

# MTD2A\_binary\_output

**MTD2A:** Model Train Detection And Action – arduino library <https://github.com/MTD2A/MTD2A>  
Jørgen Bo Madsen / V1.6 / 25-10-2025

MTD2A\_binary\_output er en brugervenlig avanceret og funktionel C++ klasse til tidsstyret håndtering af output til relæer, LED'er og meget mere. MTD2A understøtter parallel processering og asynkron eksekvering.

Klassen er blandt en række logiske byggeklodser, der løser forskellig funktioner. Fælles for dem alle:

- Understøtter en bred vifte af inputsensorer og outputenheder
- Er enkle at bruge til at bygge komplekse løsninger med få kommandoer
- Fungere ikke-blokkerende, procesorienteret og tilstandsdrevet
- Tilbyder omfattende kontrol- og fejlfindingsinformation
- Grundigt dokumenterede med eksempler

## Indholdsfortegnelse

MTD2A_binary_output.....	1
Funktionsbeskrivelse .....	1
Proces faser .....	3
Initialisering .....	3
Aktivering.....	4
Stop og reset (restart) timer .....	4
PWM .....	5
Eksempler på configuration.....	9
Set og get funktioner .....	10
print_conf();.....	11

## Funktionsbeskrivelse

MTD2A\_binary\_output processen består af 4 funktioner:

1. `MTD2A_binary_output object_name  
("object_name", outputTimeMS, beginTimeMS, endTimeMS, { BINARY | PWM }, pinBeginValue, pinEndValue);`
2. `object_name.initialize ( pinNumber, startPinValue );`  
Kaldes i `void setup ();` og efter `Serial.begin (9600);`
3. `object_name.activate ();` Aktiverer en gang og virker først igen 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 max antal parametre.  
Dog skal parametre angives i stigende rækkefølge startende fra den første. Se eksempl herunder:

```
MTD2A_binary_input object_name;
MTD2A_binary_input object_name
  ("object_name");
  ("object_name", outputTimeMS);
  ("object_name", outputTimeMS, beginTimeMS);
  ("object_name", outputTimeMS, beginTimeMS, endTimeMS);
  ("object_name", outputTimeMS, beginTimeMS, endTimeMS, pinOutputMode);
  ("object_name", outputTimeMS, beginTimeMS, endTimeMS, pinOutputMode, pinBeginValue);
  ("object_name", outputTimeMS, beginTimeMS, endTimeMS, pinOutputMode, pinBeginValue, pinEndValue);
```

Defaults:

```
("Object name", 0 , 0 , 0 , BINARY , HIGH , LOW);
```

## Eksempel

```
// Two blinking LEDs. One with symmetric interval and another with asymmetric 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, PWM, 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() == PENDING) {
    red_LED.activate();
  }
  if (green_LED.get_processState() == PENDING) {
    green_LED.activate();
  }

  MTD2A_loop_execute();
}
```

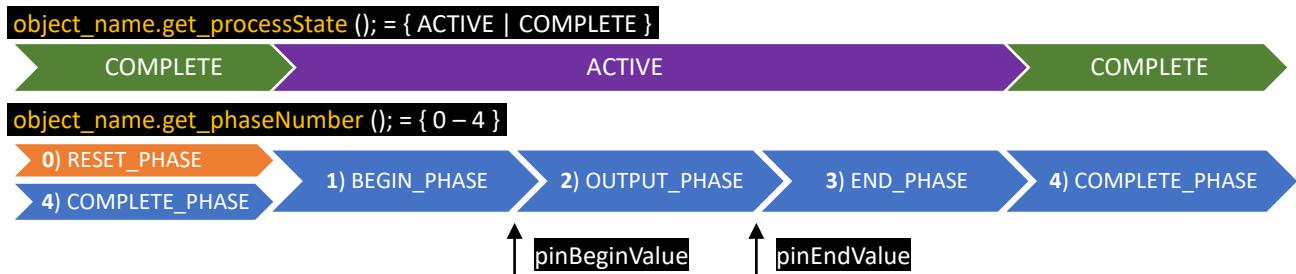
Flera eksempler og youtube video:

<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 initiale 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 `pinBeginValue` og slutter med `pinEndValue`.  
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`

Det **øjeblikkelige** fasesskift kan identificeres med funktion: `object_name.get_phaseChange(); = { true | false }`

## Proces status

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`.

