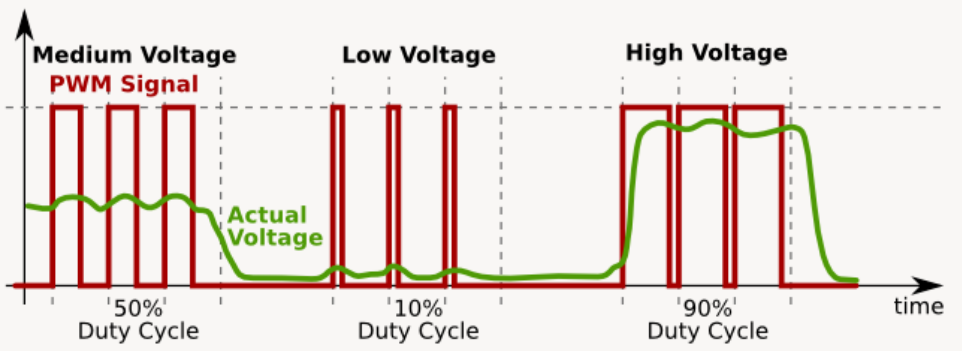
Det kan bruges til at dæmpe (= fade) lyset i en lysdiode.

Det illustreres på denne graf:



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

Den grønne graf illustrerer lysstyrken i en lysdiode, som den opleves af øjet ved pulsbredde-modulation.



Men vil man selv styre PWM-signalet, dets timebase, og dermed også dets frekvens, og dets dutycycle, kræves der lidt forståelse og opsætning af uC-en. Det kan fx være ønsket, hvis man skal styre en stor belastning vha. en MOSFet.

Her følger en gennemgang af teori om PWM, og hvordan man selv kan bruge Timer1 i Arduinoens ATMEGA328P til at generere et PWM-signal.

Arduinoens indbyggede PWM-funktion.

I Arduino – verdenen er der let adgang til at lave PWM-modulering.

Det er ” indbygget ” på de 6 pins, der er mærket med en bølgestreg ~.



Det er Pin: 3, 5, 6, 9, 10 & 11.

Disse pins kan fint bruges til fadning af Lysdioder.

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

I stedet for digitalWrite skal man bruge den **noget misvisende** funktion analogWrite:

```
analogWrite(pin, value); // Value kan have værdien fra 0 til 255.
```

De 490, - eller 980 Hz er så høj en frekvens, at man overhovedet ikke kan se at dioderne blinker. Og det virker OK. Husk formodstand for Lysdioder !

AnalogWrite-funktionen kan også bruges til at styre andre belastninger. Fx motorer, men her er det selvfølgelig nødvendig at bruge en ekstern transistor til at switche en større strøm, end Arduinoen kan håndtere.

Men man har ikke mulighed for at ændre på timebasen for PWM-signalet med analogWrite - funktionen.

Og man skal være opmærksom på, at hvis først analogWrite-funktionen er " sat i gang" – fortsætter den på pin'en. Når funktionen udføres, indstilles Hardware-timerne, og funktionen kører i baggrunden på hardware.

Vil man stoppe PWM-funktionen på en pin, kan man **måske** gøre det ved at skrive analogWrite(pin#, 0); til den pin, der bruges.

Men ifølge info fundet på nettet, vil opsætningen til PWM blive disablet ved at bruge en pinMode(pin#, OUTPUT); og dernæst digitalWrite(pin#, 0);

Herved stoppes PWM-timerne !!

Det er dog ikke sikkert, funktionen pinMode er nødvendig !!

Obs: Funktionen analogWrite definerer selv dens pin som OUTPUT !!

Lavere PWM-frekvens og MOSFet

Ideelt set vil den indbyggede PWM-funktion godt kunne bruges til at styre effektafsætning i fx en skruemaskine-motor.

Men motoren kræver meget større strøm, end en uC kan levere, Så der skal bruges en switch-transistor.

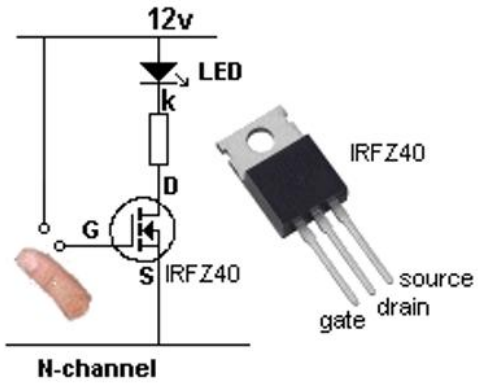
Til store strømme bruges typisk en MOSFet transistor.

