# MTD2A_binary_input

**MTD2A: M**odel **T**rain **D**etection **A**nd **A**ction – arduino library https://github.com/MTD2A/MTD2A
Jørgen Bo Madsen / V1.2 / 31-05-2025

MTD2A_binary_input er en brugervenlig avanceret og funktionel C++ klasse til tidsstyret håndtering af input fra sensorer, knapper og meget 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_input processen består af 3 funktioner:

1. `MTD2A_binary_input object_name (”object_name” , delayTimeMS, { LAST_TRIGGER | FIRST_TRIGGER }, {TIME_DELAY | MONO_STABLE}, pinBlockTimeMS);`

2. `object_name.initialize ( pinNumber, {NORMAL | INVERTED}, { INPUT_PULLUP | INPUT } );`
   Kaldes i `void setup ();` og efter `Serial.begin (”Hastighed”);`

3. `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”);
MTD2A_binary_input object_name (”object_name” , delayTimeMS);
MTD2A_binary_input object_name (”object_name” , delayTimeMS, triggerMode);
MTD2A_binary_input object_name (”object_name” , delayTimeMS, triggerMode, timerMode);
MTD2A_binary_input object_name (”object_name” , delayTimeMS, triggerMode, timerMode, pinBlockTimeMS);
Default:                                (”Object name” , 0                , LAST_TRIGGER,  TIMER_DELAY,  0);
```

### Eksempel

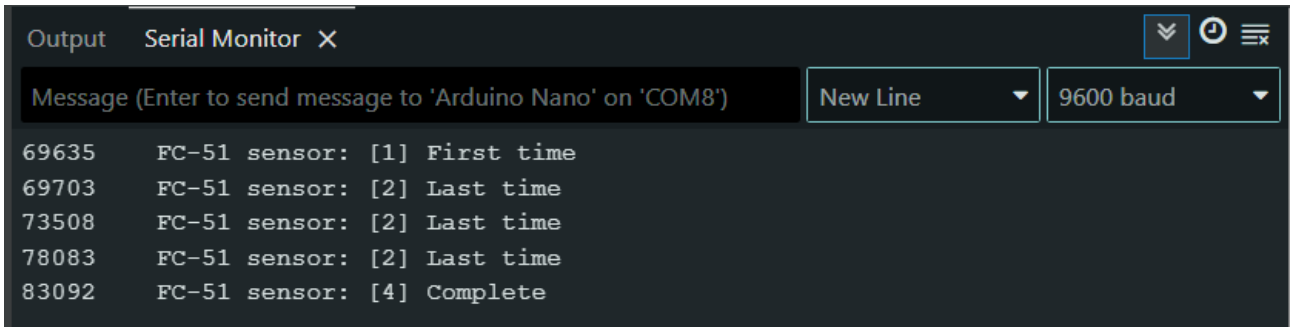
```
// Read sensor and write phase state information to Arduino IDE serial monitor
// https://github.com/MTD2A/FC-51
#include <MTD2A.h>
using namespace MTD2A_const;

MTD2A_binary_input FC_51_sensor ("FC-51 sensor", 5000);
// "FC-51 left" = Sensor (object) name, which is displayed together with status messages
// 5000 = Time delay in milliseconds (5 seconds)
// default: LAST_TRIGGER = Start calculating time from last impulse (LOW->HIGH)
// default: TIME_DELAY = Use time delay (timer function)

void setup () {
  Serial.begin (9600);  // Required and first if status messages are to be displayed
  byte FC51_SENSOR_PIN = 2;
  FC_51_sensor.initialize (FC51_SENSOR_PIN); // Arduino board pin 2 input.
  FC_51_sensor.set_debprougPrint ();  // Display status messages
}

void loop () {
  MTD2A_loop_execute ();
}
```

Eksempel på udskrift til IDE Serial Monitor:

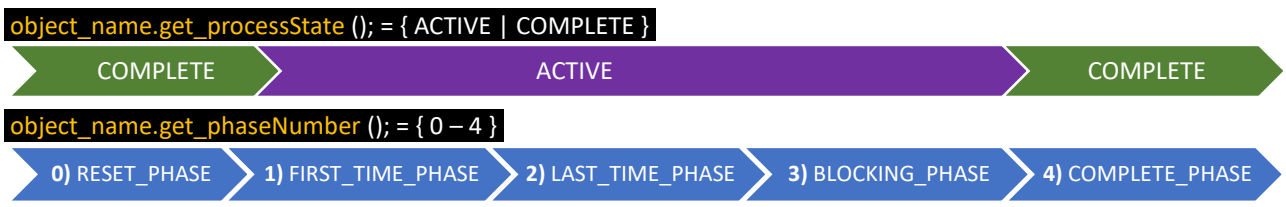



*Flere eksempler og youtybe demo video:*

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

### *Proces faser*

Afhægig af den aktuelle konfiguration gennemføres processen i mellem 3 og 5 faser.




