MTD2A_binary_output

MTD2A: Model **T**rain **D**etection **A**nd **A**ction – arduino library https://github.com/MTD2A/MTD2A
Jørgen Bo Madsen / V1.2 / 30-05-2025

MTD2A_binary_output er en avanceret og funktionel C++ klasse til tidsstyret håndtering af output til relæer, LED'er med mere samt programmet selv.

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

- Understøtter en bred vifte af inputsensorer og outputenheder
- Enkel at bruge til at bygge komplekse løsninger
- Ikke-blokerende, tilstandsdrevet, enkel og effektiv tilstandsmaskine
- Omfattende kontrol- og fejlfindingsinformation

Indholdsfortegnelse

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 parameret 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 asymetric interval.

#include <MTD2A.h>

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);
  red_LED.initialize   (9);   // Output pin 9,  common cathode
  green_LED.initialize (10);  // Output pin 10, common cathode
  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();
}
```

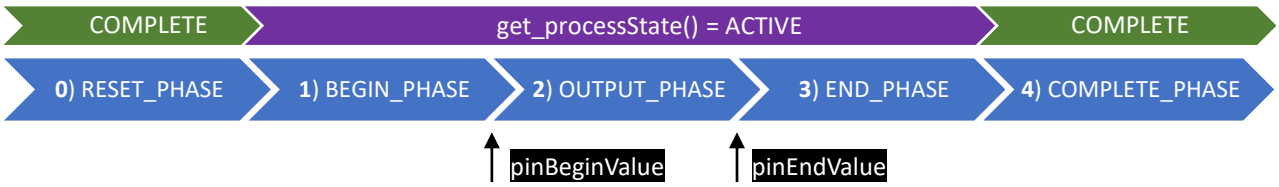
Flere eksempler og youtybe videoer:

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

Youtube videoer: På vej…

Proces faser

Afhægig 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`

Der er to funktioner der kan oplyser hvilken fase processen befinder sig i:

1. `object_name.get_phaseChange ();` = { true | false }
2. `object_name.get_phaseNumber ();` = { 0 – 4 }

Proces status

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

ProcessState kan aflæses med funktionen `object_name.get_processState (); = { ACTIVE | COMPLETE }`

Timing

Tidstagningen er ikke præcis. Dels tager det ekstra tid at genneløbe koden, og dels tilføjes tiden fra et ekstra loop gennemløb på 1- 10 millisekuner. Skal tidstagningen være mere præcis, skal det modregnes i de tider der angives når objektet oprettes. Fx ønsket tid: 2.000 – modregnet tid ca. 1.985 millisekunder.

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.

Benforbindelse initialiseres med den værde der angivet i `startPinValue`. Udelades parameteren angives `LOW`
`object_name.initialize ( pinNumber, **startPinValue** );`

Hvis benforbindelsenummer er initialiseret korrekt med `object_name.initialize`, er det muligt løbende at styre om der skal skrives til benforbinelsen 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]

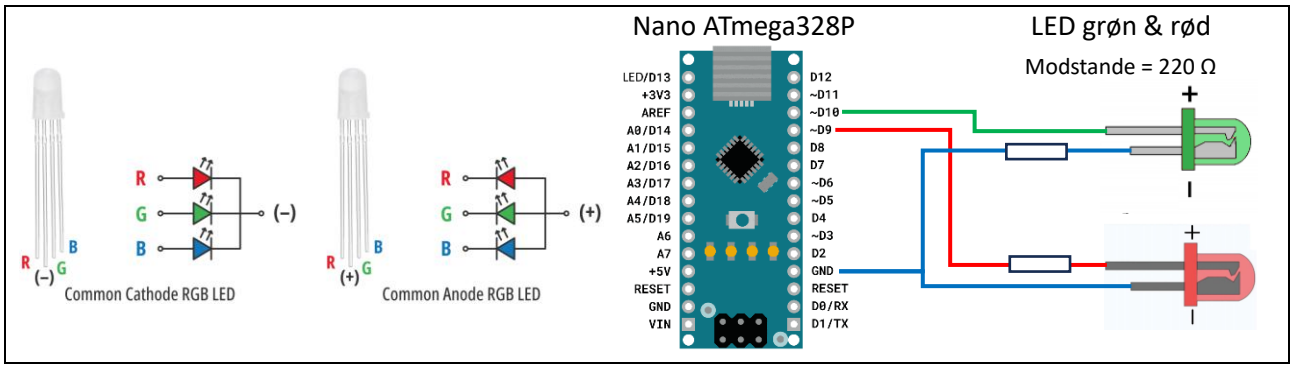
Aktivering

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 nultiller alle styrings- og process variable, og gør klar til ny start. Alle funktionskonfigurerede variable og standard værdier bibeholdes. Endvidere skrives `startPinValue` til benforbindelsen. Procesfasen skifter til `RESET_PHASE`

Eksempler på configuration

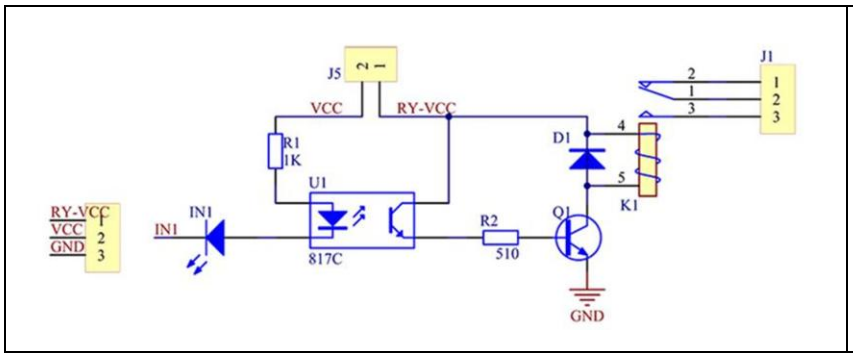
En multifarvet LED med fælles **katode** aktiveres ved at gå fra LOW til HIGH. PWM 0 -> 255 (fuld lys).
En multifarvet LED medfælles **anode** aktiveres ved at gå fra HIGH til LOW. PWM 255 -> 0 (fuld lys).

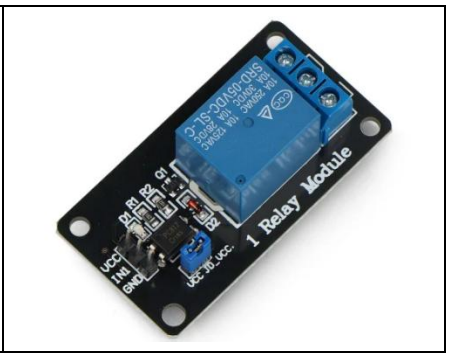


Eksempel på blinkende grøn LED med 0,5 sekunders interval.