En MOSFet transistor er en spændingsstyret transistor, der godt kan styres direkte fra en Arduino-pin på dens styre-pin, kaldet Gate.

Gaten er totalt isoleret fra Drain og Source med et Oxid-lag. Den har en indgangsmodstand på mere end 10^{12} Ohm !

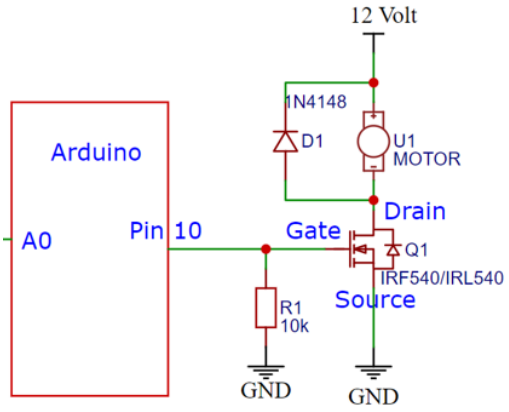
Derfor kan bare den lille strøm, der går gennem fingeren, tilføre gaten så stor ladning, at transistoren kan styres on.

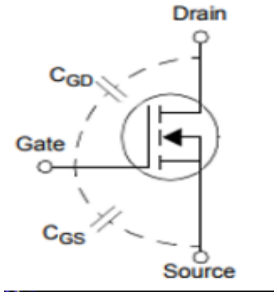
Og den forbliver on indtil ladningen fjernes ved at forbinde gaten til GND. Også bare gennem en finger.

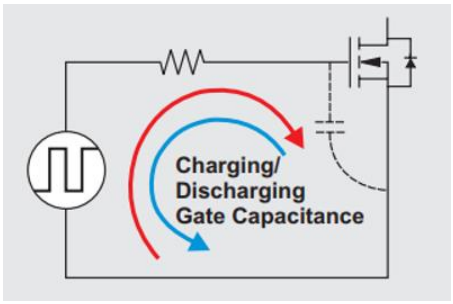


En MOSFet transistor har godt nok det, der umiddelbart ligner en uendelig stor indgangsmodstand.

Men i dens indre, på dens chip - optræder der nogle små kondensatorer, hhv. mellem Gate og Drain og mellem Gate og Source.







Spændingen på disse kondensatorer skal jo ændres i hvert skift.

Dvs. der skal løbe en strøm for at oplade / aflade kondensatorerne.

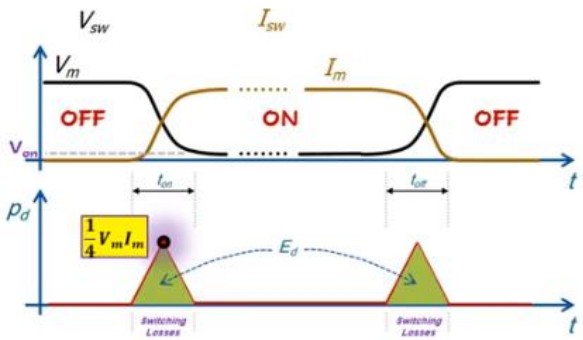
Og hvis skiftefrekvensen bliver for høj, vil strømmen jo også blive stor.

Gatestrømmen er jo begrænset af udgangsmodstanden i driverkredsløbet.

Gatestrømmen må være bestemt af $Q_G = I \times t$.
Derfor, hvis der skal switches on på kort tid, må strømmen være større.

Arduinoen kan jo ikke levere mere end 40 mA på en pin !

Og så er det desuden sådan, at hvis ikke skiftet fra ”0” til ”1” på Gaten sker ” momentant” vil transistoren ikke ”momentant” skifte fra Off til ON. Og omvendt !!



Derfor vil transistoren i hver skift gå fra Off til On i ” løbet af noget tid ”.

Så fx når transistoren er ”halvt on” leder den halvt, og der er et spændingsfald over den fra Drain til Source.

Det betydet også, at der en effektafsætning i selve transistoren. – Den bliver varm.