0. Den initelle fase når programmet starter samt når funktion `reset ();` kaldes.
1. Første gang at der sker en ændring i input (sensor eller programmet selv).
2. Sidste gang at der skete en ændring i input. Kan forekommer flere gange.
3. Blokering af input fra sensor eller programmet selv i en tidsbestemt periode.
4. Afventer ny input (tilstandsændring) fra sensor eller programmet selv.

Globale nummerkonstanter:
RESET_PHASE, FIRST_TIME_PHASE, LAST_TIME_PHASE, BLOCKING_PHASE & COMPLETE_PHASE

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

### *Proces status*

Ved overgang til FIRST_TIME_PHASE eller LAST_TIME_PHASE skifter ProcessState til ACTIVE.
Ved overgang til COMPLETE_PHASE skifter processState til COMPLETE.

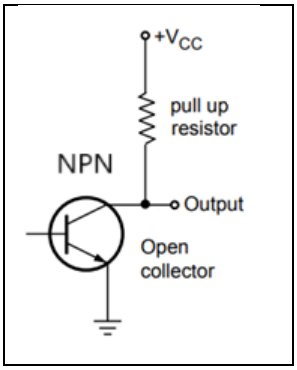
### *Timing*

Se dokumentet MTD2A.PDF og asnittet "Kadance og synkronisering" samt "Eksekveringshastighed".

## Input detektering og aktivering

MTD2A_binary_input læser input fra
1) Digital benforbindelse på boardet
2) Programmet selv.

Input fra digital benforbindelse på Arduino board er som standard konfigureret med INPUT_PULLUP. Det betyder at der er tilsluttet en modstand på typisk 10 K ohm mellem input benforbindelsen og + (plus) = HIGH. Aktivering sker ved at forbinde benforbindelsen til – (minus) = LOW.
Se mere her: INPUT | INPUT_PULLUP | OUTPUT | Arduino Documentation




Der kan benyttes alle former for bryde og slutte kontakter, momentan og skifte kontakter, relæer og alle former for kredsløb med Open Collector transistor NPN - binært og analogt. Ved analogt kredsløb skifter status efter spændingsniveauerne som beskrevet her: HIGH | LOW | Arduino Documentation

Hvis input kredsløbet også benytter egen pullup modstand, vil det som udgangspunkt virke som det skal.
I modsat fald kan INPUT_PULLUP vælges fra ved at angive INPUT i funktionen:
object_name.initialize ( pinNumber, {NORMAL | INVERTED}, {INPUT_PULLUP | INPUT } );.

| Der er to mulige input til funktionen:<br>1. Input fra digital benforbindelse på Arduino board pinState => {HIGH \| LOW} pin read only.<br>2. Input fra selve programmet inputState = {HIGH \| LOW} write & read. | pinState | inputState | CurrState |
|---|---|---|---|
| | HIGH | HIGH | HIGH |
| | HIGH | LOW | LOW |
| | LOW | HIGH | LOW |
| | LOW | LOW | LOW |

Input aflæses fra det digitale benforbindelsesnummer, der er specificeret i object_name.initialize ( pinNumber );
Kaldes funktionen ikke, bliver benforbindelsen ikke aflæst pinRead = disable og pinNumber = 255.

