MTD2A_binary_output

MTD2A: Model **T**rain **D**etection **A**nd **A**ction – **Arduino** library https://github.com/MTD2A/MTD2A
Jørgen Bo Madsen / V1.3.3 / 17-10-2026

Model Train Detection And Action er en samling af brugervenlige, avancerede og funktionelle C++ byggeklodser til tidsstyret håndtering af input og output ved hjælp af et tilstandsmaskinesystem til parallel behandling. Biblioteket er beregnet til Arduino-entusiaster uden megen programmeringserfaring, der er interesserede i elektronisk styring og automatisering, samt modeltog.

Fælles for alle byggeklodser:

- Bygger på et tilstandsmaskine system til parallel processering og synkron tidsstyring
- Understøtter en bred vifte af inputsensorer og outputenheder
- Er enkle at bruge til at bygge komplekse løsninger med få kommandoer
- Fungerer ikke-blokerende, procesorienterede og tilstandsdrevne
- Tilbyder omfattende information om styring og fejlfinding
- Grundigt dokumenteret med mange eksempler

Indholdsfortegnelse

MTD2A_binary_output 1
Funktionsbeskrivelse 1
Proces faser 3
Initialisering 4
Aktivering 4
Stop og reset (restart) timer 5
outputValue 6
PWM kurver 7
Skalering 14
outputValue og PWM 15
Eksempler på configuration 17
Set og get funktioner 18
print_conf(); 19

Funktionsbeskrivelse

MTD2A_binary_output processen består af 4 funktioner:

1. `MTD2A_binary_output object_name (”object_name”, outputTimeMS, beginTimeMS, endTimeMS, outputMode, beginValue, endValue, loopActivate, ActivateNumber);`

2. `object_name.initialize ( pinNumber, pinWriteMode, startPinValue );`
   Kaldes en gang i `void setup ();` og efter `Serial.begin (9600);`

3. `object_name.activate ();` Aktiverer en gang. Kan genaktiveres når processen er afsluttet (COMPLETE)

4. `MTD2A_loop_execute ();` Kaldes som det sidste I `void loop ();`

Alle funktioner benytter default værdier og kan derfor kaldes med ingen og op til ni parametre.
Dog skal parametre angives i stigende rækkefølge startende fra den første. Se eksempel herunder:

```
MTD2A_binary_output object_name;
MTD2A_binary_output object_name
  (”object_name”);
  ("o n" , outputTimeMS);
  ("o n" , outputTimeMS, beginTimeMS);
  ("o n" , outputTimeMS, beginTimeMS, endTimeMS);
  ("o n" , outputTimeMS, beginTimeMS, endTimeMS, outputMode);
  ("o n" , outputTimeMS, beginTimeMS, endTimeMS, outputMode, beginValue);
  ("o n" , outputTimeMS, beginTimeMS, endTimeMS, outputMode, beginValue, endValue);
  ("o n" , outputTimeMS, beginTimeMS, endTimeMS, outputMode, beginValue, endValue, loopActivate, ActivateNumer);
```

Defaults:

```
  ("o n" , 0           , 0           , 0           , BINARY    , HIGH      , LOW     , DISABLE    , 0);
```

Eksempel

```
// Two blinking LEDs. One with symmetric interval and another with asymetric interval.

 #include <MTD2A.h>
 using namespace MTD2A_const;

 MTD2A_binary_output red_LED   ("Red LED",   400, 400);  // 0.4 sec light, 0.4 sec no light
 MTD2A_binary_output green_LED ("Green LED", 300, 700, 0, P_W_M, 96);  // 0,3 / 0.7 sec PWM
dimmed

 void setup() {
   Serial.begin(9600);
   while (!Serial) { delay(10); } // ESP32 Serial Monitor ready delay

   byte RED_LED_PIN   = 9;  // Arduino board pin number
   byte GREEN_LED_PIN = 10; // Arduino board pin number
   red_LED.initialize   (RED_LED_PIN);
   green_LED.initialize (GREEN_LED_PIN);
   Serial.println("Two LED blink");
 }

 void loop() {
   if (red_LED.get_processState()   == COMPLETE) {
     red_LED.activate();
   }
   if (green_LED.get_processState() == COMPLETE) {
     green_LED.activate();
   }

   MTD2A_loop_execute();
 }
```

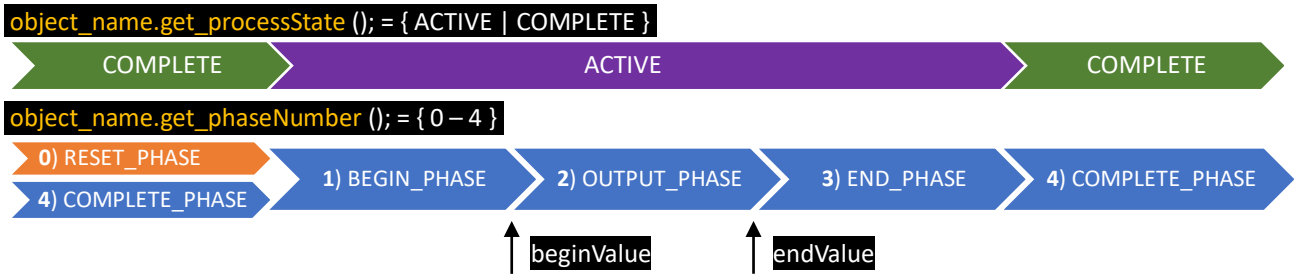
Flere eksempler og youtube video:

https://github.com/MTD2A/MTD2A/tree/main/examples

DEMO video: https://youtu.be/vLySY92JdAM

Proces faser

Afhængig af den aktuelle konfiguration gennemføres processen i mellem 1 og 5 faser.



0. Den initielle fase når programmet starter samt når funktion reset (); kaldes eller 4) COMPLETE_PHASE.
1. Start forsinkelse. Hvis sat til 0 eller ikke defineret springes fasen over.
2. Output tidsperiode. Starter med beginValue og slutter med endValue.
   Hvis sat til 0 eller ikke defineret springes fasen over, og der skrives en advarsel.
3. Slut forsinkelse. Hvis sat til 0 eller ikke defineret springes fasen over.
4. Processen er afsluttet COMPLETE og klar til ny aktivering object_name.activate ();

Globale nummerkonstanter: RESET_PHASE, BEGIN_PHASE, OUTPUT_PHASE, END_PHASE & COMPLETE_PHASE

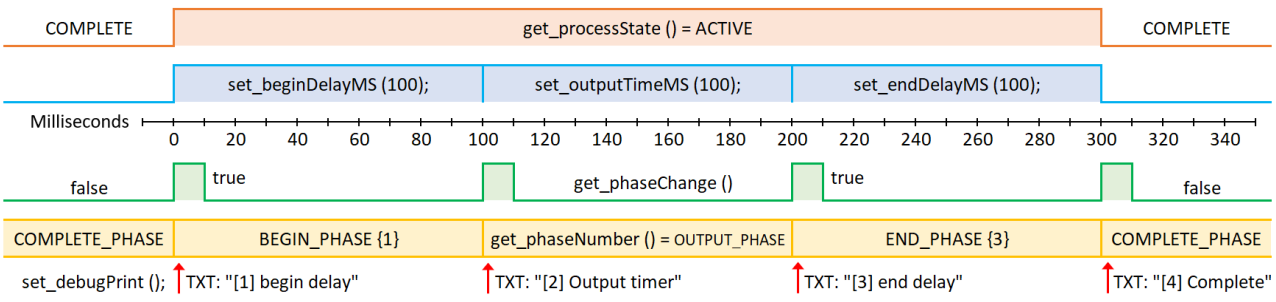
Proces status og debug information

Ved overgang til BEGIN_PHASE, OUTPUT_PHASE eller END_PHASE skifter ProcessState til ACTIVE.
Ved overgang til COMPLETE_PHASE eller RESET_PHASE skifter processState til COMPLETE.

Det **øjeblikkelige** faseskift kan identificeres med funktion: object_name.get_phaseChange (); = { true | false }
Statusinformation kan udskrives med funktionen: object_name.set_debugPrint (); = { ENABLE | DISABLE }

Eksempel på forskellige konfigurationsmetoder

```
MTD2A_binary_output relay ("Relay", 100, 100, 100, BINARY, HIGH, LOW);
// is equivalent to
MTD2A_binary_output relay ("Relay", 100, 100, 100); // default: BINARY, HIGH, LOW
// is equivalent to
MTD2A_binary_output relay ("Relay"); // default: 0, 0, 0, BINARY, HIGH, LOW
relay.set_outputTimeMS (100);
relay.set_beginDelayMS (100);
relay.set_endDelayMS   (100);
// Show phase change information
relay.set_debugPrint   (); // default enable
```



Timing

Se dokumentet MTD2A.pdf og afsnittet ”Kadence ” og ”Synkronisering” samt ”Eksekveringshastighed”.

Initialisering

object_name.initialize (pinNumber, pinWriteMode, startPinValue); Kaldes en gang i void setup (); og efter Serial.begin (9600); Dette er vigtigt for at printe mulige fejl og advarsler, herunder fra tidligere instatiering af objekter.
Output skrives til det digitale benforbindelsesnummer, der er specificeret i object_name.initialize (pinNumber);
Angives der en benforbindelser der ikke understøttes, bliver benforbindelsen ikke initialiseret, og der kan **ikke** skrives til benforbindelsen efterfølgende: pinWriteToggl = DISABLE og pinNumber = NO_PIN
object_name.initialize (); svarer til object_name.initialize (**NO_PIN**); NO_PIN = 255.