## Timing

Se dokumentet MTD2A.pdf og afsnittet "Kadance" og "Synkronisering" samt "Eksekveringshastighed".

## Initialisering

`object_name.initialize( pinNumber, startPinValue );` Kaldes i `void setup();` og efter `Serial.begin(9600);`  
Output skrives til det digitale benforbindelsesnummer, der er specificeret i `object_name.initialize( pinNumber );`  
Kaldes funktionen ikke, eller engives der en benforbinelser der ikke understøttes, bliver der ikke skrevet til  
benforbindelsen: `pinWriteToggl = DISABLE` og `pinNumber = 255`.

Output på benforbindelsen kan inverteres (omvendes) ved at sætte `pinOutput` til `INVERTED`  
Det betyde at der byttes om på HIGH og LOW og PWM værdi regnes til 255 – 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, {NORMAL | INVERTED}, startPinValue );`

Hvis benforbindelsenummer er initialiseret korrekt med `object_name.initialize( pinNumber )`, er det muligt  
løbende at styre om der skal skrives til benforbinelsen eller ej med funktionen:

`object_name.set_pinWriteToggl( {ENABLE | DISABLE} );`

Det er også muligt at skrive direkte til benforbindelsen med funktionen:

`object_name.pinWriteValue( PinValue );` binary {HIGH | LOW} / PWM {0-255}

`object_name.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()`;

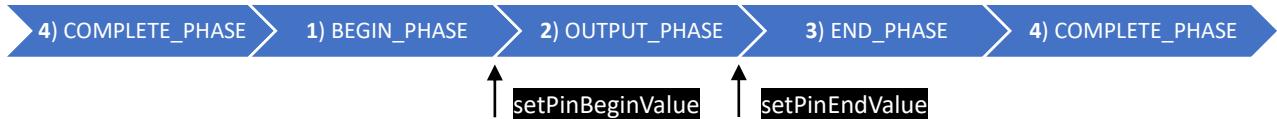
Som udgangspunkt er benforbindelsen **udefineret**. Se [Digital Pins | Arduino Documentation](#)

## Aktivering

Processen aktiveres med funktionen: `object_name.activate ()`; Dermed skifter `procesState` til **ACTIVE**. Efterfølgende aktivering har ingen effekt, så længe processen er aktiv. Så snart `procesState` skifter til **COMPLETE** kan processen aktiveres igen.

Processen kan til hver en tid nulstilles med funktionen: `object_name.reset ()`; Funktionen nulstiller 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 `pinBeginValue` og slut værdier `pinEndValue` 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 værdier. Alle eksisterende værdier forbliver de samme, med mindre de angives som parametre i funktionskaldet. Se eksempel herunder:

```
object_name.activate ();
object_name.activate (setPinBeginValue);
object_name.activate (setPinBeginValue, setPinEndValue);
object_name.activate (setPinBeginValue, setPinEndValue, setPWMcurveType);
object_name.activate (setPinBeginValue, setPinEndValue, setPWMcurveType, setOutputTimeMS);
```

`setPWMcurveType` angiver hvilken kurve der skal følges i tidsrummet `outputTimeMS`. Kurven starter med værdien `setPinBeginValue` og slutter med værdien `setPinEndValue`.

`setOutputTimeMS` angiver tidsrummet mellem `setPinBeginValue` og `setPinEndValue` i millisekunder.