Hvis benforbindelsenummer er initialiseret korrekt med ovenstående funktion, er det muligt løbende at styre om benforbinelsen skal aflæses eller ej med funktionen:
object_name.set_pinRead ( {ENABLE | DISABLE} );

Input kan også komme fra programmet selv:
object_name.set_inputState ( {HIGH | LOW}, {PULSE | FIXED} );

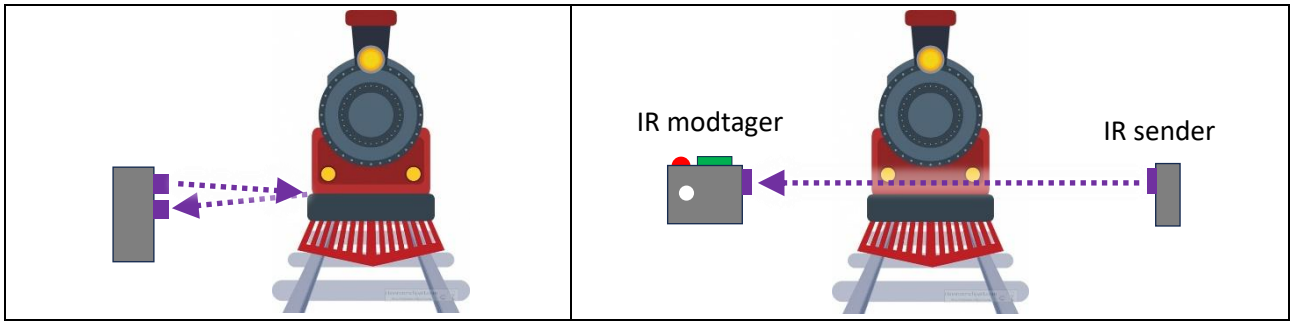
Pulse angiver en enkelt impuls (kort monostabilt) og fixed virker permanent og indtil Pulse angives.

## Pin Input mode

Der er to måder af aflæse input på:

1. NORMAL Trigger sker ved HIGH -> LOW. Output spejler input. Fx reflektionssensor.
2. INVERTED Trigger sker ved LOW -> HIGH. Output følger input. Fx strålebrydning sensor.

Default normal `object_name.initialize (pinNumber);` eller `object_name.initialize (pinNumber, INVERTED);`

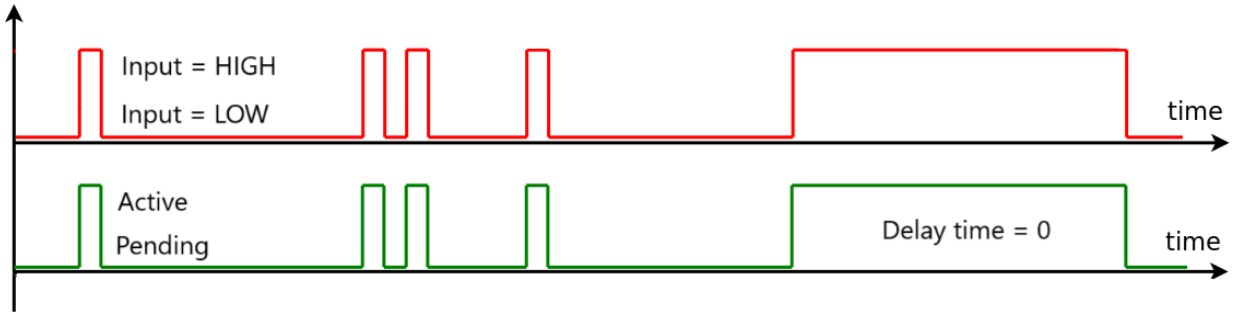



I de følgende eksempler tages der udgangspunkt i FC-51 binær strålebrydning sensor, hvor sender er placeret på den ene side af toget, og modtager erplaceret på den anden side.

## Simpel binær funktion

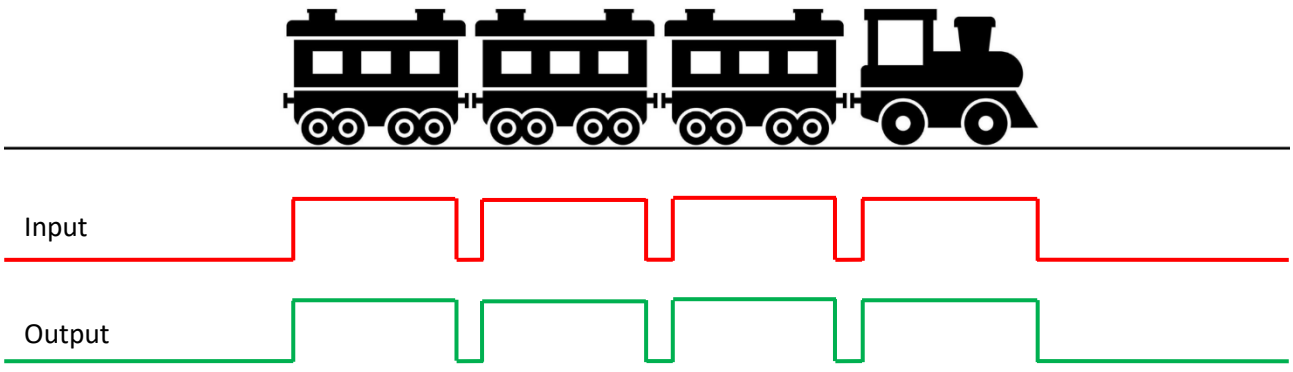
```
MTD2A_binary_input FC_51_sensor (”FC_51_sensor”);
```

Når input går fra LOW til HIGH gør output præcis det samme, og omvendt.



