MTD2A & MTD2A_base

MTD2A: Model **T**rain **D**etection **A**nd **A**ction – Arduino library https://github.com/MTD2A/MTD2A
Jørgen Bo Madsen / V1.3.2 / 16-10-2026

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
- De fungerer ikke-blokerende, procesorienterede og tilstandsdrevne
- Tilbyder omfattende information om styring og fejlfinding
- Grundigt dokumenteret med mange eksempler

Indhold

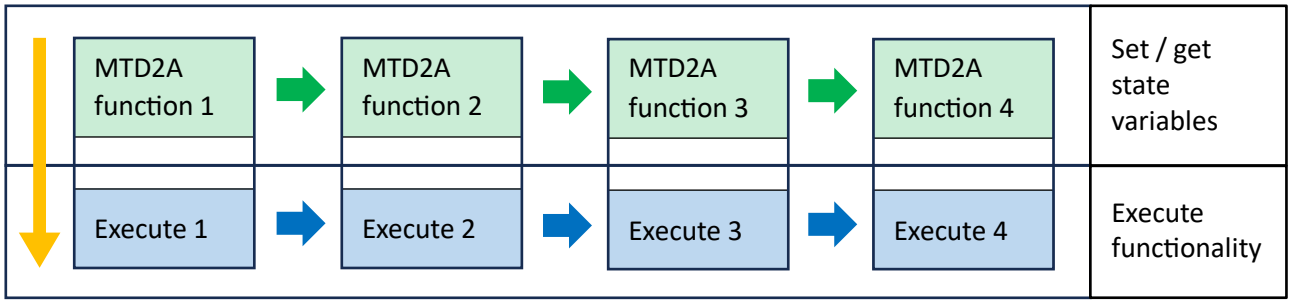
MTD2A & MTD2A_base 1
Virkemåde 2
To typer af proces flow 3
Bibliotek 5
MTD2A_const.h 5
MTD2A_base (h og cpp) 6
Tidsstyring og kadence 6
MTD2A_loop_execute() — Design og funktion 7
Fejl- og advarselsbeskeder 10
Aliases 11
Set og get funktioner 11
MTD2A_print_conf (); 11
Production 11
Eksempel på parallel processering 12

Virkemåde

MTD2A er en såkaldt tilstandsmaskine. Det betyder MTD2A objekter gennemløbes **hurtigt** i en uendelig løkke, og hvert gennemløb er opdelt i to faser:

1. Ændring af tilstandsvariable (ændre og/eller aflæse og kontrollere)
2. Eksekvere funktionalitet og kontrollere den totale gennemløbstid.

Konceptdiagram



Den øverste grønne proces gennemføres sammen med brugerdefineret kode og andre biblioteker.
Den nederste blå proces gennemføres via `MTD2A_loop_execute ();` og som det sidste i `void loop ();`

På denne måde opnås der en tilnærmet parallelisering, hvor flere funktioner i praksis kan eksekveres samtidigt. Fx to blinkende LED, hvor den ene er synkron, og den anden er asynkron. [Eksempel på Youtube]

De enkelte instantierede (aktiverede) objekter eksekveres i den rækkefølge de instantieres (aktiveres).

```
// Execution order MTD2A_loop_execute ();
MTD2A_binary_output LED_1 ("LED_1", 500, 500); // Execute first [1]
MTD2A_binary_output LED_2 ("LED_2", 500, 500); // Execute next  [2]
MTD2A_binary_output LED_3 ("LED_3", 500, 500); // Execute last  [3]
```

MTD2A biblioteket kan uden videre blandes med brugerdefineret kode og andre biblioteker, så længe eksekveringen foregår ikke blokerende (non blocking). Men det kræver en lidt anderledes tankegang når der udvikles kode, idet der hele tiden skal tages højde for, at den uendelig og hurtige løkke **ikke** må forsinkes, men også at brugerkode ikke eksekveres flere gange, end hvad der er tiltænkt. Det er ofte nødvendig at benytte forskellige former for logiske styringsflag.

