

# MTD2A\_binary\_output

**MTD2A: Model Train Detection And Action** – arduino library <https://github.com/MTD2A/MTD2A>

Jørgen Bo Madsen / V1.3 / 28-06-2025

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

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-blokerende, procesorienteret og tilstandsrevet
- Tilbyder omfattende kontrol- og fejlfindingsinformation
- Grundigt dokumenterede med eksempler

## Indholdsfortegnelse

MTD2A_binary_output.....	1
Funktionsbeskrivelse .....	1
Stop og restart .....	3
Initialisering .....	4
Aktivering.....	4
PWM kurver.....	5
Eksempler på configuration .....	6
Øvrige funktioner.....	7

## 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 ();` Aktiver 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, pinBeginVlaue);  
("object_name", outputTimeMS, beginTimeMS, endTimeMS, pinOutputMode, pinBeginVlaue, pinEndValue);
```

Defaults:

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

## Eksempel

```
// Two blinking LEDs. One with symmetric interval and another with asymeric 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);  
    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();  
}
```

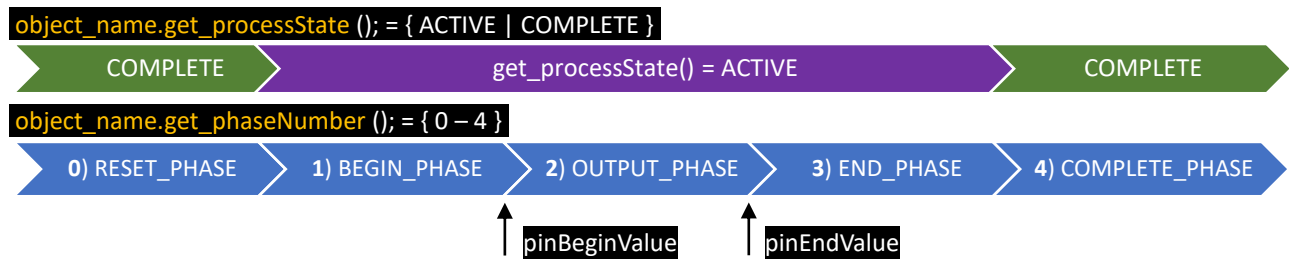
Flere eksempler og youtube video:

<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 initelle fase når programmet starter samt når funktion `reset ();` kaldes.
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.
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 asnittet "Kadance" og "Synkronisering" samt "Eksekverings hastighed".

### Stop og restart

Det er muligt at sætte et nyt start tidspunkt for aktuel timerproces, og stoppe aktuel timerproces i utide.

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