Output på benforbindelsen kan inverteres (omvendes) ved at sætte pinWriteMode til INVERTED
Det betyder at der byttes om på HIGH og LOW og PWM værdi regnes til 255 minus PWM værdi.
object_name.initialize (pinNumber, {NORMAL | **INVERTED**});

Benforbindelse initialiseres med den værdi der angivet i startPinValue
startPinValue skrives **øjeblikkeligt** til pinNumber Udelades parameteren angives LOW

object_name.initialize ({ pinNumber | **NO_PIN** }, {**NORMAL** | INVERTED}, **startPinValue** | **LOW**);

Det er muligt løbende at styre om der skal skrives til benforbindelsen eller ej med funktionen:
object_name.set_pinWriteToggl ({**ENABLE** | DISABLE});

Det er også muligt at skrive direkte til benforbindelsen med funktionenerne:
object_name.set_pinWriteValue (PinValue); binary {HIGH | LOW} / PWM {0-255}
object_name.set_pinWriteValue (PinValue, {BINARY | P_W_M); Valgfri parameter. Ingen default værdi.
Dette sker **øjeblikkeligt**, uanset om processen er aktiv eller ej, og uafhængigt af MTD2A_loop_execute ();

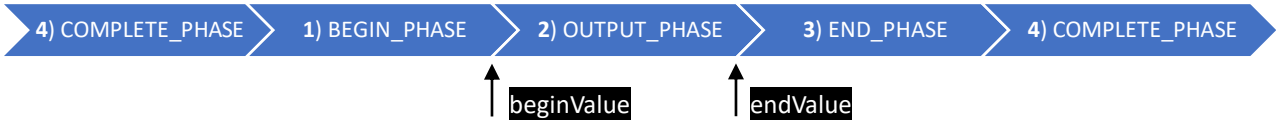
Som udgangspunkt er benforbindelsen **udefineret**. Se Digital Pins | Arduino Documentation

Aktivering

Processen aktiveres med funktionen: object_name.activate (); Dermed skifter procesState til ACTIVE
Så snart procesState skifter til COMPLETE kan processen aktiveres igen.

Processen kan til hver en tid nulstilles med funktionen: object_name.reset (); Funktionen nultiller alle styrings- og process variable **øjeblikkeligt**, før MTD2A_loop_execute (); og gør klar til ny start. Alle funktionskonfigurerede variable og standard værdier bibeholdes. Procesfasen skifter til RESET_PHASE
Hvis benforbindelsen er defineret korrekt, skrives startPinValue til benforbindelsen **øjeblikkeligt**.

Der er muligt at skrive begyndelseværdier beginValue og slut værdier endValue til benforbindelsen.



Activate funktioner benytter "function overloading". Det betyder at funktionen kan kaldes med ingen og op til 4 parametre. Dog skal parametre angives i stigende rækkefølge startende fra den første.

Der benyttes **ikke** default værder. Alle eksisterende værdier forbliver de samme, med mindre de angives som parametre i funktionskaldet. Se eksempel herunder:

object_name.activate ();
object_name.activate (beginValue);
object_name.activate (beginValue, endValue);
object_name.activate (beginValue, endValue, PWMcurveType);
object_name.activate (beginValue, endValue, PWMcurveType, outputTimeMS);

PWMcurveType angiver hvilken kurve der skal følges i tidsrummet outputTimeMS Kurven starter med værdien beginValue og slutter med værdien endValue.
outputTimeMS angiver tidsrummet mellem beginValue og endValue i millisekunder.

activate (); kan konfigueres til blive gentaget i det uendelige med funktionen:
object_name.set_loopActivate ({ENABLE | DISABLE}); Gentager processen der efterfølgende startes med activate (); Denne funktion er særligt egnet til blinkende LED og hvor der er behov for et konstant pulserende output. Det er muligt at tilføje, hvor mange gange activate (); skal gentages, ved at angive antallet i ActivateNumber. Nul betyder uendeligt. set_loopActivate ({ ENABLE | DISABLE }, ActivateNumber);

Når activate (); looping er aktiveret, gennemløber processen alle faser get_phaseNumber (); i hver cyklus, inklusive fasen COMPLETE_PHASE Complete-fasen varer en fuld cyklus, hvorefter næste cyklus starter med det samme — cyklustiden påvirkes ikke.

get_processState (); forbliver ACTIVE under alle loop-cyklusser og returnerer først COMPLETE når processen stoppes object_name.loopActivate (DISABLE);, eller ved at activateCount har nået ActivateNumber. DISABLE accepteres på et hvilket som helst tidspunkt.

Hvis ActivateNumber er større end 0 tælles antallet af loop-cyklusser op i activateCount, og det øjebklikkelige antal kan aflæses med funktionen get_activateCounter ();

Stop og reset (restart) timer

Det er muligt at sætte det nyt start tidspunkt for aktuel timerproces, og stoppe aktuel timerproces i utide.
RESET_TIMER Sætter et nyt starttidspunkt som hentes fra den globalt synkroniserede tid globalSyncTimeMS
STOP_TIMER afbryder tidskontrol processen

```
object_name.set_outputTimer ( {STOP_TIMER | RESET_TIMER} );
object_name.set_beginTimer ( {STOP_TIMER | RESET_TIMER} );
object_name.set_endTimer ( {STOP_TIMER | RESET_TIMER} );
```

Det nye starttidspunkt hentes fra den globalt synkroniserede tid og kan aflæses med funktionen:
MTD2A::get_globalSyncTimeMS ();

Tidsperioden sættes som standard når objektet instantieres (aktiveres).
Det er muligt at definere nye tidsperioder for alle tre tidsperioder:

```
object_name.set_outputTimeMS ( {0 - MAX_TIME_MS } );
object_name.set_beginDelayMS ( {0 - MAX_TIME_MS } );
object_name.set_endDelayMS ( {0 - MAX_TIME_MS } );
```

object_name.set_timers samler all tre timere i en funktion. Der benyttes "function overloading". Det betyder at funktionen kan kaldes med en og op til 3 parametre. Dog skal parametre angives i stigende rækkefølge startende fra den første.

Der benyttes **ikke** default værder. Alle eksisterende værdier forbliver de samme, med mindre de angives som parametre i funktionskaldet. Se eksempel herunder:

```
object_name.set_timers (outputTimeMS);
object_name.set_timers (outputTimeMS, beginDelayMS);
object_name.set_timers (outputTimeMS, beginDelayMS, endDelayMS);
```

outputValue

Der skrives til outputValue og benforbindelsen (PIN) i samme millisekund MTD2A_loop_execute (); processen er startet. **Men** værdien kan først aflæse med object_name.get_outputValue (); efter at MTD2A_loop_execute (); er afsluttet, og dermed først ved det efterfølgende gennemløb.
Der er derfor to logiske statusflag der kan benyttes til at identificere hvornår der er data til aflæsning:

1. object_name.get_outputState (); Er true hver gang der er nye data der kan aflæses.
2. object_name.get_outputProcess (); Er true når data kan aflæses. Dvs. i hele output perioden.

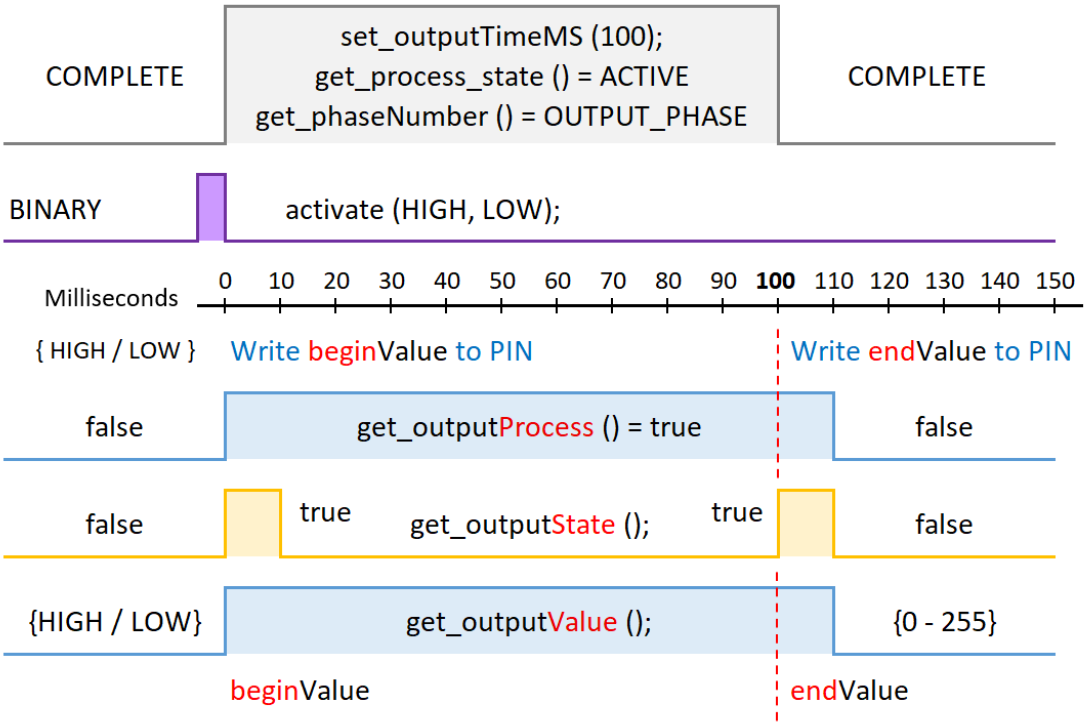
Output værdien til benforbindelsen angives uanset om benforbindelsen (PIN) er defineret eller ej. Se forklaring på initialisering uden angivelse af benforbindelse under afsnittet initialisering.

Den værdi der kan aflæses fra object_name.get_outputValue (); er altid den sidste angivne værdi.

