MTD2A_timer

MTD2A: Model **T**rain **D**etection **A**nd **A**ction – Arduino library https://github.com/MTD2A/MTD2A
Jørgen Bo Madsen / V1.3.3 / 14-10-2026

MTD2A_timer er en brugervenlig avanceret og funktionel C++ klasse til ikke-blokerende tidsstyring.
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
- Fungerer Ikke-blokerende, procesorienteret og tilstandsdrevet
- Tilbyder omfattende kontrol- og fejlfindingsinformation
- Grundigt dokumenterede med eksempler

Indholdsfortegnelse

MTD2A_timer 1

Funktionsbeskrivelse 1

Proces faser 2

Eksempel på tidsstyret proces 3

Print_conf(); 4

Set og get funktionsoversigt 5

Funktionsbeskrivelse

MTD2A_timer processen består af 3 funktioner:

1. `MTD2A_timer object_name (”object_name” , countDownMS );`
2. `object_name.timer ( { RESET_TIMER | START_TIMER | PAUSE_TIMER | STOP_TIMER }, countDownMS } );`
   Kaldes i `void setup ();`
3. `MTD2A_loop_execute ();` Kaldes som det sidste I `void loop ();`

Første argument benytter default værdi og funktionen kan kaldes med ingen og op til 2 argumenter.

```
MTD2A_timer object_name;
MTD2A_timer object_name (”Object_name”);
MTD2A_timer object_name (”Object_name”, countDownMS);
Default:                          (”Object  name”, 0                    );
```

Hvis `countDownMS` = 0 afsluttes `object_name.timer ()` uden videre og advarsel 140 registreres.

Hvis `object_name.timer ()` kaldes med argumenterne `{ PAUSE_TIMER | STOP_TIMER }, countDownMS }` ignoreres `countDownMS` og advarsel 142 registreres.

NOTE: Kald ikke MTD2A_timer fra en interrupt-handler (ISR).

Eksempel på eventstyret proces

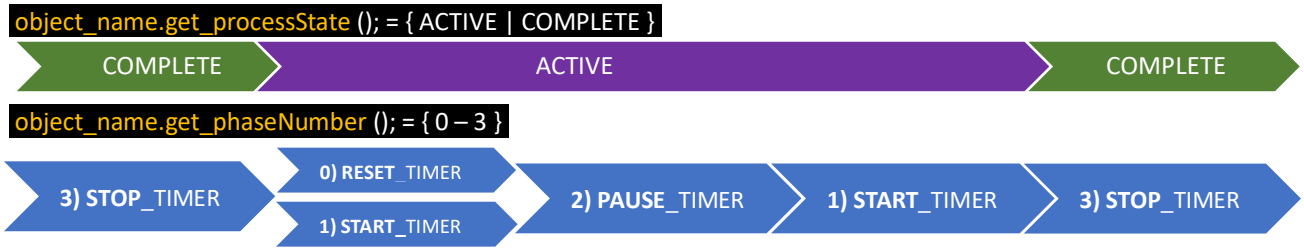
```
// Event if-condition controlled. Recommended for simple solutions
// Read infrared sensor. Short detection: One LED blink.
// continuously detection: continuously LED blink
// Suitable for Unpredictable process flows such as object detection with a sensor
if (IR_sensor.get_processState() == ACTIVE) {
  Serial.println ("Object detected");
  if (timer_GL1.get_processState() == COMPLETE) {
    timer_GL1.timer (START_TIMER, 1000); // 1   second
    green_LED_1.activate (500);          // 0,5 second
  }
}
```

Flere eksempler og Youtube demo video:

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

Proces faser

Afhængig af den aktuelle konfiguration gennemføres processen i mellem 2 og multiple faser med pauser.



3. Den initiale fase når programmet starter STOP_TIMER.
0. Når funktion object_name.timer (RESET_TIMER); kaldes. Kan forekomme når som helst
1. Når funktion object_name.timer (START_TIMER); kaldes. Kan forekomme flere gange efter hver pause.
2. Når funktion object_name.timer (PAUSE_TIMER); kaldes. Kan forekomme flere gange efter start / reset.
3. Når funktion object_name.timer (STOP_TIMER); kaldes, eller når tiden er udløbet. countDownMS = 0.

Globale nummerkonstanter:
RESET_TIMER, START_TIMER, PAUSE_TIMER og STOP_TIMER

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

Proces status

Ved overgang til START_TIMER eller RESET_TIMER skifter ProcessState, til ACTIVE.
Ved overgang til STOP_TIMER skifter processState til COMPLETE.
ProcessState forbliver ACTIVE under PAUSE_TIMER.