Eksempel:

```
oneTimeFlag = true;
void loop() {
  // user or MTD2A code
  if (oneTimeFlag == true) {
    do_something_once ();
    oneTimeFlag = false;
  }
  // user or MTD2A code
}
```

Ikke blokerende eksekvering

Det er helt afgørende, at der ikke benyttes forsinkende eller blokerende kode sammen med MTD2A biblioteket. Der må ikke benyttes forsinkende funktioner som fx `delay ();` samt biblioteker der forhindrer hurtige gennemløb af den uendelig løkke. Brug `MTD2A_timer ();` Dog tillades som standard forsinkelser op til maksimalt 10 millisekunder pr. loop gennemløb, ved standard kadence = `DELAY_10MS`. Det er i et fleste tilfælde tilstrækkelig tid til at eksekvere brugerdefineret kode og forskellige former for biblioteker samtidigt. Se uddybende forklaring i senere afsnit.

I stedet for at benytte `delay ();` kan der tælles på antal gennemløb `loopCount`. Optælling er simpel og let at overskue, men optælling er ikke helt præcis. Afvigelsen fra MCU system tiden vokser en lille smule over tid.

Er der behov for præcision eller lange procestider, anbefales det at benytte `MTD2A_timer ();` og ESP32 MCU.

```
unsigned int loopCount = 0;
// standard MTD2A loop execution time is 10 Milliseconds

void loop() {
  switch (loopCount) {
    case  200:  do_things_1 ();  break;  //  2 seconds after startup
    case  400:  do_things_2 ();  break;  //  4 seconds after startup
    case 6000:  do_things_3 ();  break;  // 60 seconds after startup
  }
  loopCount++;
}
```

Se uddybende forklaring her: [Finite-state machine - Wikipedia]

To typer af proces flow

Der er grundlæggende to typer proces flows, og de kan kombineres

1. Forudsigeligt proces flow – tidsstyret
2. Uforudsigeligt proces flow – hændelsesstyret

Forudsigeligt procesforløb

Velegnet til forudsigelige processer, der følger en ikke-kritisk tidslinje. Eksempel:

```
// Timed cascading (round robin) 4 LEDs.
switch (loopCount) {
  case   0: green_LED_1.activate(); break; // Start at once
  case  50: green_LED_2.activate(); break; // 0.5 second
  case 100: red_LED_1.activate();   break; // 1   second
  case 150: red_LED_2.activate();   break; // 1.5 second
}
loopCount++;
if (loopCount >= 200) {
  loopCount = 0;
}
```

Uforudsigeligt procesforløb

Velegnet til hændelsesstyret procesforløb som f.eks. objektdetektion med en sensor. Eksempel:

```
// Read infrared sensor and blink LED.
switch (stepCount) {
  case 0:
    if (IR_sensor.get_processState() == ACTIVE) {
      // do something once when detected
      stepCount = 1;
    }
    // do something while waiting on ACTIVE
  break;
  case 1:
    // Add 500 millisecond pause delay to blink LED
    timer_GL1.timer (START_TIMER, 500);
    stepCount = 2;
  break;
  case 2:
    if (timer_GL1.get_processState() == COMPLETE) {
      // do something once
      green_LED_1.activate();
      stepCount = 3;
    }
    // do something while waiting on sensor is COMPLETE
  break;
  case 3:
    if (green_LED_1.get_processState() == COMPLETE) {
      // do something once
      stepCount = 0; // Start over
    }
    // do something while waiting on COMPLETE
  break;
}
```

Syv eksempler til inspiration på, hvordan man koder tids- og hændelsesprocesforløb

https://github.com/MTD2A/MTD2A/tree/main/examples
DEMO video: https://youtu.be/UU4k4_8GWfM

Det er også muligt at benytte **for** og **while** løkker, så længe `MTD2A_loop_execute ();` er en del af løkken

```
void loop() {
  do { // Wait for sensor to activate
    MTD2A_loop_execute ();
  } while (IR_sensor.get_processState() != ACTIVE);
  //
  for (i = 0; i < maxIndex; i++) {
    train_forward.set_timers (stepArr[i].curveMS, stepArr[i].delayMS, stepArr[i].maintainMS);
    train_forward.activate (stepArr[i].beginValue, stepArr[i].endValue, stepArr[i].PWMcurve);
    do {
      MTD2A_loop_execute ();
    } while (train_forward.get_processState() == ACTIVE);
  }
}  // loop ()
```

Se eksempel her: MTD2A/examples/pendulum_H_bridge at main · MTD2A/MTD2A · GitHub

Bibliotek

```
#include <MTD2A.h>  // Inkluder alle biblioteksfiler
using namespace MTD2A_const; // Brugervenlig navnekonstanter
```

Nuværende byggeklodser

- MTD2A_binary_input
- MTD2A_binary_output
- MTD2A_timer

Yderligere planlagte byggeklodser

- MTD2A_tone
- MTD2A_DCC_input

MTD2A_const.h

Indeholde brugervenlig og letforståelige globale konstanter til anvendelse som parametre til de forskellige funktioner og i den kode brugeren skriver. Herunder et udvalg af globale logiske konstanter.