## Stop og reset (restart) timer

Det er muligt at sætte det nye 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::globalSyncTimeMS ();
```

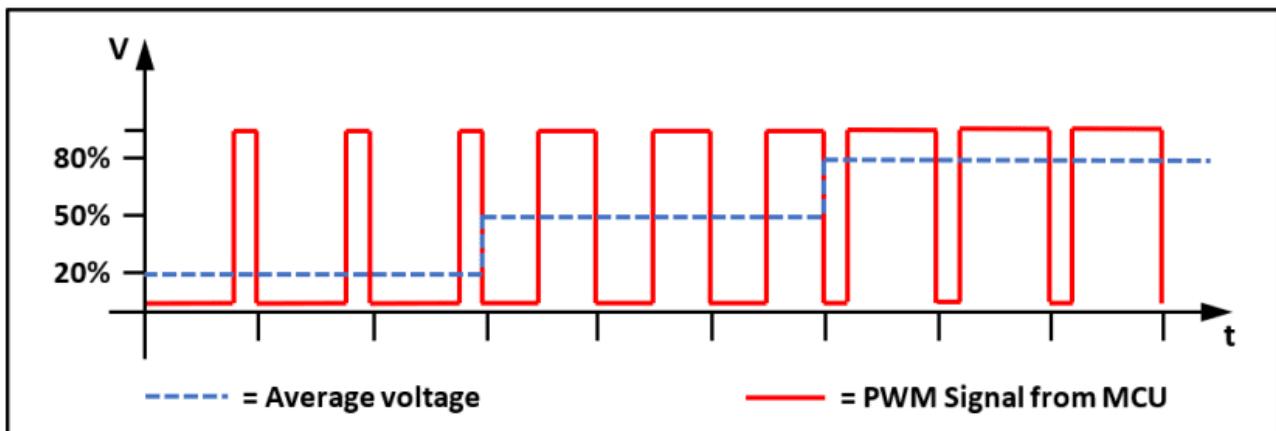
Tidsperioden sættes som standard når objektet instantieres (aktiveres).

Det er muligt at definere nye tidsperider for alle tre tidsperioder:

```
object_name.set_outputTimeMS ( {0 - 4294967295} );
object_name.set_beginDelayMS ( {0 - 4294967295} );
object_name.set_endDelayMS ( {0 - 4294967295} );
```

## PWM

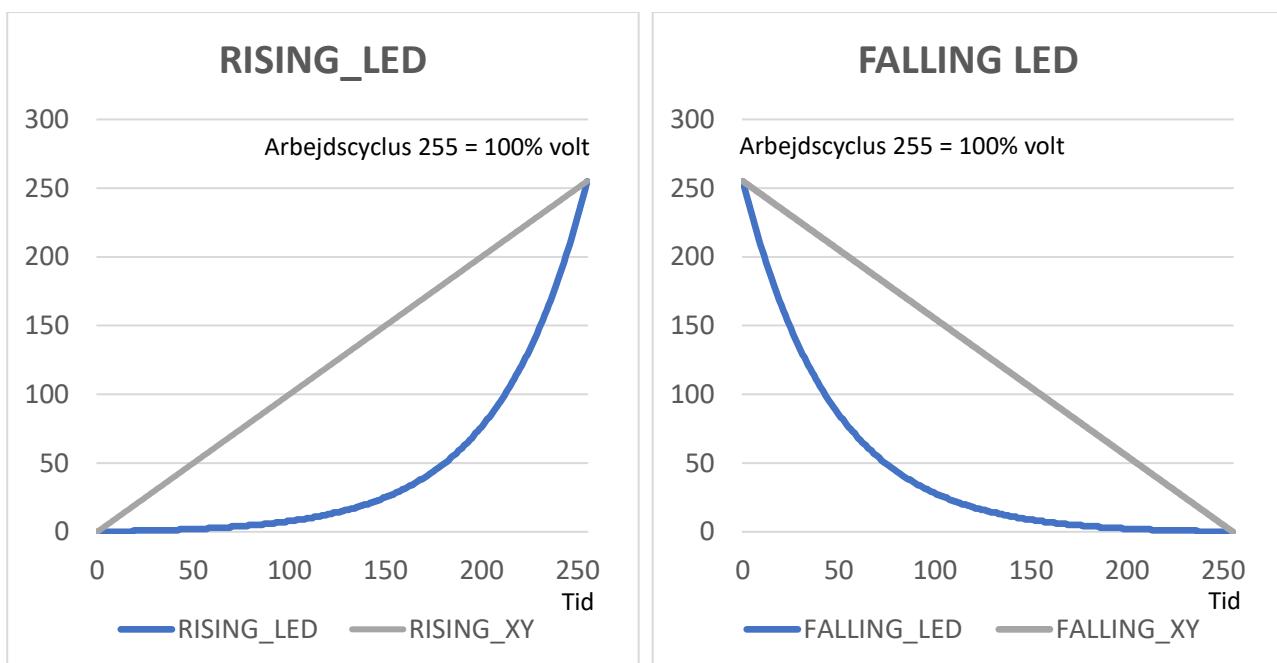
PWM (Pulse Width Modulation) er en teknik, der bruges til at kontrollere den gennemsnitlige effekt, der leveres til en elektrisk enhed ved at variere bredden af en række impulser. I bund og grund er det en digital metode til at skabe en analog effekt ved hurtigt at tænde og slukke for et signal



16 forskellige matematiske PWM-kurver er designet: 2 lineære, 6 eksponentielle og 8 potensfunktioner. Funktionerne er beregnet til at udjævne fysiske forhold, der har et ikke-lineært forhold mellem tid og funktion. F.eks. LED fade ind/fade ud og acceleration og deacceleration 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.

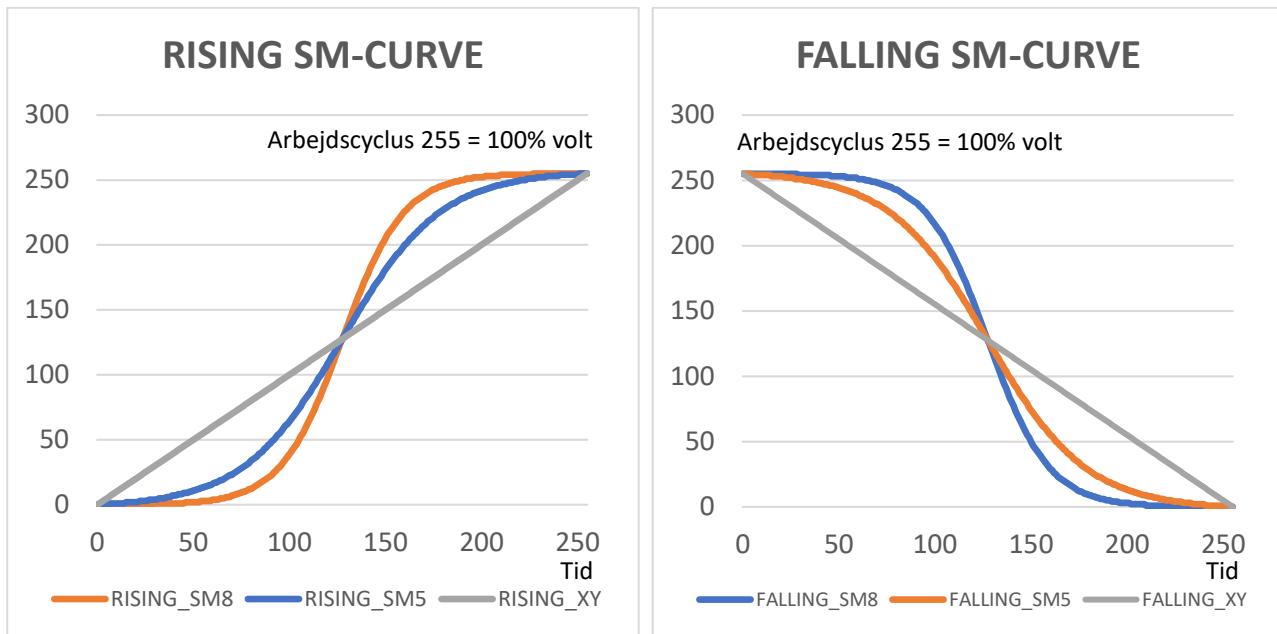