Kilde: https://www.youtube.com/watch?v=4eM3SFQGp-g&ab_channel=WalidIssaPlus

$$P = \Delta U \cdot I \; [Watt]$$

Overvej evt. brugen af en ” Gate Driver ”

Derfor er det en god ide at lade timebasen være så lang som muligt, og dermed en switch-frekvens så lav som muligt, - ud fra de fysiske forhold.

Overvej nogle funktioner, der kan styres via PWM, og overvej deres max timebase.

Alt i alt: Hvis man skal bruge en PWM-funktion til fx at styre en skruemaskine, bør frekvensen ikke være for stor.

Manuel opsætning af PWM i Arduino

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 betingelser.

Vi brugte mode 4 af de 16 modes, timeren kan opsættes til.

Det fremgår af skemaet her:

Timer1 mode select:

Ud fra dette skema ses de forskellige modes, der kan vælges:

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

I det følgende gennemgås PWM mode 15.

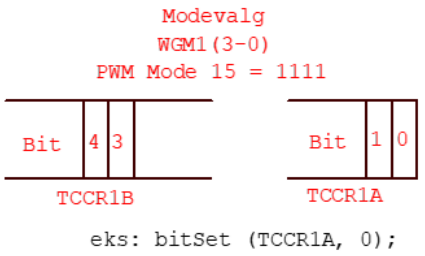
De 4 Wave Generation Mode bit skal sættes til 1111.

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

Mode 15, PWM, vælges ved at sætte 4 bit. Og lidt mærkeligt, i to forskellige registre !!

I Register TCCR1A bruges bit 0 og 1, og i TCCR1B bit 3 og 4.

Alle 4 bit skal sættes høje.



Mode 15, = 1111

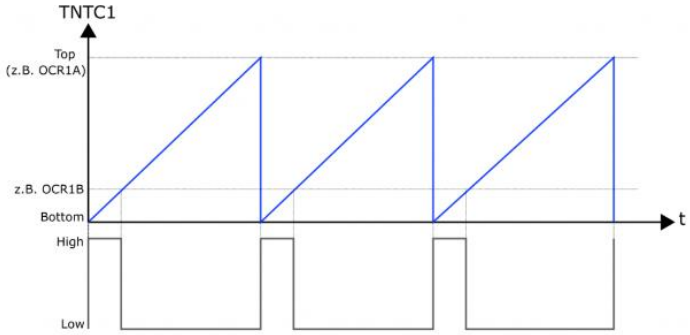
PWM MODE 15

Mode 15 kaldes også Fast PWM. Ved Fast PWM er princippet, at en tæller, - her tæller1, - tælles op fra 0x0000 indtil en bestemt værdi, hvorefter tælleren resetter, og starter forfra igen.

(I de ”langsomme” – dvs. ikke Fast Modes - tæller tælleren op fra 0x0000 til en værdi og derefter tælles ned igen til 0x0000.)

Først Princippet kort:

Tælleren Timer1 tæller opad. Den får pulser fra krystallet, evt. neddelt af prescaleren. Den blå graf skal opfattes som tællerens ”værdi” efterhånden som den tæller opad.



Skitse af funktionen i mode 15.

Nederst er vist signalet på outputpin 10.

Når tælleren når op på en bestemt værdi, som er sat ind i et Compare-register, OCR1B, påvirkes en outputpin, pin 10. (bliver fx lav).

Og når tælleren når op til værdien lig med indholdet i register OCR1A, påvirkes pin 10 igen, og samtidig resettes tælleren.

Herefter gentages forløbet.
Kilde: https://wolles-elektronikkiste.de/en/timer-and-pwm-part-2-16-bit-timer1

Timebase kan indstilles ved at vælge en prescaler, og ved at bestemme værdien timer1 skal tælle op til før den resettes.

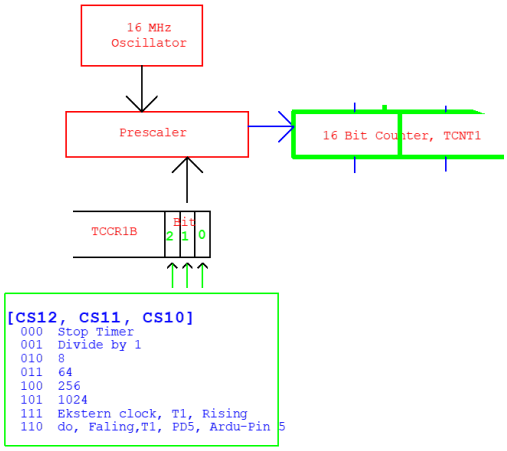
Pulsbredden kan så ændres løbende ved at ændre værdien i Compare-register OCR1B. Det kan ske i loop()-funktionen.

