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 / 28-06-2025

MTD2A er en samling af brugervenlige, avancerede og funktionelle C++ klasser - byggeklodser - til tidsstyret håntering af input og output. Biblioteket er tiltænkt Arduino interesserede uden større programmerings-erfaring, der har elektronikstyring og -automatisering som interesse, samt modeltog som hobby.

Fælles for alle byggeklodser:

- 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 tilstandsdrevet
- Tilbyder omfattende kontrol- og fejlfindingsinformation
- Grundigt dokumenterede med eksempler

Indhold

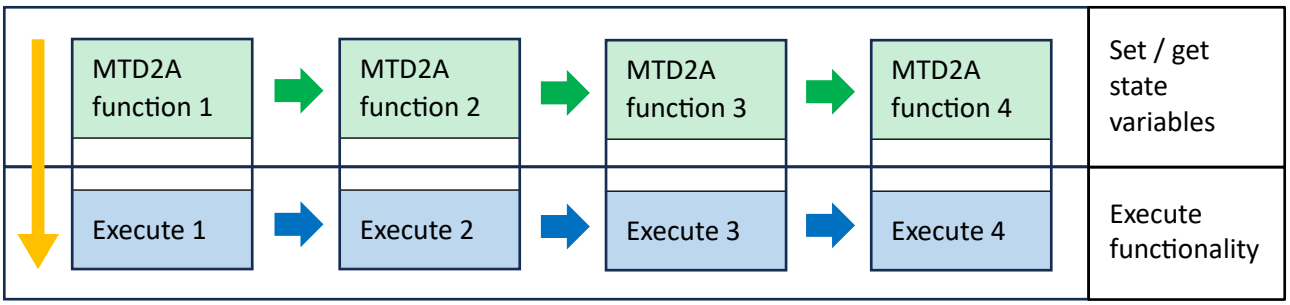
MTD2A & MTD2A_base 1
Virkemåde 1
Bibliotek 2
MTD2A_const.h 3
MTD2A_base (h og cpp) 3
Tidsstyring og kadance 4
Fejl – og advarselsbeskeder 5
Eksempel på parallel processering 6

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 tilstandsvariabele (ændre og/eller aflæse og kontrollere)
2. Eksekvere funktinalitet og kontrollere tidsforløb.

Konceptdiagram



Den øverste grønne proces gennemføres sammen med brugerdefinteret kode og andre biblioteker.

Den nederste blå process gennemføres via `MTD2A_loop_execute ();` og som det sidste i `void loop ();`

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

MTD2A biblioteket kan uden videre blandes med brugerdefineret kode og andre bilblioteker, så længe ekeveringen 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 må ikke 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 bibliotektet. Der må ikke benyttes forsinkende funktioner som fx `delay ();` samt bibliotker der forhinde hurtige gennemløb af den uendelig løkke. Dog tillades som standard forsinkelser op til maksimalt 10 millisekunder pr. gennemløb. Det er i et fleste tilfælde rigelig tid til at eksekvere brugerdefineret kode og forskellige former for bibliotker samtidigt. Se uddybende forklaring i senere afsnit.

Se mere her: Finite-state machine - Wikipedia og her: Real-time operating system - Wikipedia

Bibliotek

```
#include <MTD2A.h>
using namespace MTD2A_const;
```

Nuværende byggeklodser

- MTD2A_binary_input.h
- MTD2A_binary_output.h

Yderligere planlagte byggeklodser

• MTD2A_timer • MTD2A_astable • MTD2A_flip_flop • MTD2A_tone • MTD2A_sound • MTD2A_servo	• MTD2A_stepper • MTD2A_display • MTD2A_ultrasonic • MTD2A_laser • MTD2A_IR_ranging • 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.

ENABLE	DISABLE
ACTIVE	COMPLETE
FIRST_TRIGGER	LAST_TRIGGER
TIME_DELAY	MONO_STABLE
NORMAL	INVERTED
PULSE	FIXED
BINARY	P_W_M
RESTART_TIMER	STOP_TIMER
DELAY1MS	DELAY_10MS

Procesfaser tilhørende de enkelte moduler

RESET_PHASE = 0
BEGIN_PHASE = 1, OUTPUT_PHASE = 2, END_PHASE = 3
FIRST_TIME_PHASE = 1, LAST_TIME_PHASE = 2, BLOCKING_PHASE = 3
COMPLETE_PHASE = 4

Brugen af namespace er en metode til at håndtere navnesammenfald med andre biblioteker. Som adgangspunkt kan alle de globale konstanter gøres tilgængelige ved i starten af brugerprogrammet at skrive: `using namespace MTD2A_const;` Hvis der er navnsammenfald, skal hver af de globale konstanter der benttes gøres tilgængelige i brugerprogrammet. Det kan gøres på to mådersom vist på eksemplet herunder:
`MTD2A_const::ENABLE;` Benytters hver gang ENABLE beyttes.
`using MTD2A_const::ENABLE;` Gør ENABLE til en global konstant I brugerprogrammet.
:: = Scope Resolution Operator

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)
Absolut sidste kommando i `void loop ();` og skal altid kaldes en - og kun - en gang.