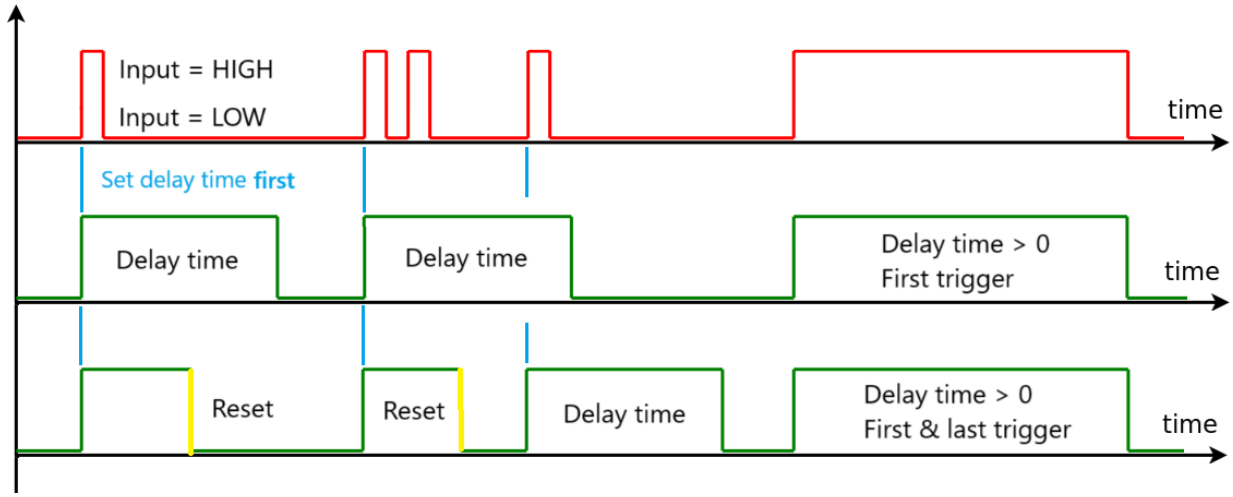

Sensordetektering af et togsæt i bevægelse vil uundgåeligt medføre et antal impulser grundet variationer i opbygning af togvognene og lokomotiv, samt ”huller” ved sammenkoblinger med mere. Disse variationer kan medføre uforudsete genaktivering af funktioner og fejl i den efterfølgende logik proces.

*Eksempel på kørende tog*



## Time delay – first trigger

Når input går fra LOW til HIGH fastholdes output HIGH ind til at den definerede tidsperiode ophører. Er input HIGH ved tidsperiodens ophør, forbliver output HIGH, ind til at input går fra HIGH til LOW.




```
object_name.reset ();
```

Nulstiller alle styrings og process variable og gør klar til ny start. Alle funktionskonfigurerede variable og standard værdier bibeholdes. Procesfasen skifter til RESET_PHASE

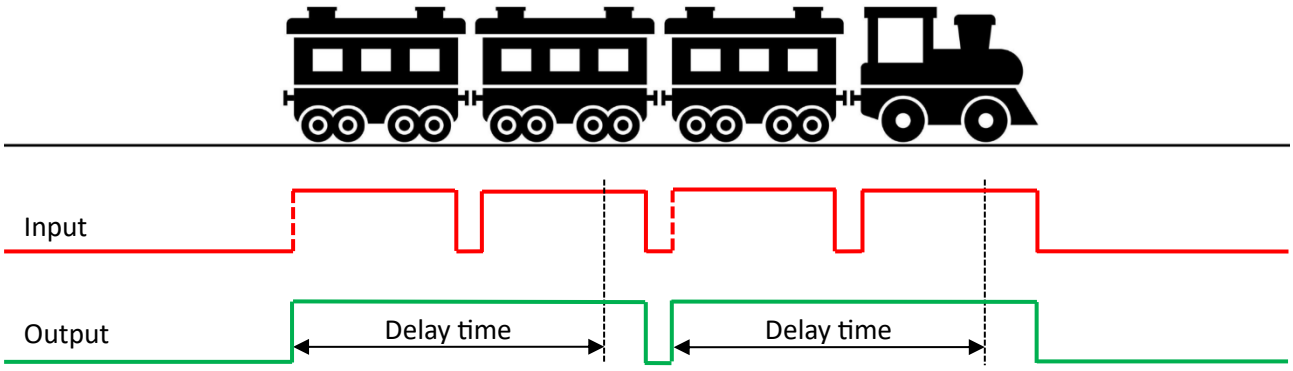
```
object_name.set_stopDelayTimer ( );
```
Stopper øjejeblikkeligt forsinkelsespersioden og går videre tilnæste fase.