Mere forklaring: Fast PWM, timer1, mode 15.

Timeren skal tælle krystallets frekvens, der jo er 16 MHz. Så det går rimeligt hurtigt !

Men heldigvis er der mulighed for at neddele frekvensen vha. en Prescaler.

Prescaleren kan indstilles til at dele med 1, 8, 64, 256 eller 1024.



Prescalerens frekvensdeling kan styres ved at sætte bit 2, 1 og 0 i dens kontrolregister, Timer Counter Control Register 1B, TCCR1B.

Nederst i skitsen ses, hvilke bit, der skal sættes, for valg af ønsket prescaler-værdi.

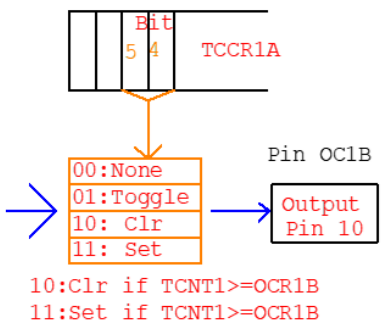
Man kan fx bruge bitSet (TCCR1B, 0);

Via prescaleren sendes frekvensen til Timer1, Timer/Counter Register 1, TCNT1, der er på 16-bit. Dvs. den kan tælle op til max 65.535, eller 0xFFFF

Undervejs i optællingen, opnås på et tidspunkt samme værdi som er lagt ind i register OCR1B. Dette bevirker, at output pin 10 påvirkes.

Tælleren fortsætter op til tallet, lagt ind i register OCR1A.

Når denne værdi nås, resetter tælleren, og starter forfra ved 0x0000. – Og output pin 10 påvirkes igen.



Når der menes at pin 10 påvirkes, betyder det, at man kan indstille, hvordan påvirkningen skal være.

Bit 7 og 6 i register TCCR1A, Timer/Counter Register 1A, bestemmer hvad der skal ske på pin 10.

00: Ingenting sker
01: Toggle
10: Clr hvis TCNT1 >= OCR1A
11: Set hvis TCNT1 <= OCR1A

Ved nu at vælge den rigtige prescaler, - og ved at indsætte en værdi i register OCR1A, - kan man indstille en passende **Timebase** for den funktion, man skal kontrollere. Det sker typisk i opsætningen !

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

Det kan fx ske efter løbende læsning af et potmeter koblet til en analog indgang. Eller læsning af et temperatursignal ? – Eller ???

Ang. Valg af Timebase og prescaler, se senere ??

Valg af Timebase

Det hele kræver selvfølgelig noget opsætning – og udregninger. Her et eksempel:

Eks: **Prescaler = 8:**

Vælges fx en **prescaler på 8**, får tælleren 16 – 8 = 2 MHz.

Maks værdi, der kan tælles med 16 bit er 65.535 (= 0xFFFF)

Tælleren resettes derfor efter $\frac{65535}{2E6}$ = 0,033 sek. = 33 mS.

Samme beregning for øvrige prescalerværdier giver følgende skema:

På skemaform ser det sådan ud:

Med Prescaler	Fås en frekvens til tælleren [Hz]	Timebase (max) [S]	Svarende til 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

Men værdien i OCR1A behøver jo ikke være på 65.535 (0xFFFF).

Vælges den mindre, bliver timebasen jo tilsvarende kortere.