NOTE: Hvis object_name.pinWriteMode er sat til INVERTED, er det den *inverterede* værdi— !værdi in BINARY mode og 255 – værdien in P_W_M mode — ikke den værdi der blev angivet i object_name.activate (); eller object_name.get_outputValue (); Med standardindstillingen object_name.pinWriteMode (NORMAL); er de to identiske.

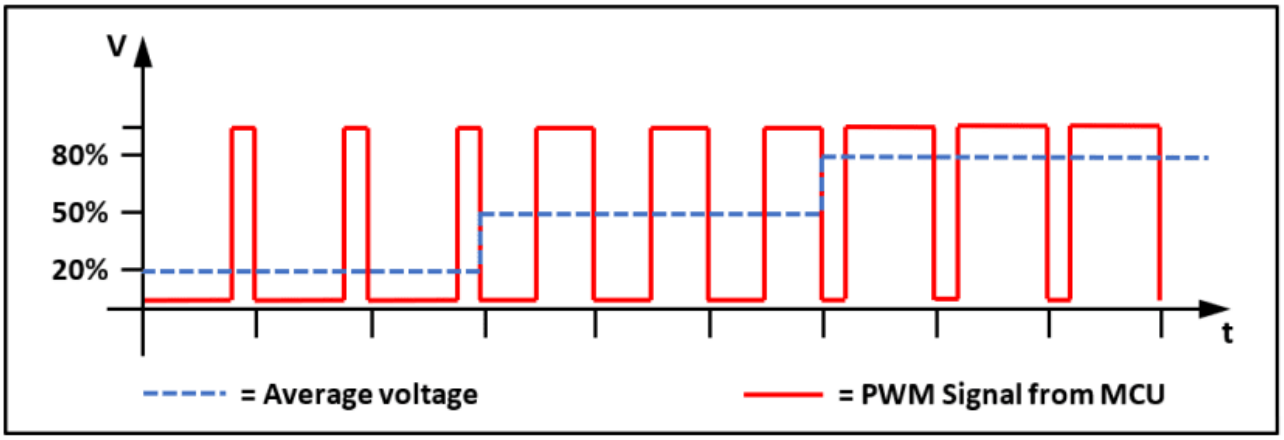
Eksempel:

```
if (relay.get_outputState() == true) {
  outputValue = relay.get_outputValue();
  Serial.print("PIN output value: "); Serial.println (outputValue);
}
```



PWM kurver

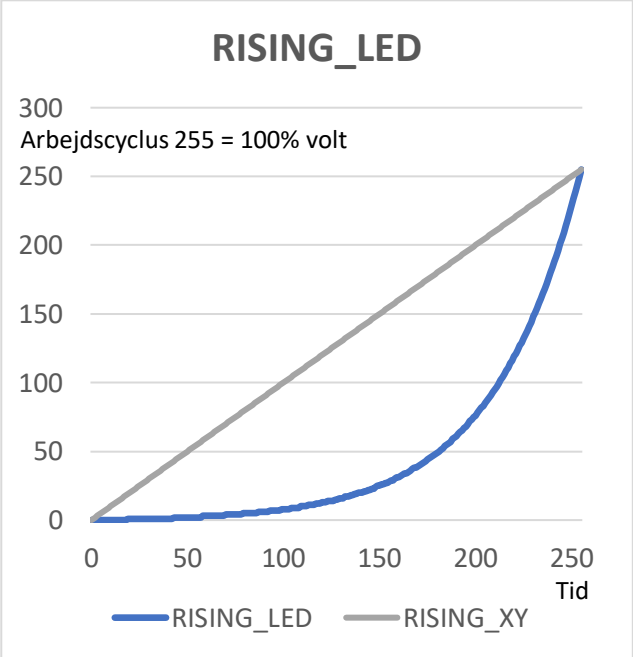
PWM (Pulse Width Modulation) er en teknik, der bruges til at kontrollere den gennemsnitlige effekt, der leveres til en elektrisk enhed ved at variere bredden af en række impulser. I bund og grund er det en digital metode til at skabe en analog effekt ved hurtigt at tænde og slukke for et signal

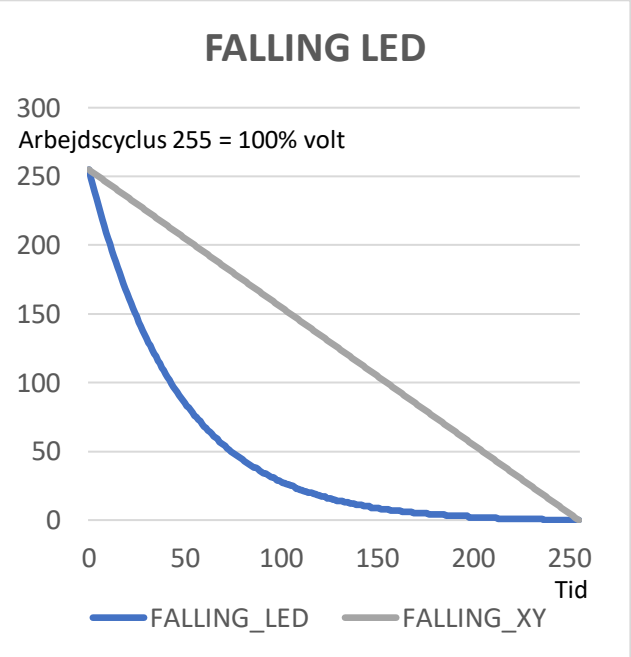


Der er designet 24 forskellige matematiske PWM-kurver: 2 lineære, 4 eksponentielle og 10 potens, 4 Bézier og 4 sinus funktioner. De er beregnet til at udjævne fysiske forhold, der har et ikke-lineært forhold mellem tid og funktion. F.eks acceleration og deceleration af DC-motorer (tog) og servo.

LED fading

PWM er en god måde at kontrollere lysstyrken på LED'er. Forholdet mellem arbejdscyklussen og den opfattede lysstyrke er dog slet ikke lineært. Mennesker opfatter lysstyrkeændringen ikke-lineært. Det menneskelige øje reagerer logaritmisk på lys, og har en bedre følsomhed ved lav luminans end høj luminans.





Arduino eksempel: MTD2A/examples/math_fade_LED at main · MTD2A/MTD2A
DEMO video: https://youtu.be/8TV6nOdXBno

Eksempel på glødepære blink

```
MTD2A_binary_output red_LED   ("Red LED", 400, 400);  // 0.4 sec light, 0.4 sec no light
bool switchUpDown = true, printFlag = true;

void setup() {
  Serial.begin(9600);
  while (!Serial) { delay(10); } // ESP32 Serial Monitor ready delay
  byte RED_LED_PIN   = 9;
  red_LED.initialize   (RED_LED_PIN);
}

void loop() {
  if (printFlag) {
    Serial.println (F("Red LED SOFT blink (like an incandescent bulb) using PWM curves"));
    printFlag = false;
  }

  if (switchUpDown) {
    if (red_LED.get_processState () == COMPLETE) {
      // Light up during 200 milliseconds and keep full lightning for 300 milliseconds
      red_LED.set_timers (200, 0, 300);
      red_LED.activate(MIN_PWM_VALUE, MAX_PWM_VALUE, RISING_LED);
      switchUpDown = false;
    }
  }
  else {
    if (red_LED.get_processState () == COMPLETE) {
      // Light down during 200 milliseconds and keep zero lightning for 300 milliseconds
      red_LED.set_timers (200, 0, 300);
      red_LED.activate(MAX_PWM_VALUE, MIN_PWM_VALUE, FALLING_LED);
      switchUpDown = true;
    }
  } // switchUpDown
  MTD2A_loop_execute();
} // loop
```

Arduino eksempel: [MTD2A/examples/blink_LED at main · MTD2A/MTD2A]
DEMO video: [https://youtu.be/nufAm2gE6pM]

potensfunktioner

Potensfunktioner er til udjævning af acceleration og deceleration.

For eksempel er brug af potensfunktioner velegnet til pendulkørsel med modeltog. Lokomotivet accelererer jævnt op i hastighed, kører med en fast hastighed i et stykke tid, deaccelererer jævnt ned i hastighed og stopper helt. Efter en kort pause gentages processen i den modsatte retning.

Inerti og friktion

Start og stop af tog (lokomotiv og vogne) involverer både inerti og friktion.

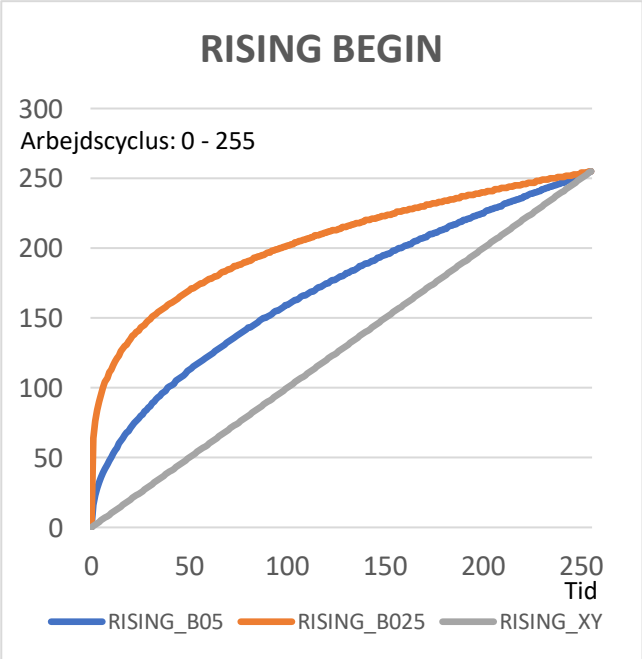
Inerti er egenskaben ved et objekt, der får det til at modstå ændringer i dets bevægelsestilstand. For at starte eller stoppe bevægelse er der brug for en ubalanceret kraft for at overvinde inerti og, i de fleste virkelige scenarier, friktion.