### *Eksempel på kørende tog*

Hvis uhensigtsmæssig genaktivering skal undgås, skal delayTimeMS være lang for at sikre, at det også fungerer ved langsomt kørende tog. I eksemplet herunder er delayTimeMS for kort, hvilket meføre to aktiveringer i stedt for en.

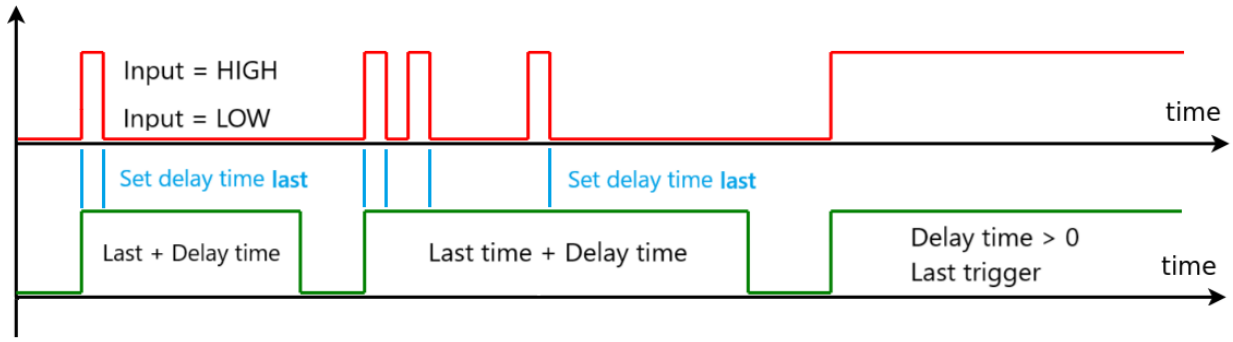
delayTimeMS = 5.000 millisekunder og triggerMode = **FIRST**_TRIGGER.
5 sekund forsinkelse målt fra første detektering af tog.

```
MTD2A_binary_input FC_51_sensor ("FC_51_sensor", 5000, FIRST_TRIGGER);
```



## Time delay – last trigger

Når input går fra LOW til HIGH fastholdes output HIGH ind til at den definerede tidsperiode ophører. Hver gang input går fra LOW til HIGH forskydes starten for tidsperioden til det nye tidspunkt. Er input HIGH ved tidsperiodens ophør, forbliver output HIGH, ind til at input går fra HIGH til LOW.

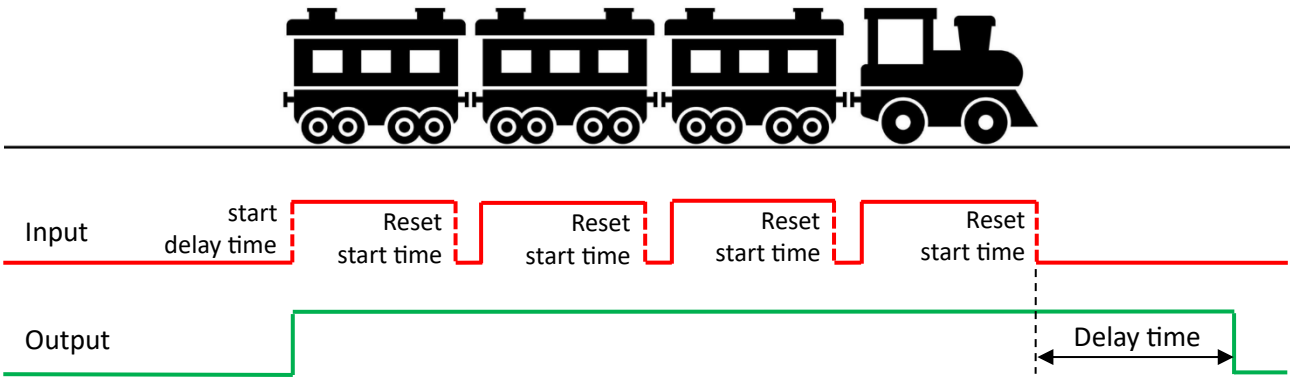



### *Eksempel på kørende tog*

Denne metode er bedst egnet til at detektere hurtigt- og langsomt kørende tog uden uhensigtsmæssige genaktiveringer.