Det kan beregnes af følgende:

$$Timebase\ [s] = \frac{(\ OCR1A) \cdot (Valgt\ Prescaler)}{16E6\ [Hz]}$$

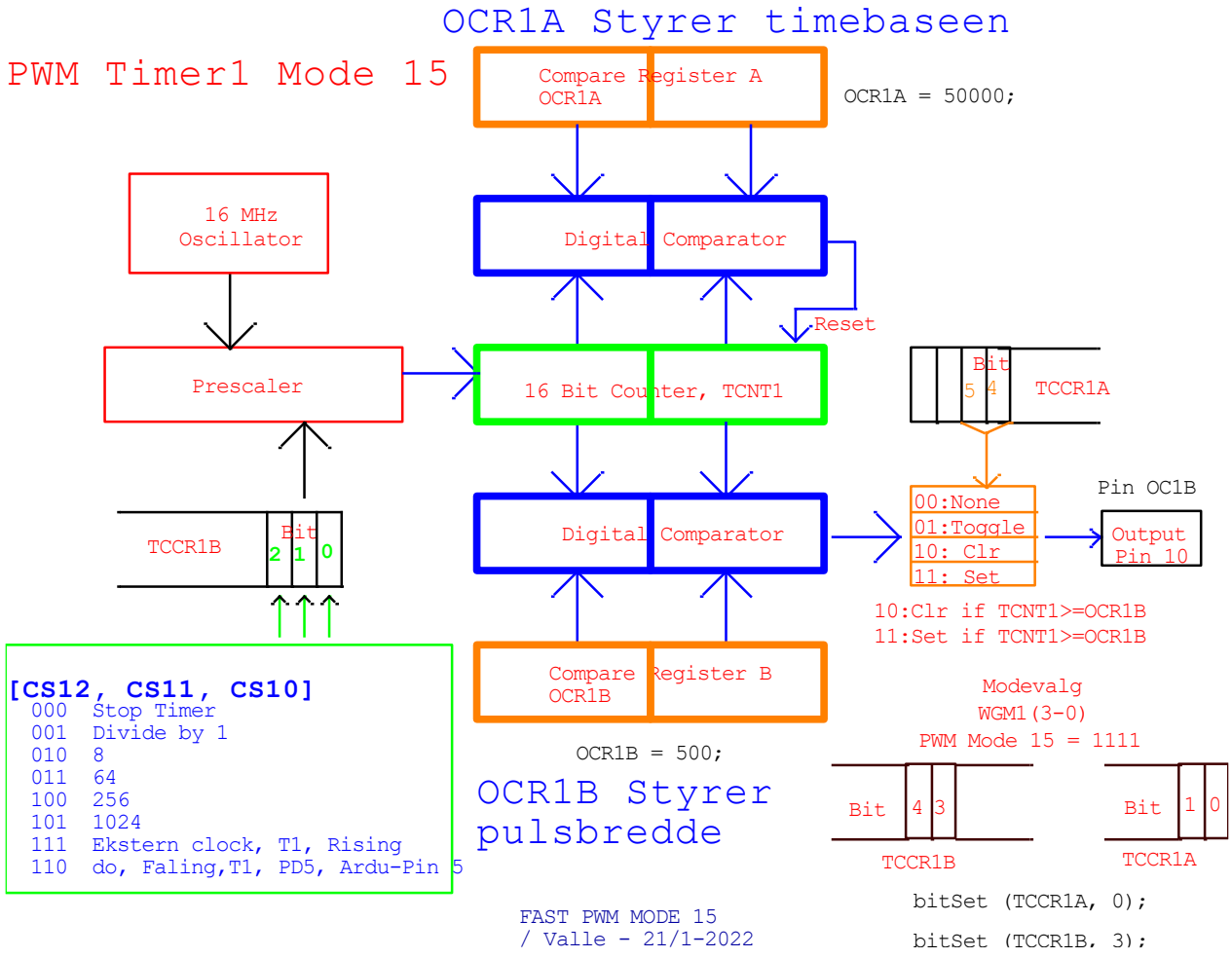
Og derfor for en given ønsket timebase findes OCR1A – værdien af:

$$OCR1A = \frac{Timebase\ [s] \cdot 16E6\ [Hz]}{(\ Valgt\ Prescaler)}$$

Her må man lige være opmærksom på, at OCR1A-værdien jo selvfølgelig ikke kan være højere end 65.535, da det er det største tal, der kan være i 16-bit registeret. Bliver den det, må der vælges en større prescaler!!

Her har jeg forsøgt at lave en oversigt over de forskellige registre og hvordan de indstilles:

Der mangler desværre en pil fra den øverste Digital Comparator og ned til Waveform generatoren før pin 10.



Men det næste er så, hvordan man får det hele indstillet i de relevante registre.

Det er ret kompliceret. Derfor følger her en subrutine, der kan kopieres og placeres efter loop() i ens sketch.