`MTD2A_globalDebugPrint ();` Aktiver fejl og status medddelelser til Arduiono IDE Serial Monitor.
Parameter = ({ENABLE | DISABLE}) Udelades parameter sættes default = ENABLE

`MTD2A_delayTimeMS ();` Sætter kadancen hvormed alle instantierede (aktiverede) objekter opdtateres.
Parameter = ({ DELAY_10MS | DELAY_1MS}) Udelades parameter sættes default DELAY_10MS

`MTD2A_maxLoopMS ();` Funktionen returnerer den maksimale målte tidsforsinkelse for hvert genneløb totalt set, og nulstiller samtidigt målingen. 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 brugen benytter, der forsinker genneløbstiden mere end `MTD2A_delayTimeMS ();` Det kan medføre kadance- og synkronisseringsproblemer.

`MTD2A_objectCount ();` Funktionen returner antallet af instantierede (aktiverede) MTD2A objekter.

Tidsstyring og kadance

Synkronisering

For at sikre at sikre at alle instantierede (aktiverede) objekter ”går i takt”, benytter alle objekter et fælles udgangspunkt for tistagning. Tiden sættes én gang i hvert `void loop ();` , efter brugerkode er eksekveret og før det første objekt eksekveres i funktiontion `MTD2A_loop_execute ();`
Den aktuelle tidstagning kan aflæses med funktion: `MTD2A_globalSyncTimeMS`

Kadance

For at sikre at sikre at alle instantierede (aktiverede) objekter følger en forudsigelig og ensarted tidsperiode, beregner funktionen `MTD2A_loop_execute ();` gennemløbstiden af alle instantierede MTD2A opbjekter, den kode som brugeren har tilføjet, og de biblioteker brugeren benytter. Funktionen modregner op til 10 millisekunder pr. samlet gennemløb `void loop ();` , ved standard kadance = `DELAY_10MS`

Hvis beregningstiden går ud over de 10 millisekunder kan der opstår problemer med at fastholde en ensartede kadance. Jo længere beregningstiden går ud over de 10 millisekuner, jo længere og mere uensartede kadance, og det kan opstå asynkrone problemer. Især ved tidskritiske og hurtigt eksekverende funktioner. I praksis kan det i mange tilfælde fungere, med mindre variationer, ved en beregningstid på op til 100 millisekunder.

Eksekveringshastighed

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

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

Fx tager det 7-8 millisekunder at forspørge VL53L4CD om data er klar, og efterfølgende aflæse data. Kommunikationsprotokoller bør, om muligt, konfigureres til høj hastighed. Det samme gælderfor Serial Monitor, der som standard benytter den relativt langsomme 9600 BAUD hastighed.

Forsinkelse ved faseskift

Hvert faseskift medfører et ekstra gennemløb på 1 - 10 millisekuner. Skal tidstagningen være helt præcis, skal det modregnes i de tider der angives når objektet instantieres (aktiveres).

Eksempel

Der ønskes et præcist samlet tidsforbrug på i alt 1.000 millisekunder.
`MTD2A_binary_output green_LED (”Green LED”, 490, 490);` 490 + 10 + 490 + 10 = 1.000 millisekunder.

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

Fejl – og advarselsbeskeder

Nuber	Error text
1	Pin not defined
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	Pin number not set (255)
11	Pin write is disabled
	Warning text
128	Pin used more than once
130	Timer value is zero
140	Output value is zero
141	All three timers are zero

MTD2A_print_conf ();

```
MTD2A_base:
----------------
 globalDebugPrint: DISABLE
 globalErrorPrint: ENABLE
 globalSyncTimeMS: 10034
 delayTimeMS     : DELAY_10MS
 lastTimeMS      : 10044
 loopTimeMS      : 0
 maxLoopTimeMS   : 1
 objectCount     : 4
```

Eksempel på parallel processering

```
 // Two blinking LEDs. One with symmetric interval and another with asymetric interval.
// Jørgen Bo Madsen / may 2025 / https://github.com/jebmdk

#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 sec light, 0.7 sec no light, PWM dimmed

void setup() {
  Serial.begin(9600);

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