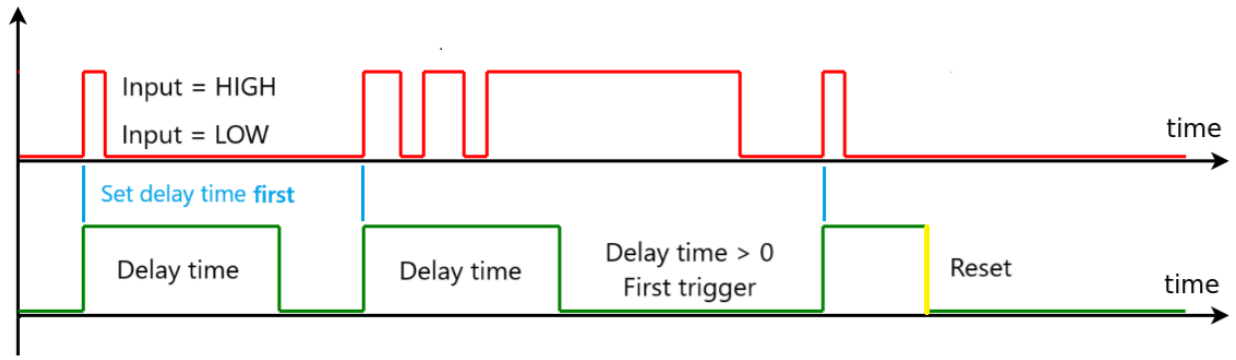
delayTimeMS = 5.000 millisekunder og triggerMode = **LAST**_TRIGGER.
5 sekunder forsinkelse målt fra sidste detektering af tog (HIGH til LOW).

```
MTD2A_binary_input FC_51_sensor ("FC_51_sensor", 5000, LAST_TRIGGER);
```



## Monostable – first trigger

Monostabilt fastholder altid den definerede tidsperiode, uanset om input skifter mellem HIGH og LOW i tidsperioden, og forbliver input enten HIGH eller LOW, ændre det ikke tidsperioden.

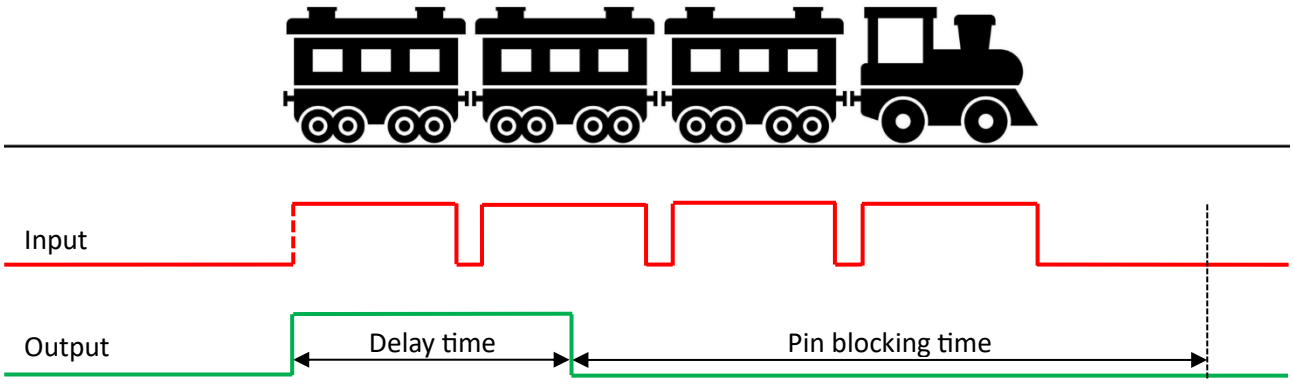



object_name.set_stopBlockTimer ( ); Stopper øjeblikkeligt forsinkelsespersioden og går videre tilnæste fase.