ENABLE	DISABLE
ACTIVE	COMPLETE
FIRST_TRIGGER	LAST_TRIGGER
TIME_DELAY	MONO_STABLE
NORMAL	INVERTED
PULSE	FIXED
BINARY	P_W_M

Timer konstanter

DELAY_10MS DELAY_5MS DELAY_2MS DELAY_1MS MAX_TIME_MS

Namespace

Brugen af namespace er en metode til at håndtere navnesammenfald med andre biblioteker. Som udgangspunkt kan alle de globale konstanter gøres tilgængelige ved i starten af brugerprogrammet ved at skrive: `using namespace MTD2A_const;` Hvis der er navnsammenfald, skal hver af de globale konstanter der benyttes gøres tilgængelige i brugerprogrammet. Det kan gøres på to måder som vist på eksemplet herunder:

`MTD2A_const::ENABLE;` Benyttes hver gang ENABLE benyttes. `::` = Scope Resolution Operator.

`using MTD2A_const::ENABLE;` Gør ENABLE til en global konstant I brugerprogrammet.

MTD2A_base (h og cpp)

Indeholder styringsfunktioner, fælles interne funktioner og globale fælles funktioner.

MTD2A_loop_execute (); eksekverer alle instantierede objekter (Linked list of function pointers), og skal altid kaldes en gang i som den sidste kommando i void loop (); MTD2A_loop_execute (); kan kaldes flere gange i samme loop, hvis der er behov for det, men vær opmærksom på at flere tilstandsflag kan ændre status inden de kan nå at aflæses og handles på. Fx processState og phaseChange

MTD2A::get_globalObjectCount (); returner antallet af instantierede (aktiverede) MTD2A objekter.

MTD2A_globalDebugPrint (); Aktiver fejl og status meddelelser til Arduino IDE Serial Monitor, og er Initialiseret til **DISABLE** Logisk parameter = { ENABLE | DISABLE } Udelades parameter sættes default = **ENABLE**.

MTD2A_globalErrorPrint (); Aktiver fejlmeddelelser til Arduino IDE Serial Monitor. Initialiseret til **ENABLE** Logisk parameter = { ENABLE | DISABLE } Udelades parameter sættes default = **ENABLE**

Tidsstyring og kadence

Synkronisering

For at sikre at sikre at alle instantierede (aktiverede) objekter "går i takt", benytter alle objekter et fælles udgangspunkt for tidsfastsættelse. Tiden sættes én gang i hvert void loop (); som det første trin i funktionen MTD2A_loop_execute (); Dvs. **efter** at brugerkode og MTD2A funktioner er eksekveret, og før alle interne styrings- og tidskontrol funktioner er eksekveret (instantierede MTD2A objekter).

Kadence

For at sikre at sikre at alle instantierede (aktiverede) objekter følger en forudsigelig og ensartet tidsperiode, beregner funktionen MTD2A_loop_execute (); gennemløbstiden af alle instantierede MTD2A objekter, den kode som brugeren har tilføjet, og de biblioteker brugeren benytter. Det samlede tidsforbrug korrigeres efter hvert gennemløb i forhold til en parallel præcis beregnet total tidsmåling. Dette sikrer præcision. Funktionen modregner forsinkelser op til 10 millisekunder pr. samlet gennemløb void loop (); ved standard kadence = DELAY_10MS

Standard kadence = 10 millisekunder = loop time. **Rød linje:** Ny fælles tidtagning til tidskontroller. Gul linje: Modregning af forsinkelser. Styret tidsforsinkelse sikrer ensartet kadence.		User code, libraries & MTD2A functions	Elapsed time A
		MTD2A_loop_execute ()	Elapsed time B
		MTD2A delay()	Delay (10 - A – B)

Hvis beregningstiden går ud over globalDelayTimeMS kan der opstår problemer med at fastholde en ensartede kadence. Der kompenseres for spontane uforudsete tidsforsinkelser ud over globalDelayTimeMS. For at synkronisere med medgået tid, forøges totalLoopCount med det antal, der svarer til tidsoverløbet. Vedvarende forsinkelser skaber en uensartede kadence, og det kan opstå asynkrone problemer. Især ved tidskritiske og hurtigt eksekverende funktioner.