## Initialisering

Output skrives til det digitale benforbindelsesnummer, der er specificeret i `object_name.initialize ( pinNumber );`  
Kaldes funktionen ikke, bliver der ikke skrevet til benforbindelsen `pinWrite = DISABLE` og `pinNumber = 255`.

Output på benforbindelsen kan inverteres (omvendes) ved at sætte `pinOutput` til `INVERTED`

Det betyder at der byttes om på HIGH og LOW og PWM værdi regnes til  $255 - \text{PWM værdi}$ .

`object_name.initialize ( pinNumber, {NORMAL | INVERTED} );`

Benforbindelse initialiseres med den værdi der angivet i `startPinValue`. Udelades parameteren angives `LOW`

`object_name.initialize ( pinNumber, {NORMAL | INVERTED}, startPinValue );`

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

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

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

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

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

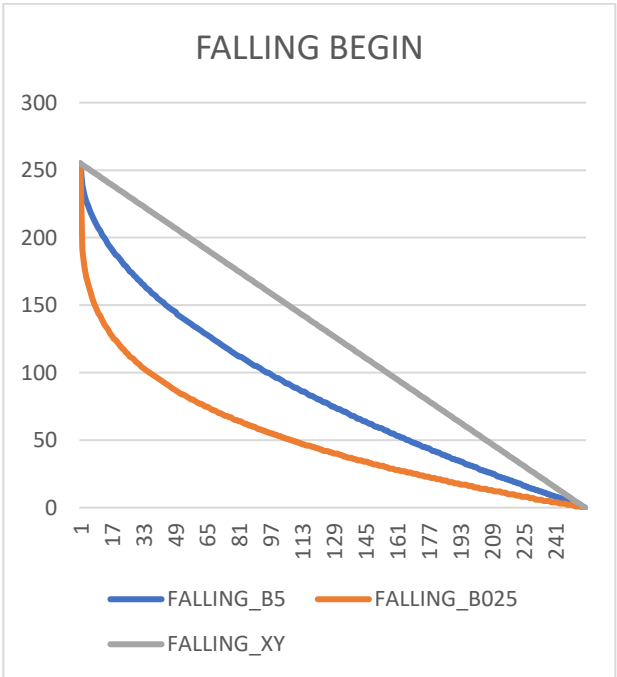
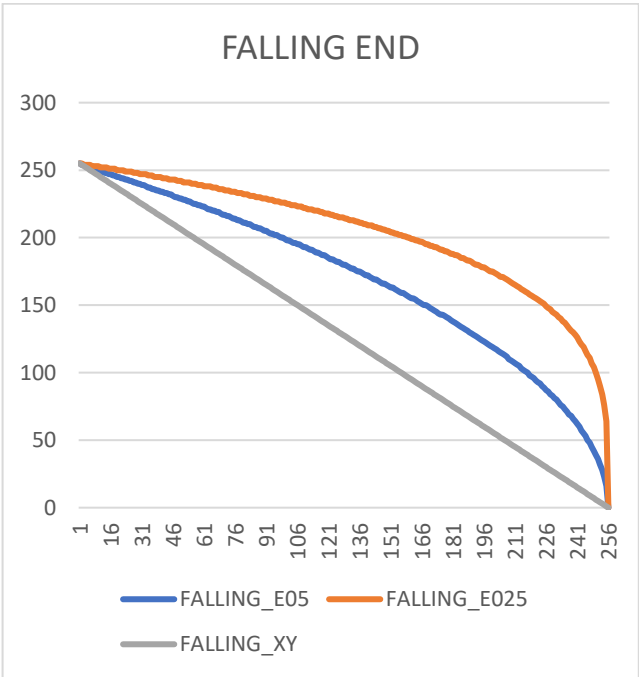
## Aktivering

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, og gør klar til ny start. Alle funktionskonfigurerede variable og standard værdier bibeholdes. Endvidere skrives `startPinValue` til benforbindelsen, hvis benforbindelsen er defineret.  
Procesfasen skifter til `RESET_PHASE`

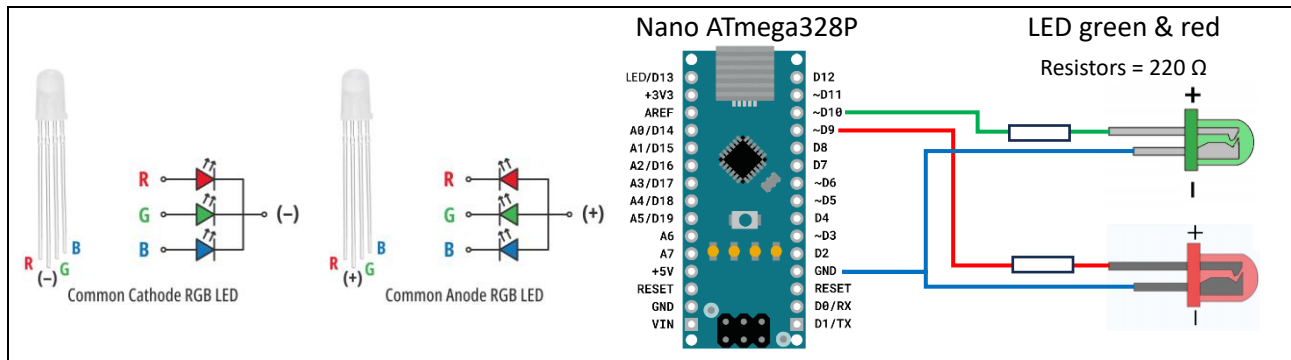
PWM kurver

Kommer i næste release



## Eksempler på configuration

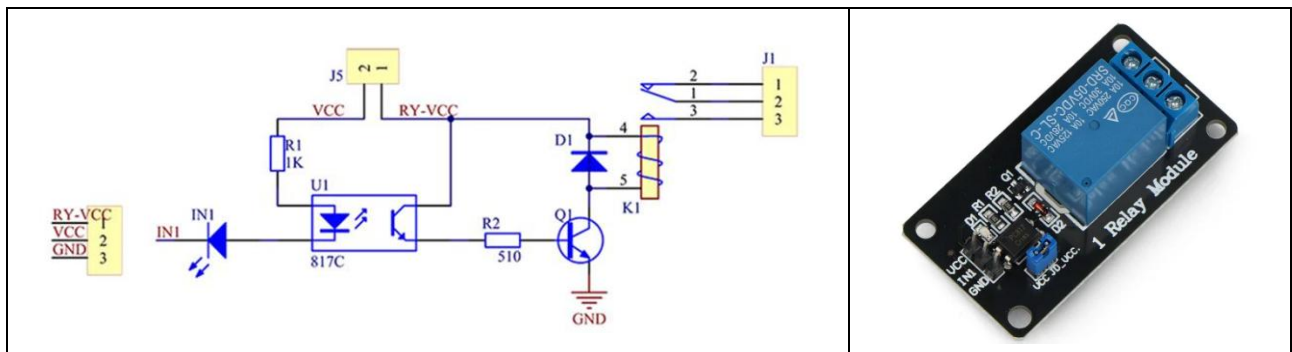
En multifarvet LED med fælles **katode** aktiveres ved at gå fra LOW til HIGH. PWM 0 -> 255 (fuld lys).  
 En multifarvet LED med fælles **anode** aktiveres ved at gå fra HIGH til LOW. PWM 255 -> 0 (fuld lys).



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 relæ med optokobler der aktiveres med 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 ();`