Arduino eksempel: [MTD2A/examples/math\\_fade\\_LED at main · MTD2A/MTD2A](#)

DEMO video: <https://youtu.be/8TV6nOdXBno>

## S-kurver

Sigmoid S-kurver Kan bruges, hvor blød acceleration og deceleration er at foretrække. For eksempel, at køre en (tungt belastet) servomotor fra punkt a til b, eller en servomotor, der sænker/rejser bomme op og ned ved en jernbaneoverskæring.



Arduino eksempel: [MTD2A/examples/servo\\_math\\_curve\\_at\\_main · MTD2A/MTD2A](#)

DEMO video: <https://youtu.be/rhQtu0iKFI8>

## Potensfunktioner

Potensfunktioner er til udjævning af acceleration og deceleration af tog.

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

## Inerti and friktion

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

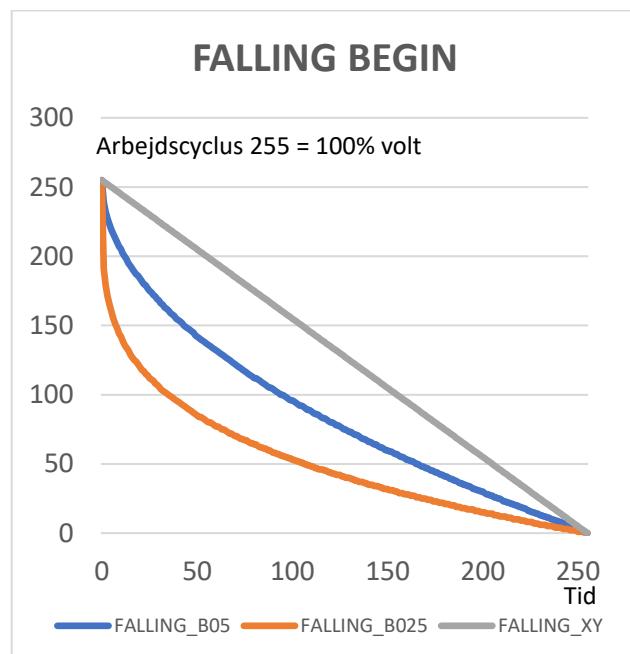
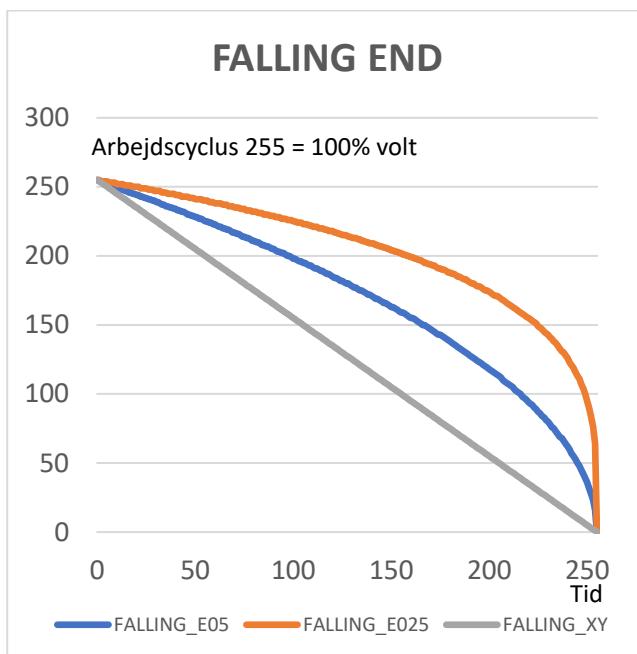
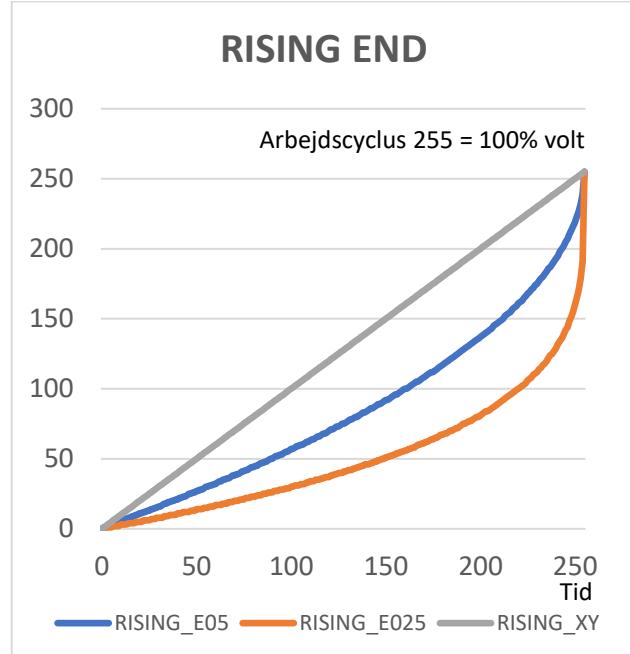
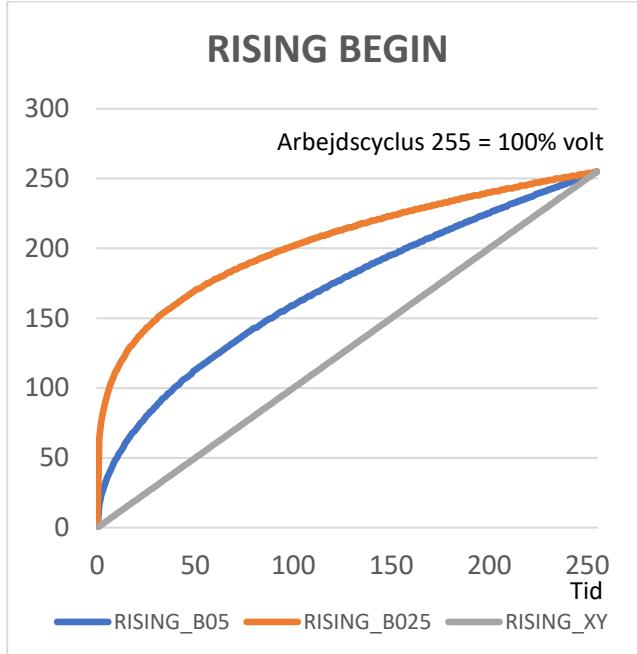