Præcisionen i tidsmålingerne er +/- 10 millisekunder for DELAY_10MS og +/- 1 millisekund for DELAY_1MS

MTD2A::set_globalDelayTimeMS (); Sætter kadencen i millisekunder hvormed alle instantierede (aktiverede) objekter eksekveres. Funktionen nulstiller tidsmålingen og kalder MTD2A_reset_stats ();
Parameter = { DELAY_10MS | DELAY_5MS | DELAY_2MS | DELAY_1MS }
Udelades parameter sættes default **DELAY_10MS**

MTD2A::get_globalDelayTimeMS (); Returnere den aktuelle tid for eksekveringskadence i millisekunder. Default DELAY_10MS.

MTD2A::get_MaxElapsedTimeMS (); Returnerer den maksimale målte tidsforsinkelse i millisekunder for hvert gennemløb samlet set. Dvs. tidsforsinkelsen for gennemløb void loop (); af instantierede MTD2A objekter (kode), samt den kode som brugeren har tilføjet. Funktionen er primært rettet mod at identificere bruger kode, og de biblioteker brugeren benytter, der forsinker gennemløbstiden mere end MTD2A::get_globalDelayTimeMS (); Millisekunder. Det kan medføre kadence- og synkroniseringsproblemer.

MTD2A::get_timeOverrunCount (); Tæller hvor mange gange gennemløbstiden void loop (); overstiger MTD2A::get_globalDelayTimeMS (); Det sker ofte hvis der skrives længere tekster til *IDE Serial Monitor* og især hvis kadence er defineret til DELAY_1MS, og hvis der anvendes mindre kraftige Arduino boards. Fx NANO.

MTD2A::get_totalLoopCount (); Returnerer antallet af gennemløb. Nulstilles efter ca. 72 minutter tidsmåling.

MTD2A::get_globalSyncTimeMS (); Returnerer den aktuelle tid som alle instantierede (aktieverede) objekter benytter til tidskontrol. Nulstilles efter ca. 72 minutters tidsmåling.

MTD2A_reset_stats (); nulstiller målevariable: lastElapsedTimeMS, maxElapsedTimeMS, timeOverrunCount

Eksekveringshastighed

Som udgangspunkt er de mindre kraftige Arduino boards tilstrækkelige Fx MEGA,UNO og NANO. Hvis der benyttes mange instantierede (aktiverede) MTD2A objekter, eller CPU krævende PWM kurver beregninger, anbefales at benytte de mere kraftige Arduino boards (MCU) som fx ESP32 serien.

Gennemløbstiden kan stige betydeligt ved benyttelse af sensorer med egen MCU (fx VL53L4CD laser afstandsmåler), kommunikationsprotokoller fx UART samt skrivning til Arduino IDE Serial Monitor.

Fx tager det 7-8 millisekunder at forespørge VL53L4CD om data er klar, og efterfølgende aflæse data. Kommunikationsprotokoller bør, om muligt, konfigureres til høj hastighed. Det samme gælder for Serial Monitor, der ofte benytter den langsomme 9600 BAUD hastighed.

Hvis der er behov for meget hurtig aflæsning og eksekvering, kan DELAY_1MS vælges. Fx hurtige aflæsning af infrarød sensor til nødstop, eller mere præcis tidsstyring. Her anbefales det at benytte de mere kraftige Arduino boards (MCU) som fx ESP32 serien.

MTD2A_loop_execute() — Design og funktion

MTD2A_loop_execute (); er den centrale timing-motor i MTD2A-biblioteket. Den kaldes gentagne gange fra slutningen af Arduino void loop (); funktionen og leverer:

- **En konstant, forudsigelig kadence** (f.eks. præcis 10 millisekunder pr. loop-iteration)
- **En synkroniseret referencetid** (globalSyncTimeUS), der deles af alle instantierede objekter
- **Automatisk driftkorrektion**, så den akkumulerede tid forbliver nøjagtig over tusindvis af gennemløb
- **Diagnostik til overvågning** af kodeudførelsestid og detektering af overløb

Designprincip — Epokeforankret timing

Traditionelle Arduino-timingloops måler, hvor lang tid hver iteration tog, og forsinker resten:

```
delay (interval - forløbet) // relativ: fejl akkumuleres.
```

Denne tilgang akkumulerer afrundingsfejl, interupt-afbrydelser og måleoverhead over tusindvis af loops. Efter 60.000 loops ved 10 ms kan den samlede forløbne tid afvige med hundredvis af millisekunder.