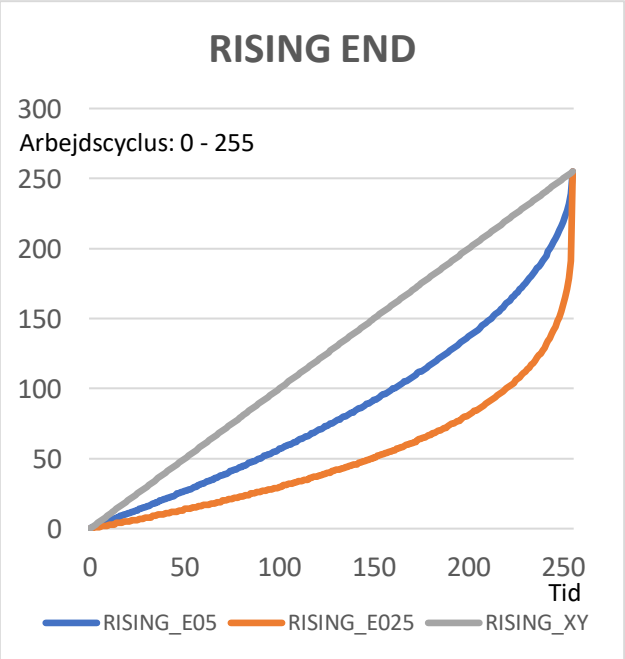
1. En bil, der accelererer, kræver en kraft for at overvinde bilens inerti, som ønsker at forblive stationær.
2. En rullende kugle stopper til sidst på grund af friktion. Uden friktion ville den fortsætte med at rulle.
3. Skubbe en kasse: Du skal skubbe med tilstrækkelig kraft til at overvinde statisk friktion og få kassen i bevægelse. Når du bevæger dig, skal du fortsætte med at anvende kraft for at overvinde kinetisk friktion og opretholde hastigheden.

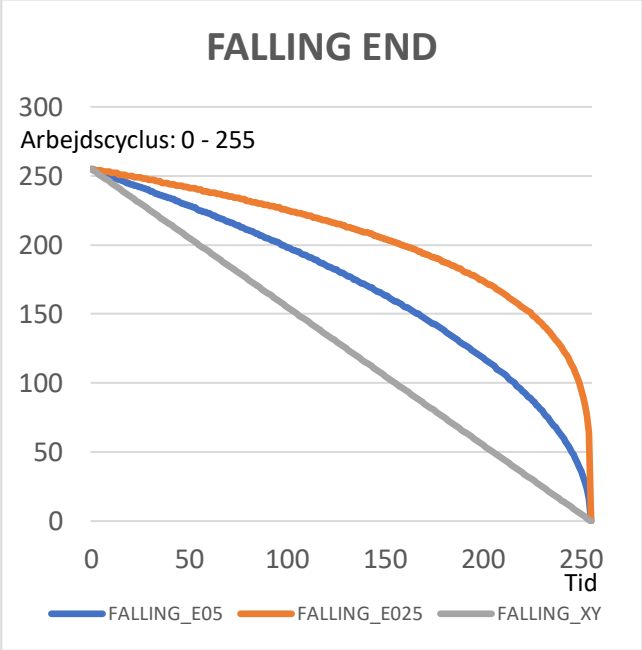
Inerti og friktion er grundlæggende begreber i fysik, der forklarer, hvorfor objekter bevæger sig (eller ikke bevæger sig), og hvordan bevægelse kan startes og stoppes. Forholdet mellem tid og funktion er **ikke lineært**.

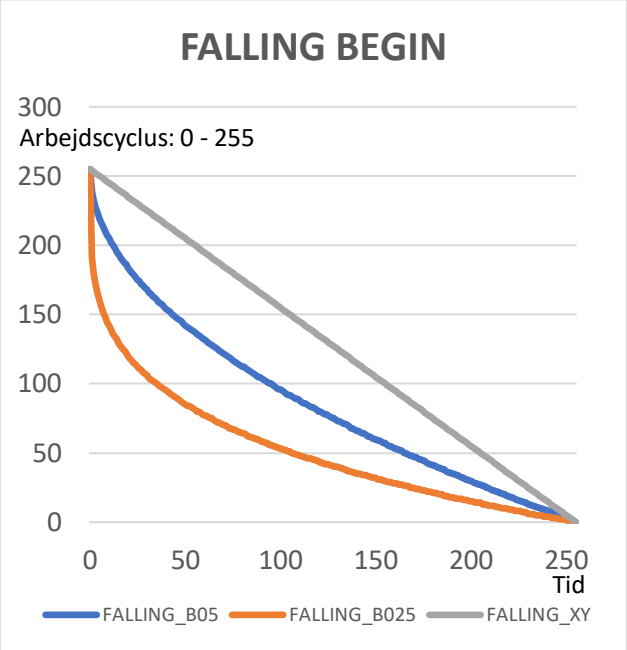
Desuden kan der være en effekt i jævnstrøms motorer kaldet "ankerreaktion". Afhængigt af hvordan motoren er fremstillet, kan der være en asymmetri mellem rotation frem og tilbage.

De følgende 8 effektfunktioner vil resultere i mere jævn togacceleration og -deceleration.





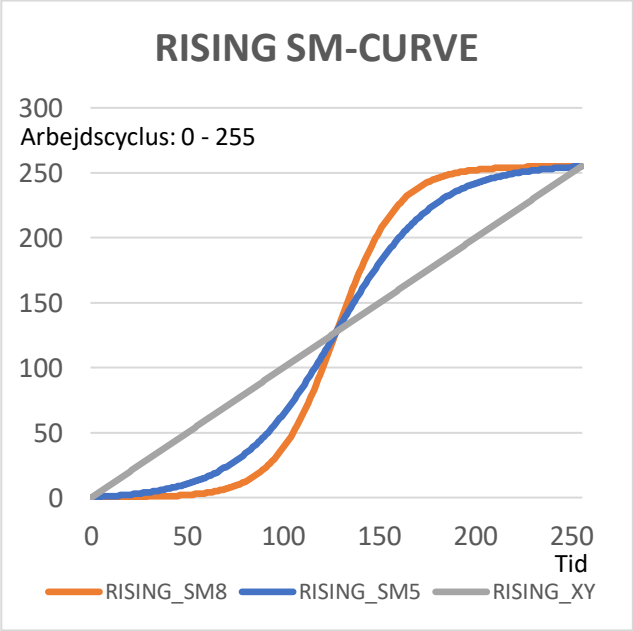


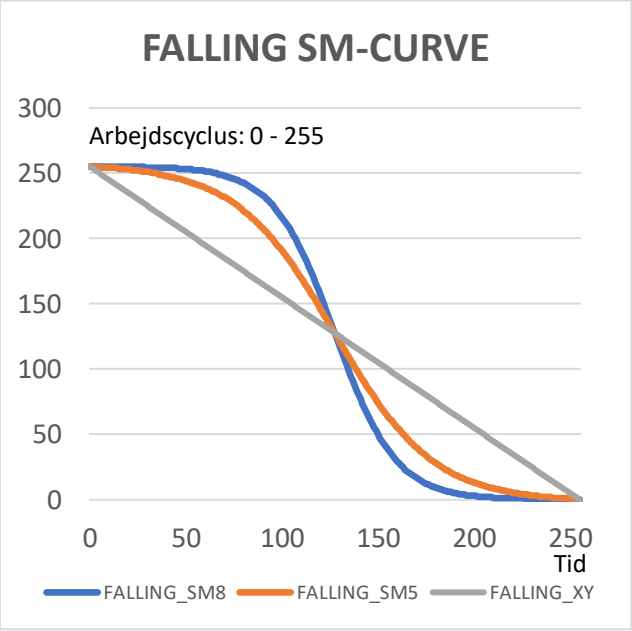


Arduino eksempel: [MTD2A/examples/PWM_power_curves at main · MTD2A/MTD2A]
DEMO video: https://youtu.be/Fi9D1hrzT9M

S-kurver

[Sigmoid S-kurver] Kan bruges, hvor blød acceleration og deceleration er at foretrække. For eksempel, at styre en (tungt belastet) servomotor fra punkt a til b, eller en blød start acceleration af et lokomotiv og en blød slut deceleration til forsat konstant hastighed.