### *Eksempel på kørende tog*

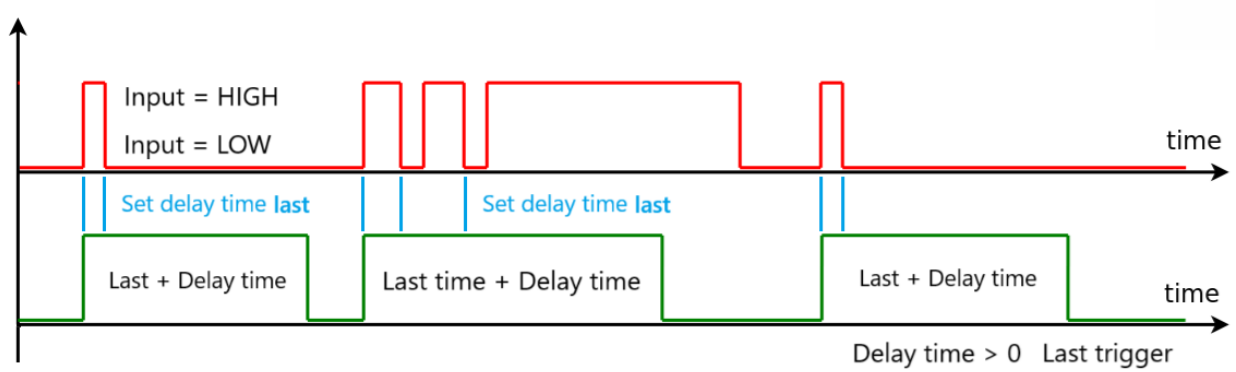
delayTimeMS = 5.000 millisekunder, triggerMode = **FIRST**_TRIGGER, timerMode = MONO_STABLE, pinBlockMS = 12.000 millisekunder (tidsperiode hvor input fra benforbindelsen blokeres fra delayTimeMS afslutning og frem til monostabil afslutning).
5 sekunder forsinkelse målt fra første detektering af tog (LOW til HIGH).

```
MTD2A_binary_input FC_51_sensor ("FC_51_sensor", 5000, FIRST_TRIGGER, MONO_STABLE, 12000);
```



## Monostable – last trigger

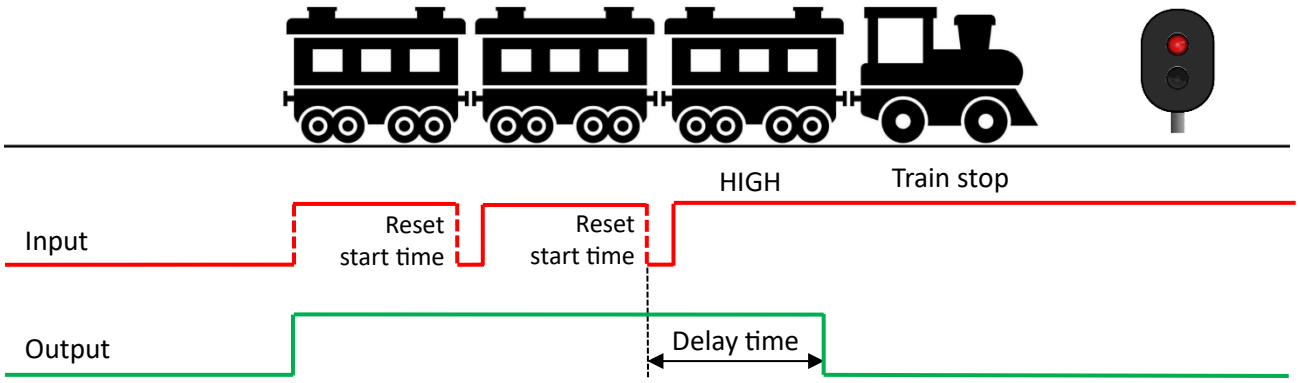
Monostabilt fastholder altid den definerede tidsperiode, men skifter input i mellem HIGH og LOW i tidsperioden, forlænges tidsperioden hver gang. Efter den samlede tidsperiode skifter output til LOW, uanset om input enten er HIGH eller LOW.



