MTD2A_timer

MTD2A: Model **T**rain **D**etection **A**nd **A**ction – arduino library https://github.com/MTD2A/MTD2A
Jørgen Bo Madsen / V1.2 / 09-10-2025

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
- Fungere Ikke-blokerende, procesorienteret og tilstandsdrevet
- Tilbyder omfattende kontrol- og fejlfindingsinformation
- Grundigt dokumenterede med eksempler

Indholdsfortegnelse

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 argumenet 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                    );
```

Eksempel på eventstyret process

```
// 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

Afhægig af den aktuelle konfiguration gennemføres processen i mellem 2 og multible faser med pauser.

object_name.get_processState (); = { ACTIVE | COMPLETE }

object_name.get_phaseNumber (); = { 0 – 3 }

3. Den initelle 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 fere 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 fasesskift 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.

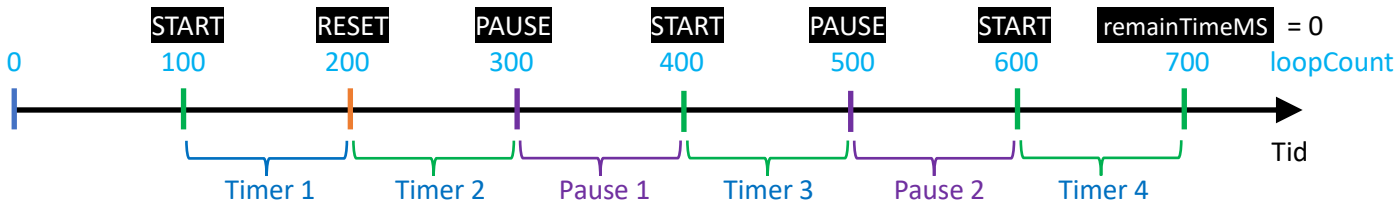
Timing

Tidsperioden kan sættes når objektet instantieres. Efterfølgende kan nye tidsperioder defineres med:
object_name.set_countDownMS ({0 - 4294967295}); Men kun når processState er COMPLETE

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

Eksempel på tidsstyret process

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



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

MTD2A_timer object_name ("Timer" , 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 (loopCount == 700) { // 7 seconds
  T1.print_conf ();
  loopCount = 0;
}
```

IDE Serial Monitor

```
18:04:52.200 -> Timer [1] Start timer
18:04:53.204 -> Timer [0] Reset timer
18:04:54.205 -> Timer [2] Pause timer
18:04:55.216 -> Timer [1] Start timer
18:04:56.216 -> Timer [2] Pause timer
18:04:57.250 -> Timer [1] Start timer
18:04:58.222 -> Timer [3] Stop timer
```

```
MTD2A_timer:
--------------------
  objectName    : Timer
  processState  : COMPLETE
  phaseText     : [3] Stop timer
  debugPrint    : ENABLE
  globalDebugPr: DISABLE
  errorPrint    : DISABLE
  globalErrorPr: ENABLE
  errorNumber   : 0 OK
  countDownMS   : 3000
  remainTimeMS  : 0
  elapsedTimeMS: 3000
  startTimeMS   : 3012
  stopTimeMS    : 8026
  pauseTimeMS   : 2012
  pauseBeginMS  : 6032
  pauseEndMS    : 7038
```

Sidste målte pauseperiode

Print_conf();

`object_name.print_conf ();`

Maksimal tidspræcision med Arduino ESP32 og kadance DELAY_1MS

```
MTD2A_timer:
--------------------
  objectName    : Timer
  processState  : COMPLETE
  phaseText     : [3] Stop timer
  debugPrint    : ENABLE
  globalDebugPr: DISABLE
  errorPrint    : DISABLE
  globalErrorPr: ENABLE
  errorNumber   : 0 OK
  countDownMS   : 8000
  remainTimeMS  : 6000
  elapsedTimeMS: 2000
  startTimeMS   : 4576
  stopTimeMS    : 9576
  pauseTimeMS   : 3000
  pauseBeginMS  : 6576
  pauseEndMS    : 9576
```

Sidste målte pauseperiode

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

Set og get funktionsoversigt

Set functions	Comment
set_countDownMS ({0 - 4294967295});	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_processtState (); 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	Last START_TIMER
get_stopTimeMS (); return uint32_t milliseconds	STOP_TIMER or remainTimeMS is zero
get_pauseTimeMS (); return uint32_t milliseconds	Acuumulated pause time (sum of all pause periods). Zero if no pause initiated
get_remainTimeMS (); return uint32_t milliseconds	Remining time since first start
get_elapsedTimeMS (); return uint32_t milliseconds	elapsed time since first start
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 stopTimeMS_1 > stopTimeMS_2
object_name_1 << object_name_2	bool stopTimeMS_1 < stopTimeMS_2