```
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
pinBeginValue   : 10
pinEndValue     : 0
pinNumber       : 9
pinWrite        : ENABLE
pinOutput       : NORMAL
startPinValue   : 0
setPinValue     : 0
setBeginMS      : 0
setOutputMS     : 2014
setEndMS        : 4028
```

## Øvrige funktioner

Set functions	Comment
set_pinWrite ( { <b>ENABLE</b>   DISABLE} );	Enable or disable pin writing
Set_pinOutput ( { <b>NORMAL</b>   INVERTED} );	Invert all output (pin writing)
set_setPinValue ( { <b>BINARY</b> {HIGH   LOW} / PWM {0-255} )	Write directly to output pin (if enabled).
set_outputTimer( { <b>STOP_TIMER</b>   RESTART_TIMER} )	Stop timer process immediately or restart
set_beginTimer ( { <b>STOP_TIMER</b>   RESTART_TIMER} )	Stop timer process immediately or restart
set_endTimer ( { <b>STOP_TIMER</b>   RESTART_TIMER} )	Stop timer process immediately or restart
set_debugPrint ( { <b>ENABLE</b>   DISABLE} );	Activate print phase number and text

Fælles for alle set-funktioner er en ekstra parameter: LoopFastOnce = {ENABLE | **DISABLE**} disable er default.

Get functions	Comment
get_processtState (); return <b>bool</b> {ACTIVE   COMPLETE}	Procedure process state.
get_pinWrite (); return <b>bool</b> {ENABLE   DISABLE}	Write to pin is enabled or disabled
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]

Operator overloading	Function
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
object_name_1 >> object_name_2	bool setOutputMS_1 > setOutputMS_2
object_name_1 << object_name_2	bool setOutputMS_1 < setOutputMS_2