MTD2A_loop_execute (); bruger i stedet en epokeforankret referencetilgang. Ved det første kald registreres en absolut starttid (epoken). Måltiden for hver efterfølgende gennemløb beregnes direkte fra dette anker:

targetTimeUS = epochTimeUS + (totalLoopCount × delayTimeUS)

Dette betyder, at målet for løkke N altid er relativt til den oprindelige tidsfastsættelse, ikke til det forrige loop. Hvis et loop er 200 mikrosekunder lang, får det næste loop automatisk 200 mikrosekunder mindre forsinkelse. Fejl fra individuelle løkker akkumuleres ikke.

Resultat: Efter N gennemløb konvergerer den samlede forløbne tid til N × interval, uanset jitter pr. loop.

Eksekveringsflow

1. Registrer aktuel tid (globalSyncTimeUS)
2. Initialisering af første kald (epoch anker)
3. Eksekver alle registrerede instantierede objekter
4. Mål kodeudførelsestid
5. Overløbskorrektion
6. Gap-detektion og genforankring
7. Epokeforankret forsinkelse
8. Fremløbsløjfetæller (totalLoopCount)

BEMÆRK: MTD2A_loop_execute (); Bruger Arduino micros(); til præcisionsberegning, og vil løbe over (gå tilbage til nul) efter 71,58 minutter ≈ 4.294.967.295 (0xFFFFFFFF) mikrosekunder. MTD2A håndterer både overflow og underflow uden problemer ved alle beregninger. Alle tidsberegninger er præcise. Det vil dog have indflydelse på aflæsninger og udskrivning af forskellige variabler til tidsmåling, som vil ændre sig til et meget lille tal i tilfælde af overløb. Integer overflow - Wikipedia

Overflow eksempel fra MTD2A_timer (); (afrundede millisekunder – ikke mikrosekunder)

Sidste gennemløb før overløb	Næste gennemløb
MTD2A_timer: ---------- startTimeMS : 4.294.952 stopTimeMS : 4.294.962	MTD2A_timer: ---------- startTimeMS : 4.294.962 stopTimeMS : 5

Initial nulstilling af timere til beregninger

Anker og reference tidsmåling nulstilles ved første gennemløb. Men på dette tidspunkt kendes tidsforsinkelse ikke, som følge af bruger kode og de biblioteker brugeren benytter. Det betyder at første gennemløb kan tage længe tid end forventet. For at sikre præcis tidsmåling kan MTD2A_loop_execute (); Kaldes i void setup ();

```
void setup() {
  Serial.begin(250000);
  while (!Serial) { delay(1); } // ESP32 Serial Monitor ready delay
  // One initialization loop to set epoch and calculate correct target time.
  MTD2A_loop_execute();
}
```

Eksempel på overflow håndtering

```
uint32_t beginTime, elapsedTime, endTime;

void setup() {
  Serial.begin(250000);
  while (!Serial) { delay(10); } // ESP32 Serial Monitor ready delay

  Serial.println("micros() and millis() test of correct overflow handling");
  Serial.println("Elapsed\tBegin\t\tEnd");
  beginTime = 4294967293;   endTime = 4294967294;
  for (byte index = 0; index < 6; index++) {
    elapsedTime = endTime - beginTime;
    Serial.print(elapsedTime); Serial.print(":\t"); Serial.print(beginTime);
    if (beginTime < 10)   Serial.print("\t\t");   else   Serial.print("\t");
    Serial.println(endTime);
    beginTime = endTime;
    endTime++;
  }
} // setup

void loop() {
  delay (10000);
}
```

Output from overflow handling

```
micros() and millis() test of correct overflow handling
Elapsed         Begin                End
1:        4294967293        4294967294
1:        4294967294        4294967295
1:        4294967295                 0
1:                 0                 1
1:                 1                 2
1:                 2                 3
```