Ideen er at man udkommenterer de linjer, der ikke skal bruges ved opsætningen af PWM-funktionen.

Funktionen skal kaldes i setup().

Funktion til opsætning af Timer1 som PWM generator:

```
void init_timer1_pwm()  // OBS: Mode 15.  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, tæller til OCR1A
  bitSet(TCCR1B, 3);
  bitSet(TCCR1A, 1);
  bitSet(TCCR1A, 0);

// Vælg Pin-10 output funktionsmåde., Compare-B match med OCR1B

  bitSet(TCCR1A, 5);  // 00=none, 01=Toggle, 10=Clear,
  bitSet(TCCR1A, 4);  // 11=Set. Modsat fase af ”10”

// 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 Compareværdi ( eksempler )

  OCR1B = 1000;  //  Pin 10 output skift-værdi
  OCR1A = 20000; // Top, Timer1-resetværdi i mode 15

// enable pin som output:
  pinMode (10, OUTPUT);  // Compare-B-output-Pin
} // End-timer1-PWM-Setup

// Valle
```

Start og stop af PWM-funktionen

PWM-funktionen, styres af to bit, bit 5 & 4 i register TCCR1A.
Sættes de til ”00”, er der ingen PWM-signal på udgangen, og udgangen er 0. !!

Derfor, ved bitSet(TCCR1A, 5); tænder man for PWM på pin 10. Og bitClear(TCCR1A, 5); stopper funktionen med Pin10 Lav.

Bilag

Opsætning af timerer1 til PWM:

I maven på ATMEGA328P er der flere Special Function Registre, - SFR, - 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 "Wave Generation Mode Bit" - WGM1(0-3)

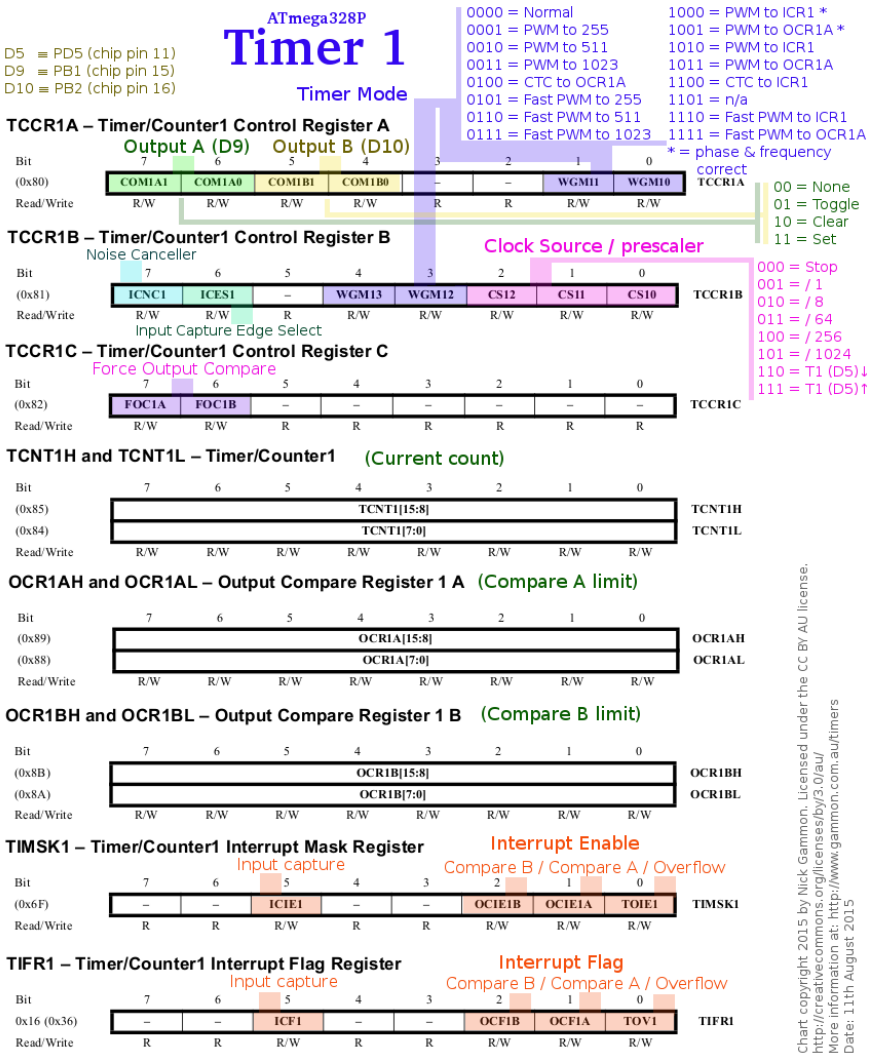
De er placeret i hhv. register TCCR1A og TCCR1B.