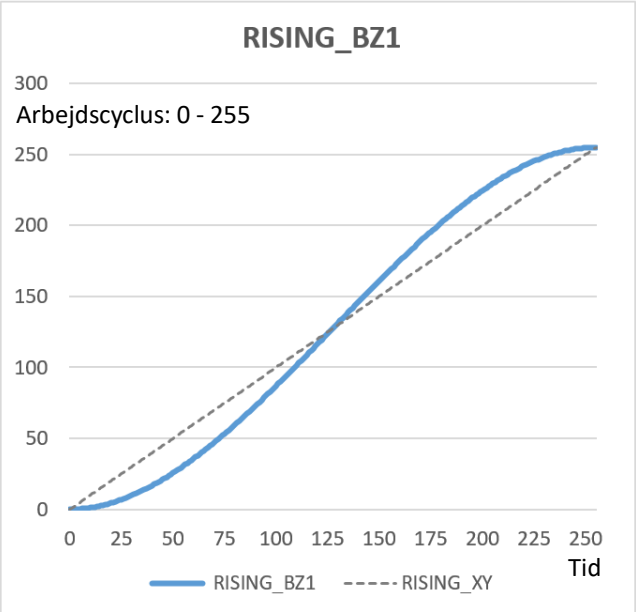


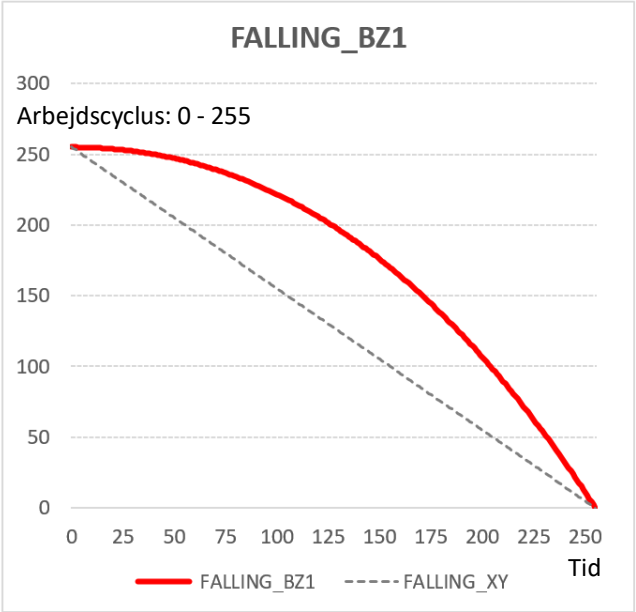


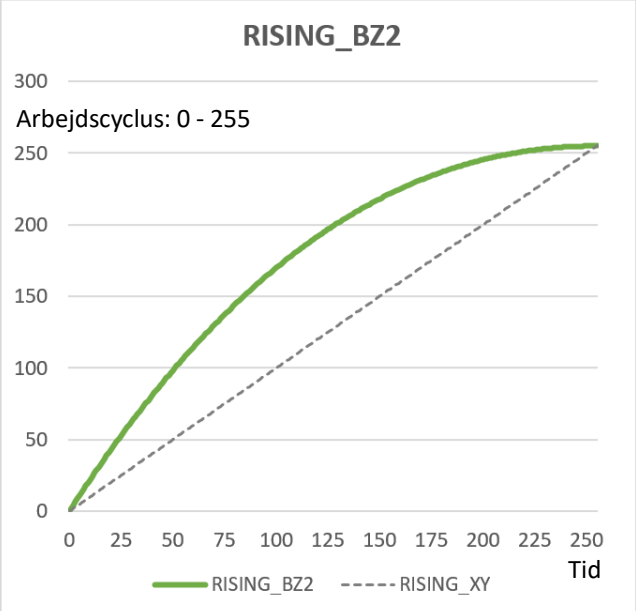
Arduino eksempel: [MTD2A/examples/servo_math_curve at main · MTD2A/MTD2A]
DEMO video: https://youtu.be/rhQtu0iKFl8

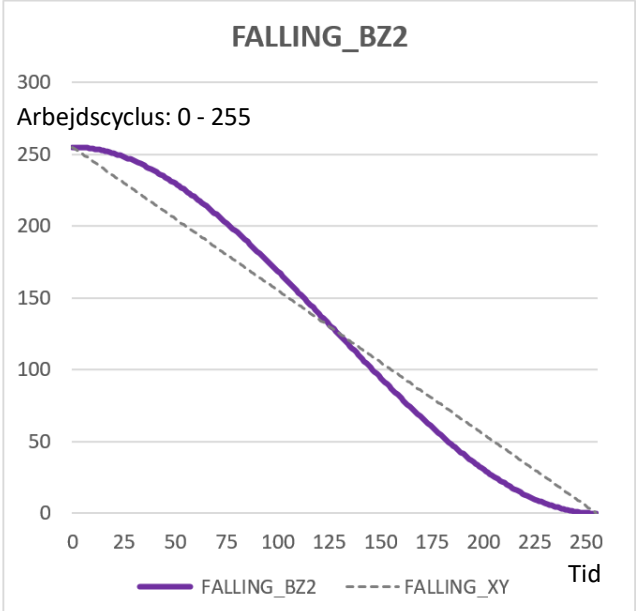
Bézier kurver

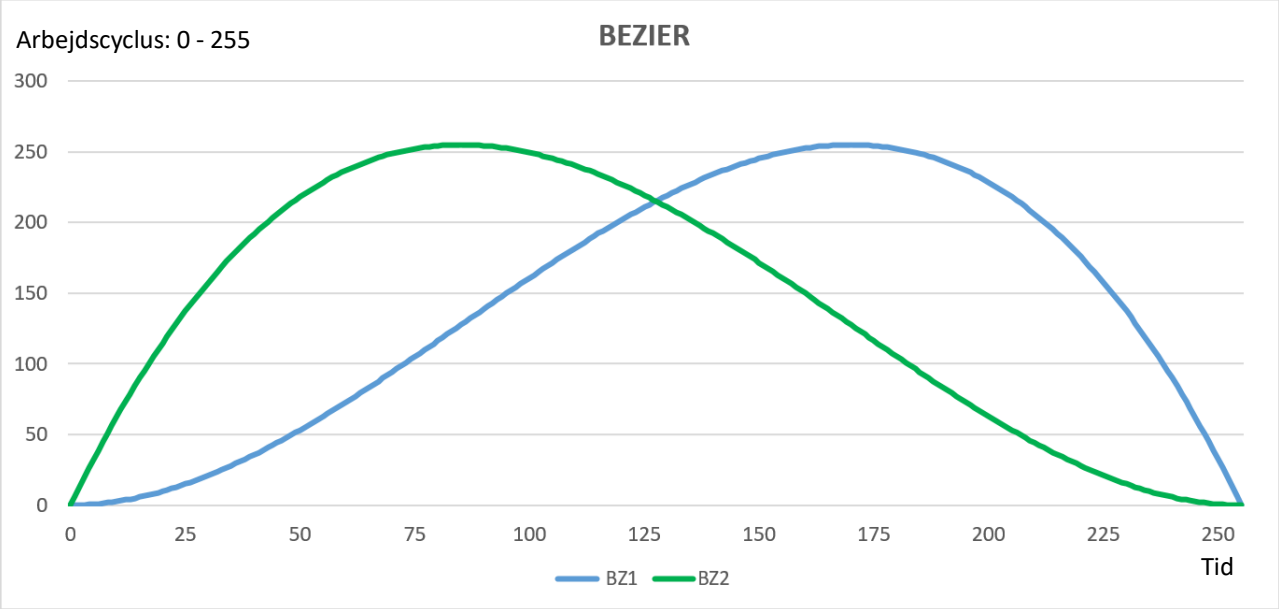
Bézier kurver minder om Sigmoid S-kurver og kan bruges, hvor en mere blød acceleration og deceleration er at foretrække. For eksempel, at styre en (tungt belastet) servomotor fra punkt a til b, eller en blød start acceleration af et lokomotiv og en blød slut deceleration til forsat konstant hastighed.