```
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 aktiveres med ved at gå fra HIGH til LOW.





Eksempel på et optokoblet rellæ der skifter med 0,5 sekunders interval.
Alle standard værdier er omvendt af hvad der er brug for. Derfor skal alle parametre angives.

```
5.  MTD2A_binary_output opto_relay ("Opto relay", 500, 500, 0, BINARY, LOW, HIGH);
6.  opto_relay.initialize (10, HIGH);
7.  opto_relay.activate ();
8.  MTD2A::loop_execute ();
```

```
object_name.print_conf ();
```

```
MTD2A_binary_output:
--------------------
  objectName    : LED_1
  processState  : ACTIVE
  phaseText     : [3] End delay
  debugPrint    : DISABLE
  errorNumber   : 0 OK
  outputTimeMS  : 2000
  beginDelayMS  : 0
  endDelayMS    : 2000
  pinOutputMode: PWM
  pinBeginValue: 10
  pinEndValue   : 0
  pinNumber     : 9
  pinWrite      : ENABLE
  startPinValue: 0
  setPinValue   : 0
  setBeginMS    : 0
  setOutputMS   : 2014
  setEndMS      : 4028
```

Øvrige funktioner

Set functions	Comment
set_pinWrite ({ENABLE | DISABLE});	Enable or disable pin writing
set_setPinValue ({BINARY {HIGH | LOW} / PWM {0-255})	Write directly to output pin (if enabled).
set_debugPrint ({ENABLE | DISABLE});	Activate print phase number and text

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

Get functions	Comment
get_processtState (); return bool {ACTIVE | COMPLETE}	Procedure process state.
get_pinWrite (); return bool {ENABLE | DISABLE}	Write to pin is enabled or disabled
get_phaseChange (); return bool {true | false}	Momentarily phase change (one loop time)
get_phaseNumber (); return uint8_t {0 - 4}	Initialize & reset = 0, begin = 1, output = 2, end = 3, complete = 4.
get_setBeginMS (); return uint32_t milliseconds	Start time for **begin** proces
get_setOutputMS (); return uint32_t milliseconds	Start time for **output** proces
get_setEndMS (); return uint32_t milliseconds	Start time for **end** proces
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_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