Inerti er egenskaben ved et objekt, der får det til at modstå ændringer i dets bevægelsestilstand. For at starte eller stoppe bevægelse er der brug for en ubalanceret kraft for at overvinde inerti og, i de fleste virkelige scenarier, friktion.

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/pendulum\\_H\\_bridge at main · MTD2A/MTD2A](#)  
 DEMO video: Under udarbejdelse.

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

FALLING\_XY  
 FALLING\_LED  
 RISING\_SM5  
 RISING\_B025  
 FALLING\_B025

FALLING\_SM8  
 RISING\_E05  
 FALLING\_E05

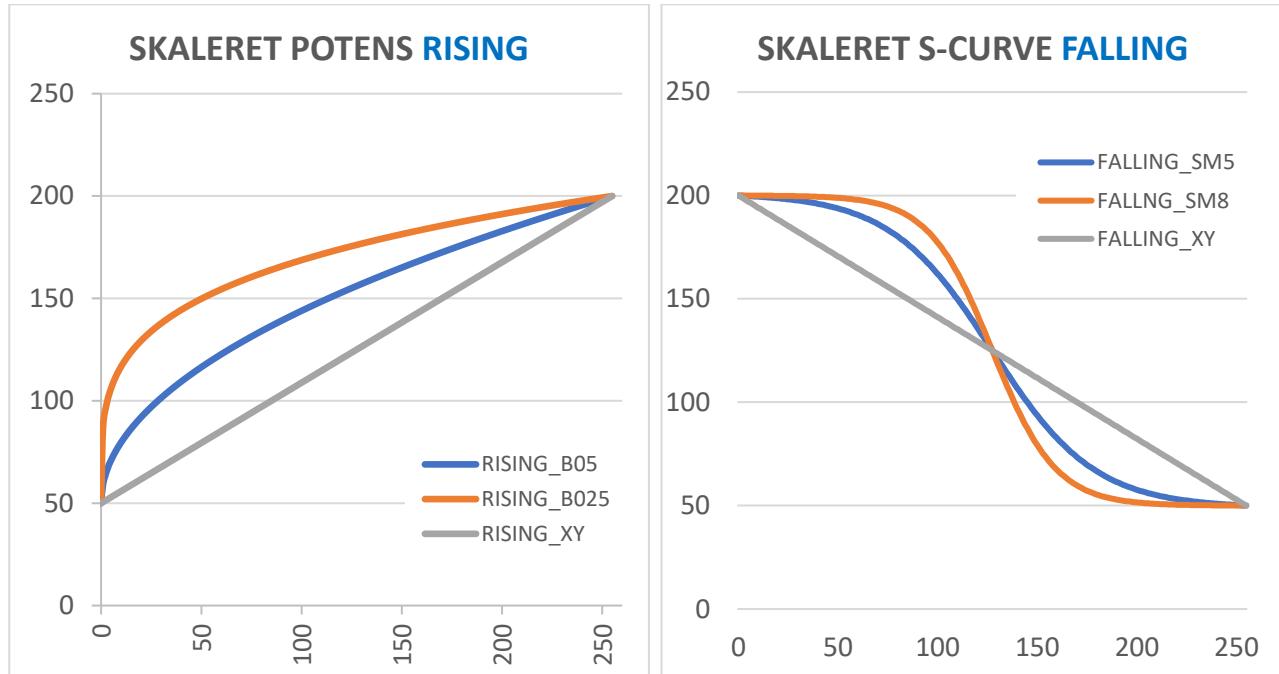
FALLING\_SM5  
 RISING\_E025  
 FALLING\_E025

Alle kurver kan skaleres. Hvis værdien `setPinBeginValue` og værdien `setPinEndValue` er defineres 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, RISING_XY);  
servo.activate (200, 50, RISING_SM8);  
servo.activate (200, 50, RISING_SM5);
```