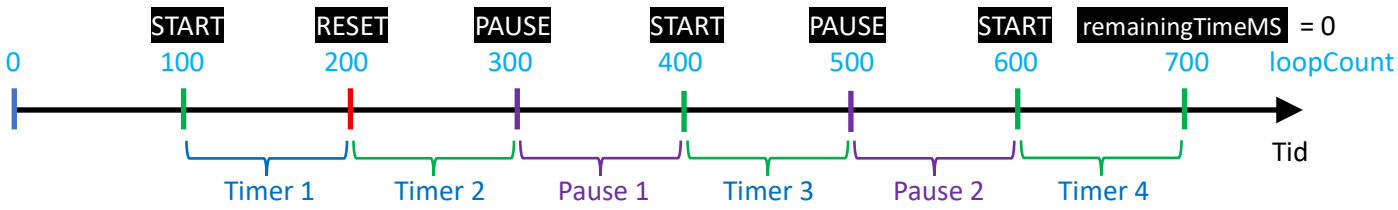
Timing

Tidsperioden kan sættes når objektet instantieres. Efterfølgende kan nye tidsperioder defineres med:
object_name.set_countDownMS ({ 0 – MAX_TIME_MS }); Men kun når processState er COMPLETE

Se dokumentet MTD2A.PDF og afsnittet ”Kadence”, ”Synkronisering” samt ”Eksekveringshastighed”.

Eksempel på tidsstyret proces

Timer funktion aktiveres med 1000 millisekunders interval (100 loopCount).



Timer 1 udgår
elapsedTimeMS = Timer 2 + Timer 3 + Timer 4 = ca. 3.000 millisekunder.
remainingTimeMS = countDownMS – elapsedTimeMS = 0 millisekunder.
pauseTimeMS = Pause 1 + Pause 2 = ca. 2.000 millisekunder.

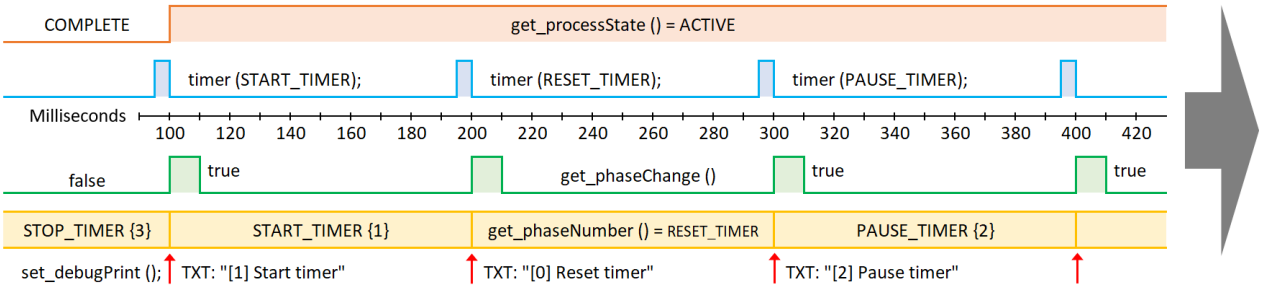
MTD2A_timer T1 ("T1" , 3000);

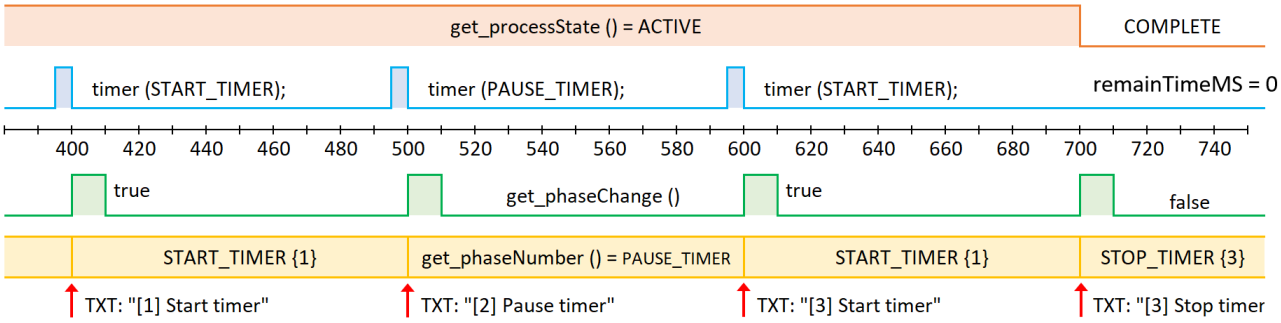
```
// Standard Loop executing time = 10 milliseconds
if (loopCount ==   0) {  T1.set_debugPrint ();    }
if (loopCount == 100) {  T1.timer (START_TIMER);  }  // 1 second
if (loopCount == 200) {  T1.timer (RESET_TIMER);  }  // 2 seconds
if (loopCount == 300) {  T1.timer (PAUSE_TIMER);  }  // 3 seconds
if (loopCount == 400) {  T1.timer (START_TIMER);  }  // 4 seconds
if (loopCount == 500) {  T1.timer (PAUSE_TIMER);  }  // 5 seconds
if (loopCount == 600) {  T1.timer (START_TIMER);  }  // 6 seconds

loopCount++;
if (T1) { // Equivalent to: phaseChange == true  &&  phaseNumber == STOP_TIMER;
  T1.print_conf ();
  loopCount = 0;
}
```