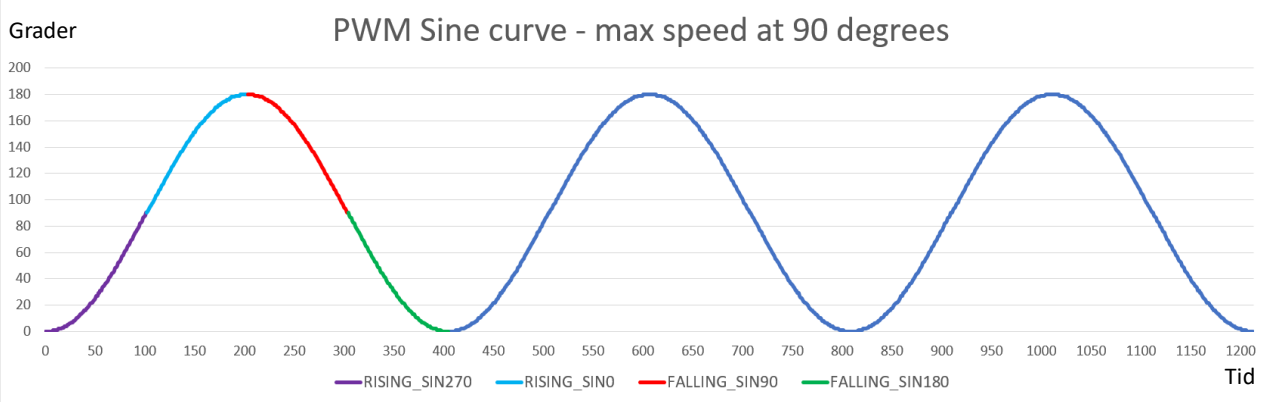


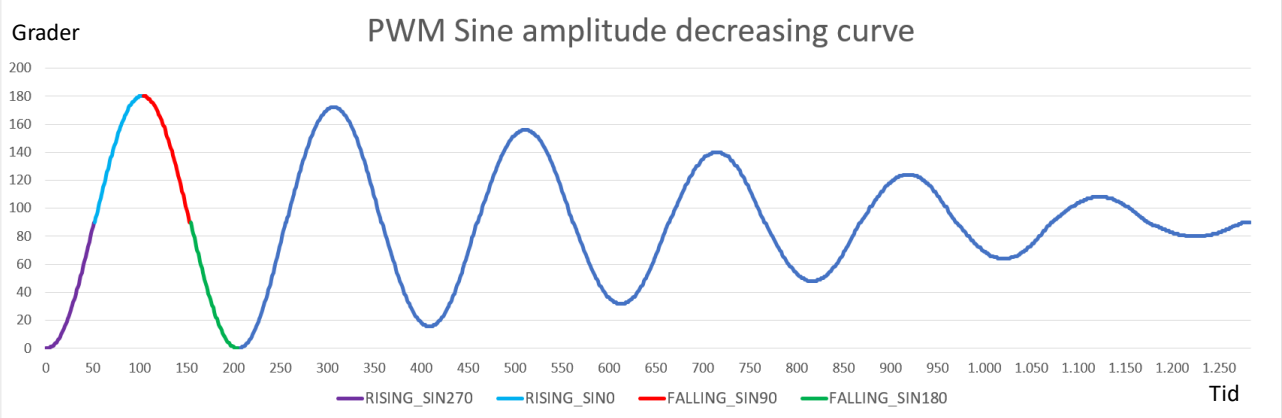


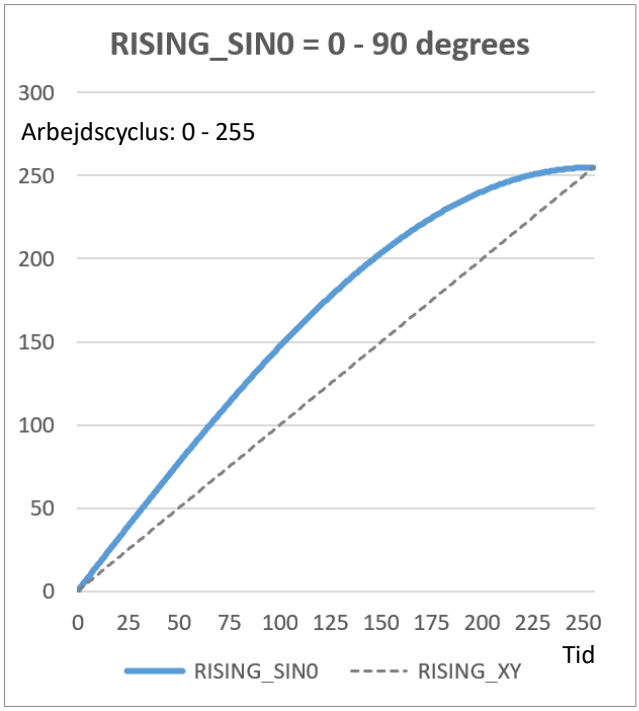
Arduino eksempel: [MTD2A/examples/advanced_servo_curves at main · MTD2A/MTD2A]
DEMO video: [https://youtu.be/parVpl98mDk]

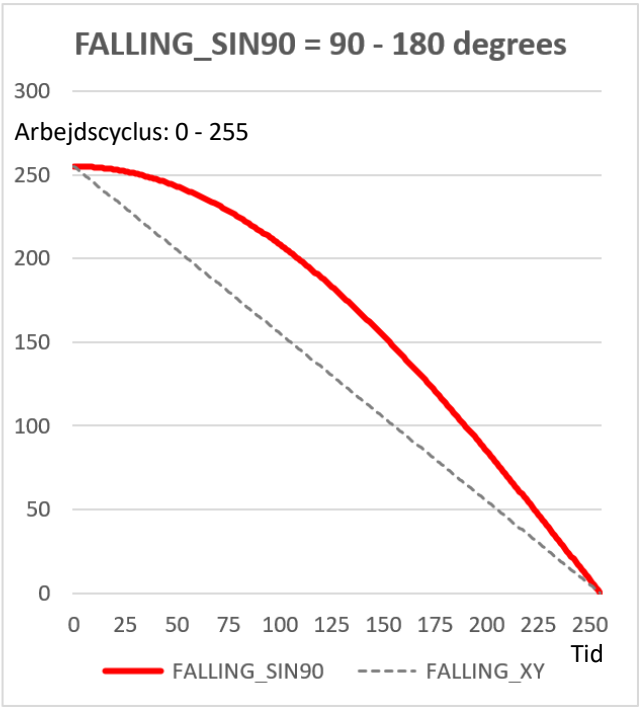
Sinus kurve

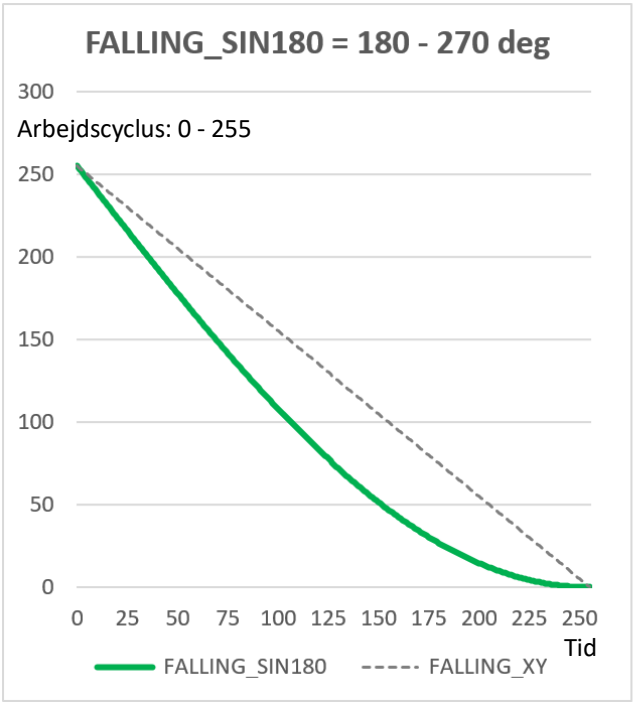
[Sinus kurven] er i opdelt I fire dele på hver på 90 grader, og fordelt samlet i området 0 – 360 grader. På den måde at det muligt at sammensætte de enkelt dele til forskellige nye funktioner.

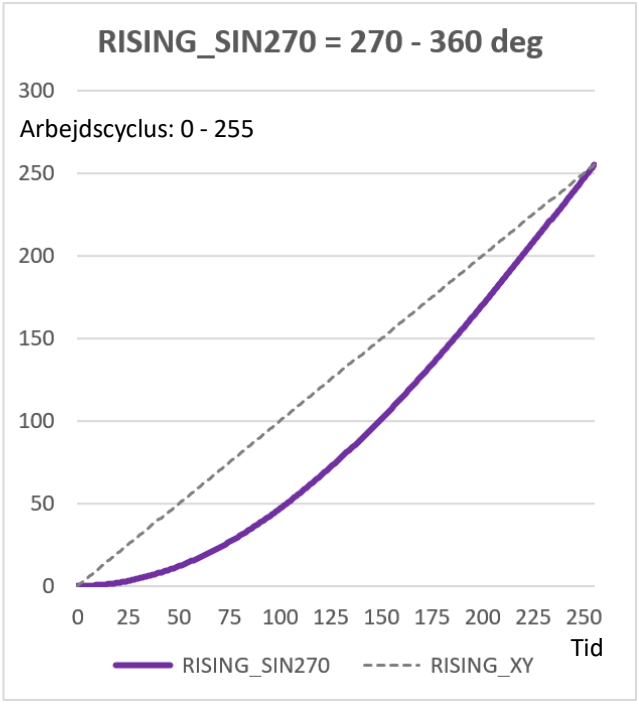


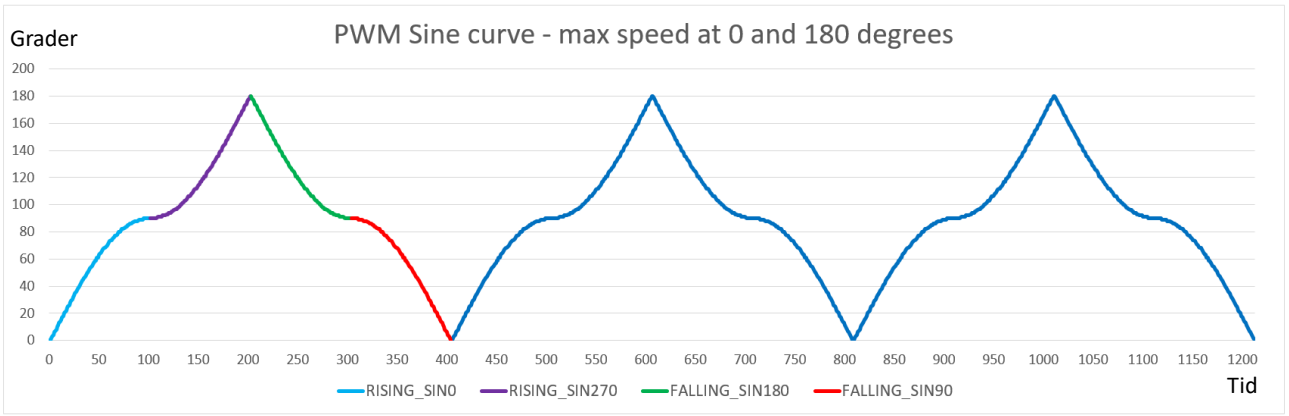












Arduino eksempel: [MTD2A/examples/advanced_servo_curves at main · MTD2A/MTD2A]
DEMO video: [https://youtu.be/parVpl98mDk]

Pendul kørsel med matematiske PWM kurver og H-broer: https://youtu.be/1i-cGc6Dk4E

De forskellige kurver er navngivet som globale konstanter (MTD2A_const.h):