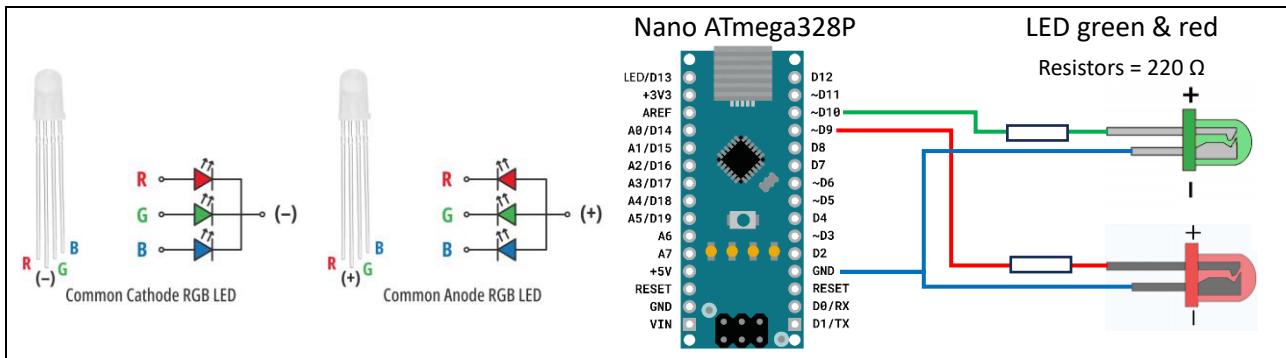


De enkelte beregnede punkter på kurven kan aflæses med funktionen: `object_name.getOutputValue ()`;

## Eksempler på configuration

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

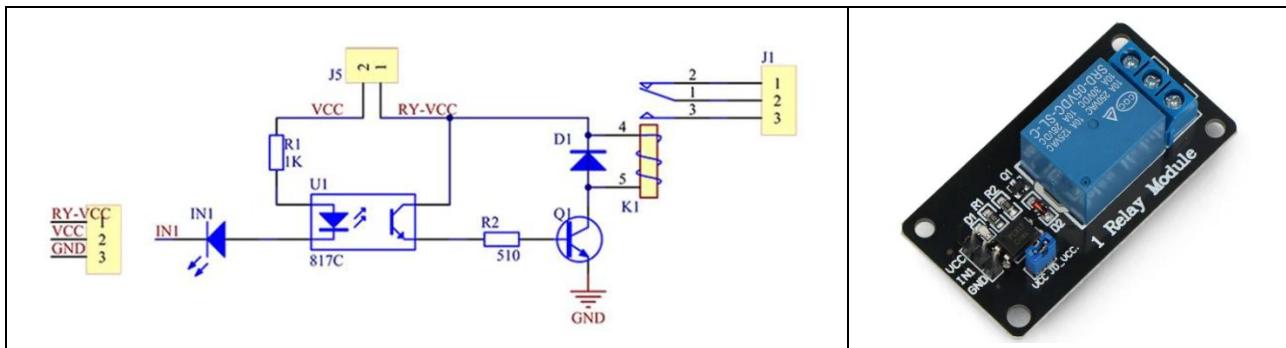
En multifarvet LED med fælles **anode** aktiveres ved at gå fra HIGH til LOW. PWM 255 -> 0 (fuld lys).