Alle 4 bit skal sættes for at vælge mode 15.

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

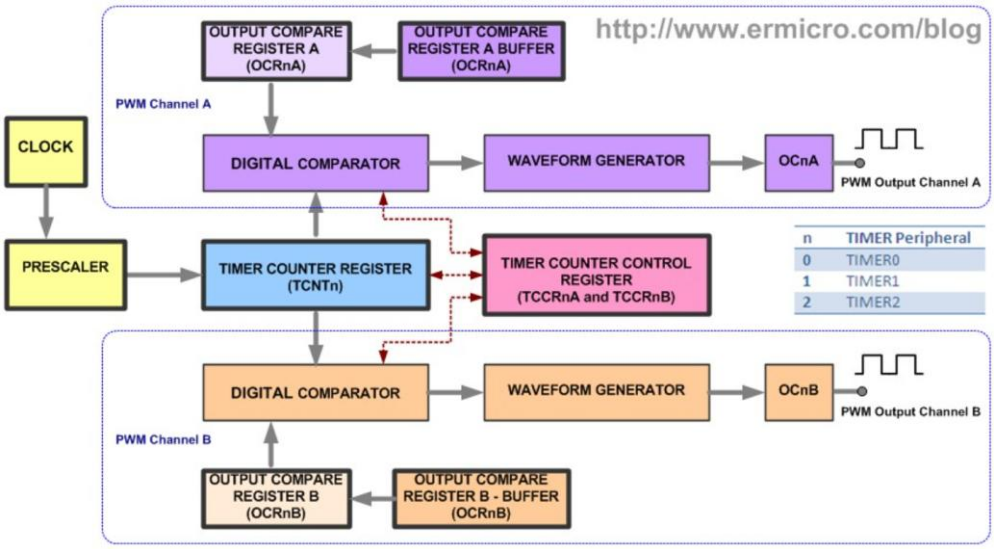
De sættes til Clear, = 10, som betyder at output Pin10 cleares når tællerens værdi er lig med compareværdien.

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:

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

Forsøgsopstilling.

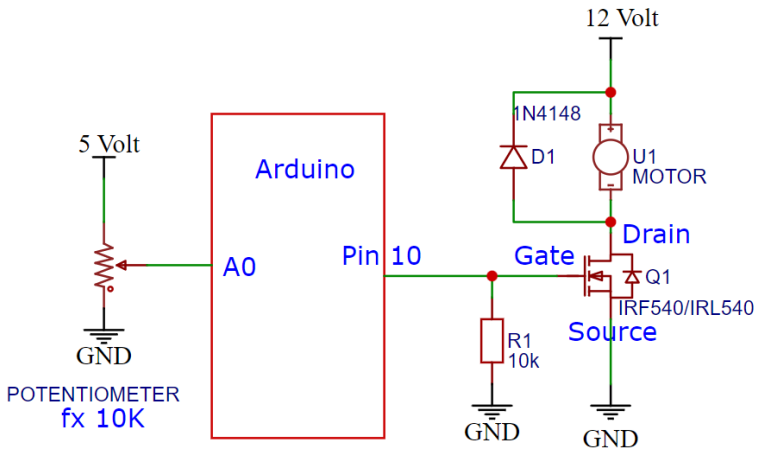
Her bruges et potmeter til at ændre spændingen mellem 0 og 5 Volt på den analoge indgang A0.
I loop() læses A0, og værdien tilpasses og sættes ind i register OCR1B

R1 sidder der for at holde transistorens Gate 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:

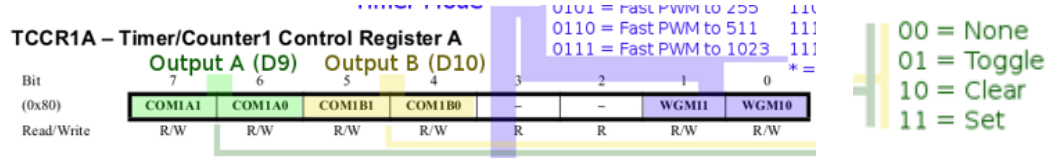
Mode 14 til PWM.