MIN_PWM_VALUE

MAX_PWM_VALUE

NO_CURVE
RISING_XY
RISING_LED
RISING_SM8
RISING_B05
FALLING_B05
RISING_BZ1
RISING_SIN0

FALLING_XY
FALLING_LED
RISING_SM5
RISING_B025
FALLING_B025
FALLING_BZ1
FALLING_SIN90

FALLING_SM8
RISING_E05
FALLING_E05
RISING_BZ2
RISING_SIN270

FALLING_SM5
RISING_E025
FALLING_E025
FALLING_BZ2
FALLING_SIN180

Skriv gerne hvis der er ønske om andre PWM kurver: MTD2A@jorgen-madsen.dk

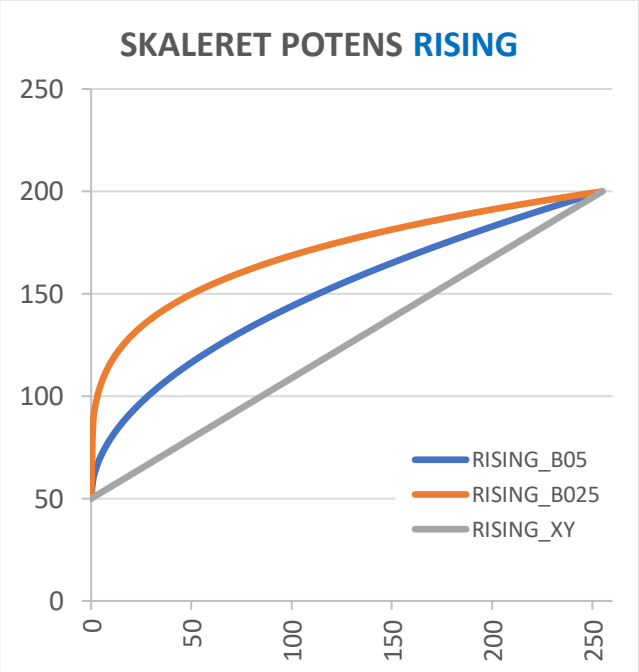
Skalering

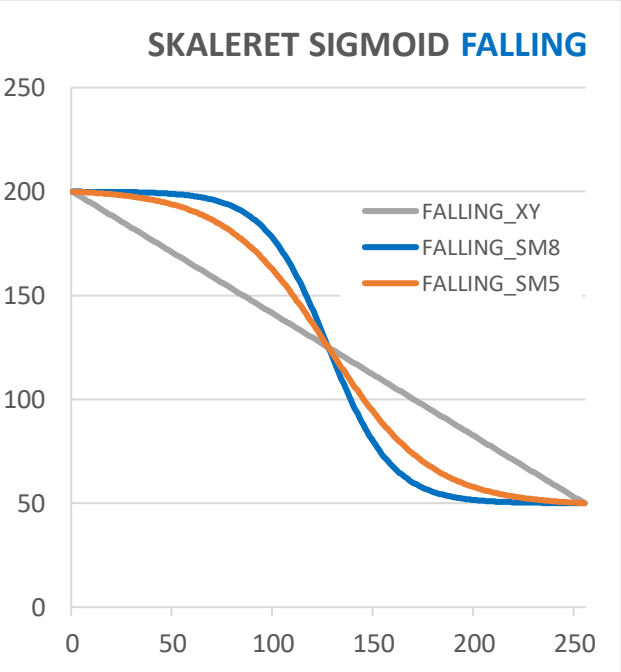
Alle kurver kan skaleres. Hvis værdien beginValue og værdien endValue er defineret skaleres hele kurven inden for tidsperioden outputTimeMS

Eksempler på skalering:

```
motor.activate (50, 200, RISING_XY);
motor.activate (50, 200, RISING_B05);
motor.activate (50, 200, RISING_B025);
```

```
servo.activate (200, 50, FALLING_XY);
servo.activate (200, 50, FALLING_SM8);
servo.activate (200, 50, FALLING_SM5);
```





outputValue og PWM

Der henvises til afsnittet outputValue for en introduktion.

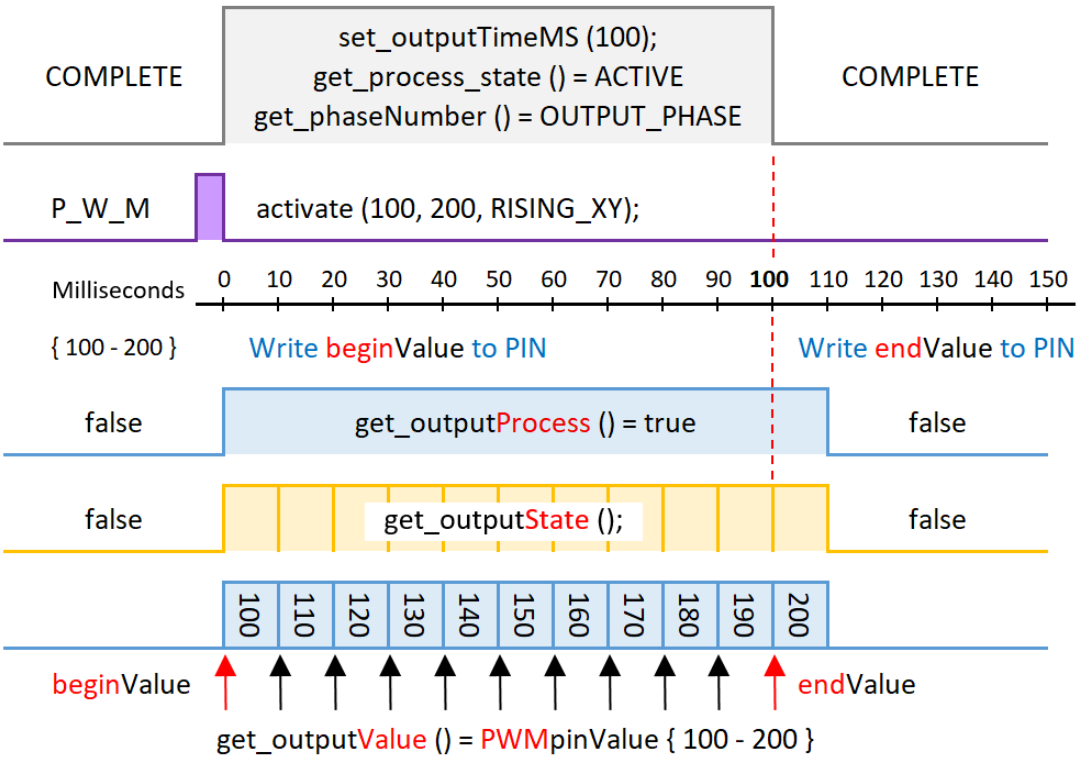
De enkelte PWM kurver beregnes punktvis og ligeligt hen over `outputTimeMS` perioden, og med tidsmæssige trin, der bestemmes af `globalDelayTimeMS` Se uddybende forklaring om tidsstyring og -synkronisering i dokumentet MTD2A_base.pdf.

De enkelte beregnede punkter på kurven kan aflæses med funktionen: `object_name.get_OutputValue ();`

`object_name.get_outputState ();` er `true` når der er **nye data** til aflæsning. Det betyder, at hvis samme værdi opstår flere gange i træk, returnerer funktionen kun `true` den første gang værdien opstår.

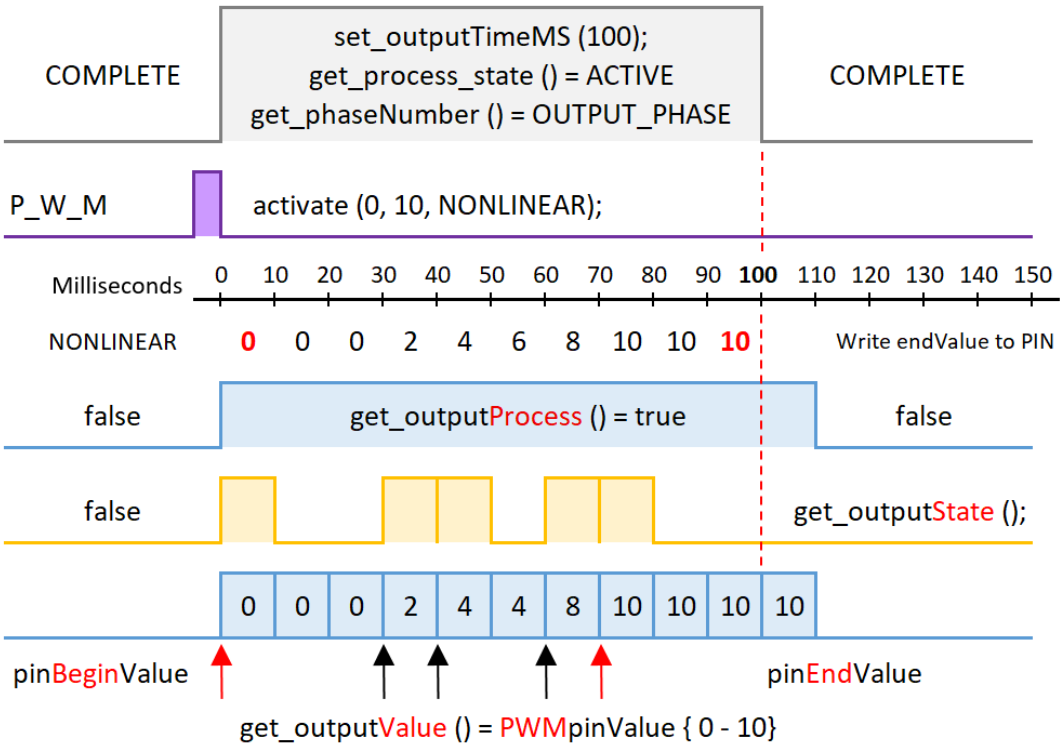
Om der skal optimeres eller ej styres med funktionen: `object_name.set_outputOptimize ( {ENABLE | DISABLE} );`

Den aktuelle konfiguration kan aflæses med funktionen `object_name.get_outputOptimize ( );`



Eksempel på liniær PWM funktion RISING_XY

```
MTD2A_binary_output PWM_curve ("PWM_curve", 3000); // 3 seconds slow movement
Servo myServo;
// Initialize and loop etc.
if (PWM_curve.get_processState() == COMPLETE) {
      PWM_curve.activate (0, 180, RISING_XY); // begin 0 and end 180 degrees
}
if (PWM_curve.get_outputState () == true) {
  myServo.write( PWM_curve.get_outputValue() );
}
```