IDE Serial Monitor

```
36:04 -> T1 [1] Start timer
36:05 -> T1 [0] Reset timer
36:06 -> T1 [2] Pause timer
36:07 -> T1 [1] Start timer
36:08 -> T1 [2] Pause timer
36:09 -> T1 [1] Start timer
36:10 -> T1 [3] Stop timer
```





Pause time begrænsning

Det samlede tidsbudget for én timer-kørsel er begrænset: **countDownMS + akkumuleret pausetid må ikke overstige MAX_TIME_MS** (4.200.000 millisekunder ≈ 70 minutter). Årsagen er, at alle tidsberegninger bruger 32-bit mikrosekund aritmetik, som overflower efter cirka 71,6 minutter. Grænsen efterlader en sikkerhedsmargin under overflow-punktet.

Grænsen overvåges automatisk i hver løkke, mens timeren er sat på pause. Hvis tiden overskrides, giver timeren ikke forkerte resultater — i stedet stoppes den sikkert, og advarsel **141** udsendes: *"Tid + pause overstiger MAX_TIME_MS"*

PAUSE_TIMER er beregnet til pauser af begrænset varighed. Hvis en proces skal vente længe eller uforudsigeligt (for eksempel en sensorbegivenhed, der kan tage timer), må timeren ikke stå på pause. STOP timeren i stedet, og RESET_TIMER / START_TIMER den igen, når processen genoptages.

Print_conf();

object_name.print_conf ();

Maksimal tidspræcision med Arduino ESP32 og kadence DELAY_10MS

```
MTD2A_timer:
--------------------
 objectName    : T1
 processState  : COMPLETE
 phaseText     : [3] Stop timer
 debugPrint    : ENABLE
 globalDebugPr: DISABLE
 errorPrint    : DISABLE
 globalErrorPr: ENABLE
 errorNumber   : 0 OK
 countDownMS   : 3000
 elapsedTimeMS: 3000
 remainTimeMS  : 0
 startTimeMS   : 4144
 stopTimeMS    : 9144
 pauseTimeMS   : 2000
 pauseBeginMS  : 7144
 pauseEndMS    : 8144
```

Hvis der er flere pauser undervejs i tidsperioden, vises den samlede pause tid pauseTimeMS, men pauseBeginMS og pauseEndMS vises kun for den sidste pause.

MTD2A_print_conf ();

```
MTD2A_base:
--------------------
  globalDebugPrint : DISABLE
  globalErrorPrint : ENABLE
  globalObjectCount: 1
  globalDelayTimeMS: 10
  globalSyncTimeMS : 8766
  lastElapsedTimeMS: 0.00
  maxElapsedTimeMS : 0.00
  timeOverrunCount : 0
  totalLoopCount   : 701
```

Set og get funktionsoversigt

Set functions	Comment
set_countDownMS({0 – MAX_TIME_MS});	Set new count down time in milliseconds
set_debugPrint ({ENABLE | DISABLE});	Activate print phase number and text
set_errorPrint ({ENABLE | DISABLE});	Activate error messages

Get functions	Comment
get_processState (); return bool {ACTIVE|COMPLETE}	Process state
get_phaseChange (); return bool {true | false}	Momentarily phase change (one loop time)
get_phaseNumber (); return uint8_t {0 - 3}	RESET_TIMER = 0, START_TIMER =1, PAUSE_TIMER = 2, STOP_TIMER = 3
get_startTimeMS (); return uint32_t milliseconds	Time set at start/reset
get_stopTimeMS (); return uint32_t milliseconds	Time set at stop
get_pauseTimeMS (); return uint32_t milliseconds	Accumulated pause time (sum of all pause periods). Zero if no pause initiated
get_remaingTimeMS(); return uint32_t milliseconds	Remaining time since start/reset
get_elapsedTimeMS(); return uint32_t milliseconds	elapsed time since start/reset
get_reset_error (); return uint8_t {0-255}	Get error/warning number and reset Error [1 – 127] warning [128 – 255]

Operator overloading	Function
object_name & !objectName	bool phaseChange == true && phaseNumber == STOP_TIMER
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