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 tæller timeren op til top, compare match med register ICR1. Det er også en 16 bit-register.

Derfor kan man bruge både OCR1A og OCR1B til PWM-funktioner.

Til at styre en H-bro kunne det være godt at have to uafhængige PWM-pins. Det kan indstilles i Mode 14.



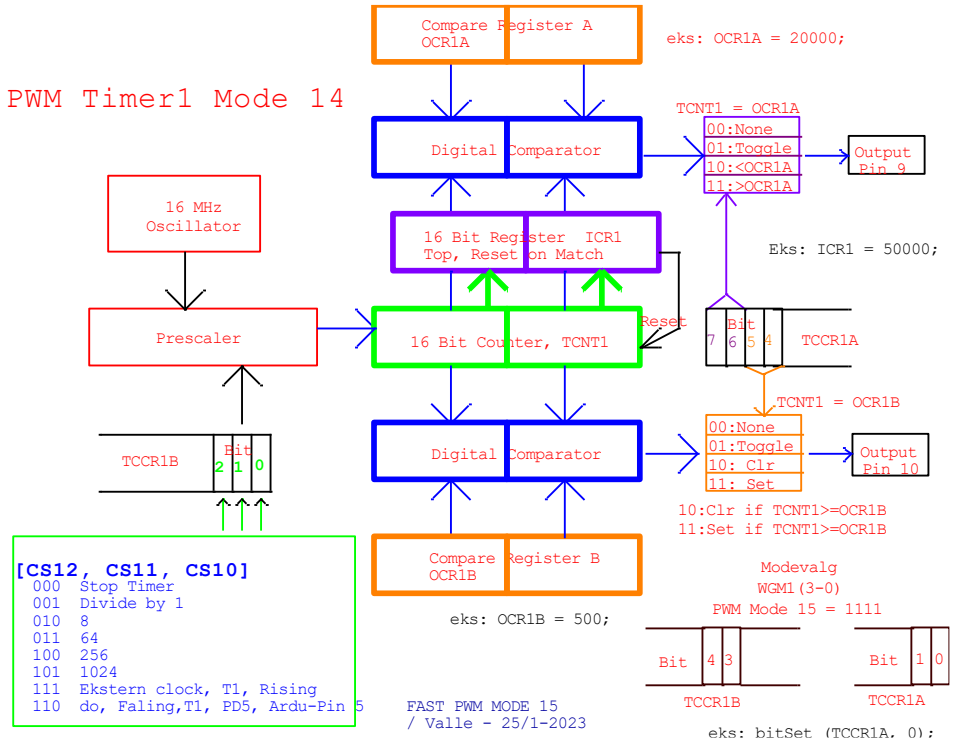
Tællerens top indsættes i register ICR1. Det er egentlig et register, der bruges til Input Capture, !!!

Output dutycycle på pin 9 styres af værdien i register OCR1A og dens opførsel styres af bit 7 og 6 i register TCCR1A.

Og tilsvarende,

Output dutycycle på pin 10 styres af værdien i register OCR1B og dens opførsel styres af bit 5 og 4, også i register TCCR1A.

Opsætning af timer1 til PWM mode 14:

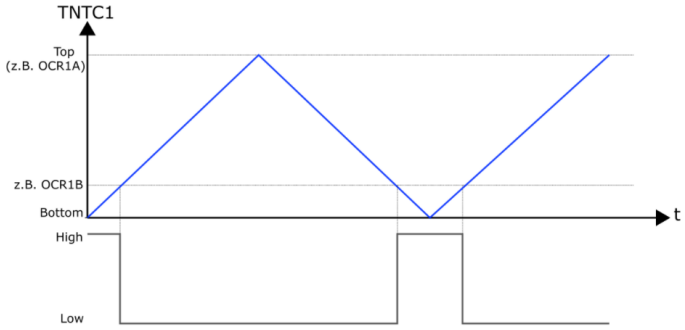


Mode 8, 9, 10 & 11

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 som ved Fast PWM.**

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.

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/