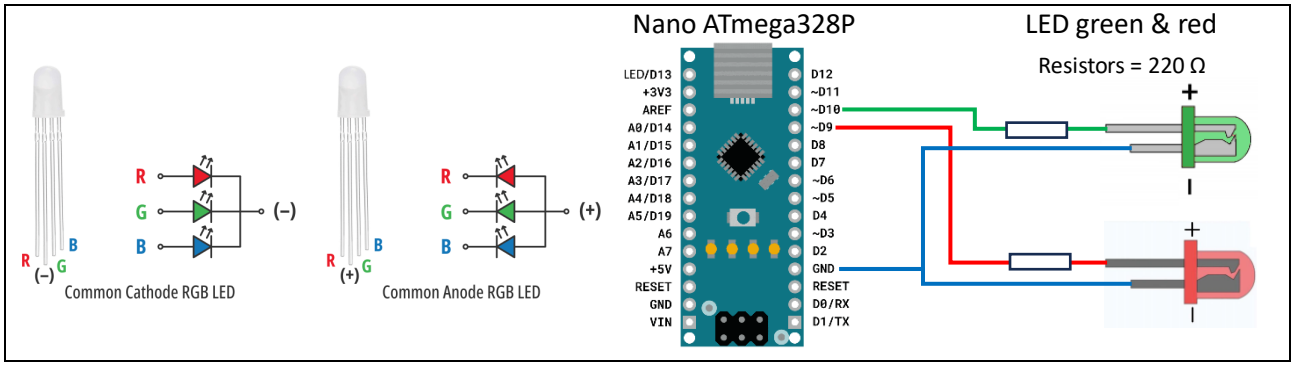


Eksempel på ulineiær PMW funktion RISING_SM5

```
MTD2A_binary_output PWM_curve ("PWM_curve", 3000); // 3 seconds slow movement
Servo myServo;
// Initialize and loop etc.
if (PWM_curve.get_processState() == COMPLETE) {
      PWM_curve.activate (0, 180, RISING_SM5); // begin 0 and end 180 degrees
}
if (PWM_curve.get_outputState () == true) {
  myServo.write( PWM_curve.get_outputValue() );
}
```

Eksempler på configuration

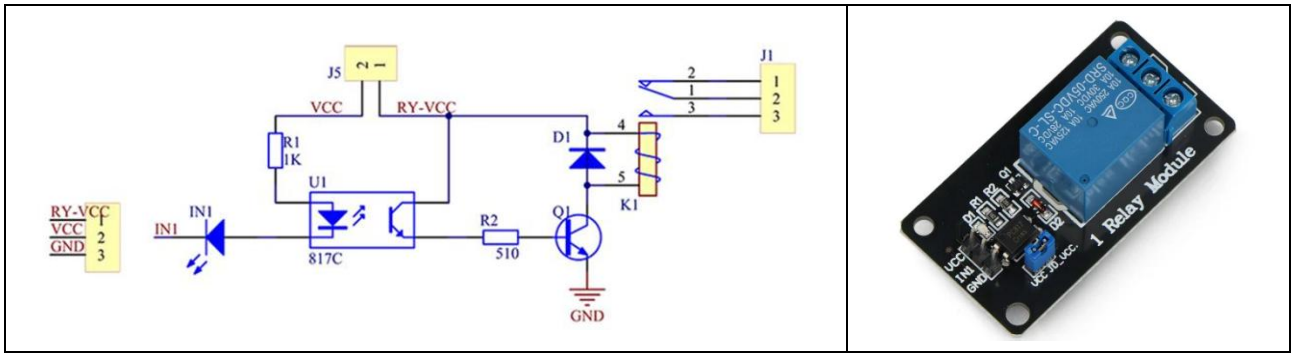
En multifarvet LED med fælles **katode** aktiveres ved at gå fra LOW til HIGH. PWM 0 -> 255 (fuld lys).
En multifarvet LED med fælles **anode** aktiveres ved at gå fra HIGH til LOW. PWM 255 -> 0 (fuld lys).



Eksempel på grøn LED der venter et halvt sekund og lyser herefter i et halvt sekund.

```
1. MTD2A_binary_output green_LED ("Green LED", 500, 500);
2. green_LED.initialize (10);
3. green_LED.activate ();
4. MTD2A_loop_execute ();
```

Et standard rellæ med optokobler der aktiveres ved at gå fra HIGH til LOW.



Eksempel på et optokoblet relæ, der venter et halvt sekund, og aktiveres herefter i et halvt sekund.
Alle standard værdier er omvendt af hvad der er default. Derfor skal alle parametre angives.

```
1. MTD2A_binary_output opto_relay ("Opto relay", 500, 500, 0, BINARY, LOW, HIGH);
2. opto_relay.initialize (12, NORMAL, HIGH);
3. opto_relay.activate ();
4. MTD2A_loop_execute ();
```

Alternativt INVERTED hvilket inverterer (omvender) alle PIN output værdier.

```
1. MTD2A_binary_output opto_relay ("Opto relay", 500, 500);
2. opto_relay.initialize (12, INVERTED);
3. opto_relay.activate ();
4. MTD2A_loop_execute ();
```

Set og get funktioner

Set functions Grøn er standard, ved ingen parameter	Comment
set_outputMode ({ BINARY | P_W_M });	Select binary or PWM output
set_outputOptimize ({ENABLE | DISABLE});	ENABLE : outputState is only when outputValue change from last outputValue DISABLE: outputState always true
set_PWMcurveType (PWM curve number/name)	Select PWM curve to use in PWM mode
set_pinWriteValue ({BINARY {HIGH | LOW} / PWM {0-255}) , {BINARY | P_W_M});	Write directly to output PIN (if enabled) **Optional** parameter. No default.
set_pinWriteToggl ({ENABLE | DISABLE});	Enable or disable pin writing
set_pinWriteMode ({NORMAL | INVERTED});	Invert all output (pin writing)
set_outputTimeMS ({0- MAX_TIME_MS })	Set new **output** time period milliseconds
set_beginDelayMS ({0- MAX_TIME_MS })	Set new **begin** delay time period millisec.
set_endDelayMS ({0- MAX_TIME_MS })	Set new **end** delay time period millisec.
set_timers (outputTimeMS, beginDelayMS, endDelayMS)	Set one, two or three timers milliseconds **Optional** parameters. No defaults.
set_outputTimer ({ STOP_TIMER | RESET_TIMER})	Stop timer process immediately or reset
set_beginTimer ({ STOP_TIMER | RESET_TIMER})	Stop timer process immediately or reset
set_endTimer ({ STOP_TIMER | RESET_TIMER})	Stop timer process immediately or reset
set_loopActivate ({ENABLE | DISABLE}, {0-65.535});	Repeat the activation process until DISABLE or activateCount has reached activateNumber **Optional** parameter. Default = 0
set_debugPrint ({ENABLE | DISABLE});	Enable print phase number and text
set_errorPrint ({ENABLE | DISABLE});	Enable error messages

Get functions	Comment
get_outputMode (); return bool { BINARY | P_W_M }	Currently selected binary or PWM output
get_outputOptimize (); return bool {ENABLE | DISABLE}	Currently selected active or not
get_PWMcurveType (); Return uint8_t curve number	Current selected PWM curve
get_processState (); return bool {ACTIVE | COMPLETE}	Process state.
get_pinWriteMode (); return bool {NORMAL | INVERTED}	Output is normal or inverted
get_pinWriteToggl (); return bool {ENABLE | DISABLE}	Write to pin is enabled or disabled
get_outputValue () return uint8_t {HIGH | LOW} / {0- 255}	Current output value
get_outputState ();return bool {true | false}	Indicates when to call get_outputValue ();
get_outputProcess (); return bool {ACTIVE | COMPLETE}	Output process state
get_phaseChange (); return bool {true | false}	Momentarily phase change (one loop time)
get_phaseNumber (); return uint8_t {0- 4}	Reset = 0, begin = 1, output = 2, end = 3, complete = 4.
get_setBeginMS (); return uint32_t milliseconds	Start time for **begin delay** proces
get_setOutputMS (); return uint32_t milliseconds	Start time for **output** process
get_setEndMS (); return uint32_t milliseconds	Start time for **end delay** proces
get_loopActivate (); return bool {ENABLE | DISABLE}	loopActivate status
get_activateNumber (); return uint16_t number	Number of activations to perform
get_activateCounter ();return uint16_t number	Current number of activations.
get_reset_error (); return uint8_t {0-255}	Get error/warning number and reset number: Error [1 – 127] warning [128 – 255]

Operator overloading	Function
object_name & !objectName e.g. if (objectName) { activate };	bool phaseChange == true && phaseNumber == COMPLETE_PHASE
object_name_1 == object_name_2	bool processState_1 == processState_2
object_name_1 != object_name_2	bool processState_1 != processState_2
object_name_1 > object_name_2	bool processState_1 = ACTIVE & processState_2 = COMPLETE
object_name_1 < object_name_2	bool processState_1 = COMPLETE & processState_2 = ACTIVE

print_conf();

object_name.print_conf ();

```
MTD2A_binary_output:
--------------------
  objectName    : LED_1
  processState  : ACTIVE
  phaseText     : [3] End delay
  debugPrint    : DISABLE
  globalDebugPr: DISABLE
  errorPrint    : DISABLE
  globalErrorPr: ENABLE
  errorNumber   : 0 OK
  outputTimeMS  : 2000
  beginDelayMS  : 0
  endDelayMS    : 2000
  outputMode    : PWM
  PWMcurveType  : 0 (NO_CURVE)
  PWMloopCount  : 200
  beginValue    : 10
  endValue      : 0
  outputValue   : 255
  loopActivate  : DISABLE
  activateNumbr: 0(infinite)
  activateCount: 0
  pinNumber     : 9
  pinWriteToggl: ENABLE
  pinWriteMode  : NORMAL
  pinStartValue: 0
  PinWriteValue: 0
  setOutputMS   : 2014
  setBeginMS    : 0
  setEndMS      : 4015
```