Fejl- og advarselsbeskeder

Number	Error text
1	Pin number not defined (255)
2	Digital pin number out of range
3	Analog pin number out of range
4	Output pin already in use
5	Pin does not support PWM
6	tone() conflicts with PWM pin
7	Pin does not support interrupt
8	Must be INPUT or INPUT_PULLUP
9	Timer set to globalDelayTimeMS = X
10	Timer set to MAX_TIME_MS
11	Pin write is disabled
12	Process state must be COMPLETE
13	Select STOP_TIMER or RESET_TIMER
14	Unknown TIMER argument
15	globalDelayTimeMS must be: 1 - 10 MS
16	Process state must be ACTIVE
17	Unforeseen system error
18	Must be ACTIVE and timer configured
19	Already initialized
20	Object instantiation error / warning

Number	Warning text
128	Digital Pin check not possible
129	Analog Pin check not possible
130	Pin used more than once
131	PWM Pin check not possible
132	Interrupt Pin check not possible
133	PWM disabled: Timer claimed by another library
140	Timer value is zero
141	Time + pause exceeds MAX_TIME_MS
142	setCountDownMS argument is ignored
150	Output timer value is zero
151	All three timers are zero
152	Binary pin value > 1. Set to HIGH
153	Undefined PWM curve
154	Use RISING curve instead of FALLING
155	Use FALLING curve instead of RISING
156	BeginValue = endValue => NO_CURVE
157	Time = globalDelayTime => NO_CURVE
158	PAUSE already active
159	loopActivate ENABLE is ignored
160	Value not stepping by globalDelayTimeMS: X

Aliases

MTD2A_loop_execute ();	Er et mere brugervenligt alias for	MTD2A::loop_execute ();
MTD2A_globalDebugPrint ();	Er et mere brugervenligt alias for	MTD2A::set_globalDebugPrint ();
MTD2A_globalErrorPrint ();	Er et mere brugervenligt alias for	MTD2A::set_globalErrorPrint ();
MTD2A_print_conf ();	Er et mere brugervenligt alias for	MTD2A::print_conf ();
MTD2A_reset_stats ();	Er et mere brugervenligt alias for	MTD2A::reset_stats ();

Set og get funktioner

Set functions Grøn er standard, ved ingen parameter	**Comment**
set_globalDebugPrint ({ ENABLE | **DISABLE** });	Print phase state number and state text
set_globalErrorPrint ({ **ENABLE** | DISABLE });	Print error and warning text
set_globalDelayTimeMS ({ DELAY_1MS | DELAY_2MS | DELAY_5MS | DELAY_10MS });	Main loop delay in milliseconds and calls reset_stats ();

Get functions	**Comment**
get_globalDelayTimeMS (); Return uint8_t milliseconds	Main loop delay time
get_globalSyncTimeMS (); Return uint32_t milliseconds	Current common reference time Resets after approx 72 minutes time delay
get_globalObjectCount (); Return uint8_t count	Number of instantiated MTD2A objects
get_maxElapsedTimeMS ();Return uint32_t milliseconds	max MTD2A code, user code and other library loop execution time.
get_timeOverrunCount (); Return uint32_t count	Number of times loop time that exceeds globalDelayTimeMS
get_totalLoopCount (); Return uint64_t count	Number of loop count since startup. Resets after approx 72 minutes time delay

MTD2A_print_conf ();

```
MTD2A_base:
--------------------
  globalDebugPrint : DISABLE
  globalErrorPrint : ENABLE
  globalObjectCount: 4
  globalDelayTimeMS: 10
  globalSyncTimeMS : 10978
  lastElapsedTimeMS: 0.00
  maxElapsedTimeMS : 0.00
  timeOverrunCount : 0
  totalLoopCount   : 839
```

Production

For at minimere forsinkelser og sikre maksimal præcision, bør den relative langsomme skrivning til *Serial Monitor* blokeres som det sidste, før programmet overgår til produktion. **MTD2A_print.h**

```
// Development an debug
#define PortPrint(x)   Serial.print(x)
#define PortPrintln(x) Serial.println(x)
// Optimized production
// #define PortPrint(x)
// #define PortPrintln(x)
```

Eksempel på parallel processering

```
// Two flashing LEDs. One with symmetric interval & another with asymmetric interval.
// https://github.com/MTD2A/MTD2A/blob/main/examples/blink_LED/blink_LED.ino

#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 sec light, 0.7 sec no light, PWM dimmed

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

  red_LED.initialize   (9);   // Output pin 9
  green_LED.initialize (10);  // Output pin 10

  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/eyGRazX9Bko