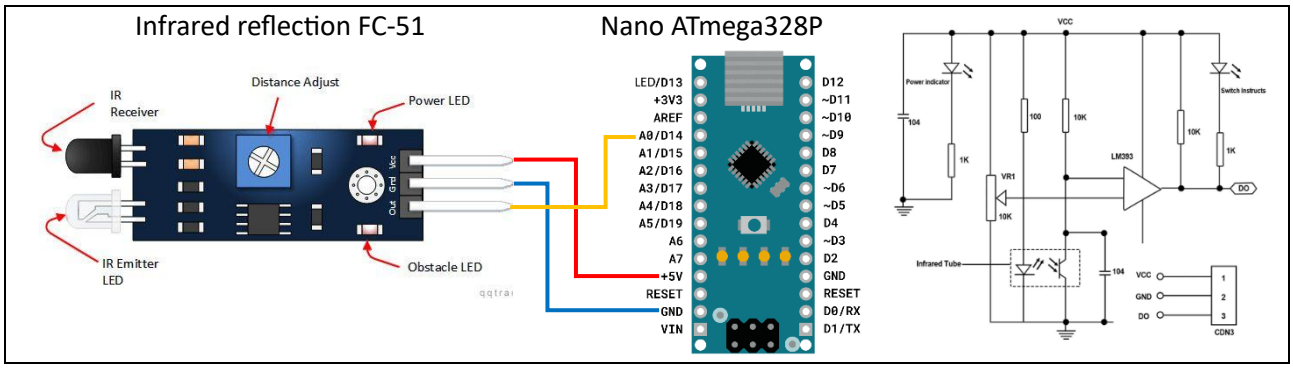

*Eksempel på kørende tog der standser over sensor*

delayTimeMS = 3.000 millisekunder, triggerMode = LAST_TRIGGER, timerMode = MONO_STABLE.
3 sekunder forsinkelse målt fra første detektering af tog (LOW til HIGH).

```
MTD2A_binary_input FC_51_sensor ("FC_51_sensor", 3000, LAST_TRIGGER, MONO_STABLE);
```




## Eksempler på configuration




1. MTD2A_binary_input FC_51_sensor ("FC_51_sensor", 5000);
2. FC_51_sensor.initialize (A0);
3. FC_51_sensor.set_debugPrint ();
4. MTD2A_loop_execute ();

# Øvrige funktioner

| Set functions | Comment |
|---|---|
| set_pinRead ( {ENABLE \| DISABLE} ); | Enable or disable pin reading |
| set_pinInput ( {NORMAL \| INVERTED} ); | Configure pin trigger input |
| set_InputState ( {HIGH \| LOW}, {PULSE \| FIXED} ); | Activate input state and set input mode |
| set_stopDelayTimer (); | Stop first og last timer process immediately. |
| set_stopBlockTimer (); | Stop blocking timer process immediately. |
| set_debugPrint ( {ENABLE \| DISABLE} ); | Activate print phase number and text |

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

| Get functions | Comment |
|---|---|
| get_processtState (); return bool {ACTIVE \| COMPLETE} | Procedure process state |
| get_pinState (); return bool {HIGH \| LOW} | Current pin input state |
| get_phaseChange (); return bool {true \| false} | Momentarily phase change (one loop time) |
| get_phaseNumber (); return uint8_t {0- 4} | Reset = 0, firstTime =1, lastTime = 2, blocking = 3, pending = 4 |
| get_firstTimeMS (); return uint32_t milliseconds | First time trigger time (falling edge) |
| get_lastTimeMS (); return uint32_t milliseconds | Last time trigger time (rising edge) |
| get_endTimeMS (); return uint32_t milliseconds | End time (total delay time) |
| get_inputGoLow (); return bool {true \| false} | Falling edge detected |
| get_inputGoHigh (); return bool {true \| false} | Rising edge detected |
| 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 lastTimeMS_1 > lastTimeMS_2 |
| object_name_1 << object_name_2 | bool lastTimeMS_1 < lastTimeMS_2 |

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

```
MTD2A_binary_input:
-------------------
  objectName    : Left
  processState  : COMPLETE
  phaseText     : [2] Last time
  debugPrint    : ENABLE
  errorNumber   : 0 OK
  triggerMode   : LAST_TRIGGER
  timerMode     : TIME_DELAY
  pinNumber     : 2
  pinType       : INPUT_PULLUP
  pinRead       : ENABLE
  pinInput      : NORMAL
  inputMode     : PULSE
  delayTimeMS   : 10000
  firstTimeMS   : 4517
  lastTimeMS    : 7919
  endTimeMS     : 0
  blockTimeMS   : 0
  pinState      : HIGH
  inputState    : HIGH
```