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 med ved at gå fra HIGH til LOW.



Eksempel på et optokoblet rellæ, 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

<b>Set functions</b> <i>Green is default when no parameter</i>	<b>Comment</b>
set_PinOutputMode ( { <b>BINARY</b>   P_W_M } );	Select binary or PWM output
set_pinWriteToggl ( { <b>ENABLE</b>   DISABLE} );	Enable or disable pin writing
set_pinWriteMode ( { <b>NORMAL</b>   INVERTED} );	Invert all output (pin writing)
set_pinWriteValue ( {BINARY {HIGH   LOW} / PWM {0-255} } , { <b>BINARY</b>   P_W_M} );	Write directly to output pin (if enabled) <b>Optional</b> parameter. No default
set_outputTimeMS ( {0- 4294967295} )	Set new <b>output</b> time period milliseconds
set_beginDelayMS ( {0- 4294967295} )	Set new <b>begin</b> delay time period millisec.
set_endDelayMS ( {0- 4294967295} )	Set new <b>end</b> delay time period millisec.
set_outputTimer( { <b>STOP_TIMER</b>   RESET_TIMER} )	Stop timer process immediately or reset
set_beginTimer ( { <b>STOP_TIMER</b>   RESET_TIMER} )	Stop timer process immediately or reset
set_endTimer ( { <b>STOP_TIMER</b>   RESET_TIMER} )	Stop timer process immediately or reset
set_debugPrint ( { <b>ENABLE</b>   DISABLE} );	Enable print phase number and text
set_errorPrint ( { <b>ENABLE</b>   DISABLE} );	Enable error messages

<b>Get functions</b>	<b>Comment</b>
get_processState (); return <b>bool</b> {ACTIVE   COMPLETE}	Procedure process state.
get_pinWriteMode (); return <b>bool</b> {NORMAL   INVERTED}	Output is normal or inverted
get_pinWriteToggl (); return <b>bool</b> {ENABLE   DISABLE}	Write to pin is enabled or disabled
get_pinOutputValue() return <b>uint8_t</b> {HIGH   LOW} / {0- 255}	Current output value
get_phaseChange (); return <b>bool</b> {true   false}	Momentarily phase change (one loop time)
get_phaseNumber (); return <b>uint8_t</b> {0- 4}	Reset = 0, begin = 1, output = 2, end = 3, complete = 4.
get_setBeginMS (); return <b>uint32_t</b> milliseconds	Start time for <b>begin</b> proces
get_setOutputMS (); return <b>uint32_t</b> milliseconds	Start time for <b>output</b> proces
get_setEndMS (); return <b>uint32_t</b> milliseconds	Start time for <b>end</b> proces
get_reset_error (); return <b>uint8_t</b> {0-255}	Get error/warning number and reset number: Error [1 – 127] warning [128 – 255]

<b>Operator overloading</b>	<b>Function</b>
object_name_1 == object_name_2	<b>bool</b> processState_1 == processState_2
object_name_1 != object_name_2	<b>bool</b> processState_1 != processState_2
object_name_1 > object_name_2	<b>bool</b> processState_1 = ACTIVE & processState_2 = COMPLETE
object_name_1 < object_name_2	<b>bool</b> processState_1 = COMPLETE & processState_2 = ACTIVE
object_name_1 >> object_name_2	<b>bool</b> setOutputMS_1 > setOutputMS_2
object_name_1 << object_name_2	<b>bool</b> setOutputMS_1 < setOutputMS_2

```
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  
pinOutputMode: PWM  
PWMcurveType   : 0  
pinBeginValue: 10  
pinEndValue    : 0  
pinNumber      : 9  
pinWriteToggl: ENABLE  
pinOutput      : NORMAL  
startPinValue: 0  
PinWriteValue: 0  
setOutputMS   : 2014  
setBeginMS    : 0  
setEndMS       : 4028
```