MTD2A_binary_input

MTD2A: Model **Train Detection And Action – arduino library** https://github.com/MTD2A/MTD2A
Jørgen Bo Madsen / V1.6 / 19-09-2025

MTD2A_binary_input is an easy-to-use, advanced and functional C++ class for time-controlled handling of inputs from sensors, buttons and more, as well as the program itself. MTD2A supports parallel processing and asynchronous execution.

The class is among a number of logical building blocks that solve different functions.
Common to all building blocks are:
• They support a wide range of input sensors and output devices
• Are simple to use to build complex solutions with few commands
• They operate non-blocking, process-oriented and state-driven
• Offers extensive control and troubleshooting information
• Thoroughly documented with examples

Table of contents

Feature Description

MTD2A_binary_input process consists of 3 functions:

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 });`
   Called in `void setup ();` and after `Serial.begin ("Velocity");`

3. `MTD2A_loop_execute ();` Called as the last instruction in `void loop ();`

All functions use default values and can therefore be called with none and up to the maximum number of parameters. However, the parameter must be specified in ascending order. See example below:

```
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);
```

Example

```
// 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 sensor" = 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
  while (!Serial) { delay(10); } // ESP32 Serial Monitor ready delay

  byte FC51_SENSOR_PIN = 2;
  FC_51_sensor.initialize (FC51_SENSOR_PIN); // Arduino board pin 2 input.
  FC_51_sensor.set_debugPrint (); // Display status messages
}

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

Sample printout for IDE Serial Monitor:

```
Output   Serial Monitor ×
Message (Enter to send message to 'Arduino Nano' on 'COM8')   New Line   9600 baud
69635    FC-51 sensor: [1] First time
69703    FC-51 sensor: [2] Last time
73508    FC-51 sensor: [2] Last time
78083    FC-51 sensor: [2] Last time
83092    FC-51 sensor: [4] Complete
```

More examples and youtube demo video:

https://github.com/MTD2A/MTD2A/tree/main/examples

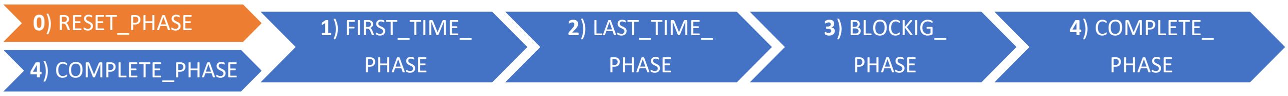
DEMO video: https://youtu.be/RDFgEbhYUzE

Process phases

Depending on the current configuration, the process is carried out in between 3 and 5 stages.

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

object_name.get_phaseNumber (); = { 0 – 4 }



0. 0) when the function reset (); is called. 4) The initial phase when the program starts.
1. The first time that there was a change in the input (sensor or the program itself).
2. Last time there was a change in input. May occur several times.
3. Blocking input from the sensor or the program itself for a timed period.
4. Awaiting new input (change of state) from the sensor or the program itself.

Global number constants:

RESET_PHASE, FIRST_TIME_PHASE, LAST_TIME_PHASE, BLOCKING_PHASE & COMPLETE_PHASE

The instantaneous phase shift can be identified by the function: object_name.get_phaseChange (); = { true | false }

Process status

When transitioning to FIRST_TIME_PHASE or LAST_TIME_PHASE, ProcessState switches to ACTIVE.
When transitioning to COMPLETE_PHASE, the processState switches to COMPLETE.

Timing

The time periods are set by default when the object is instantiated (activated).
It is possible to define new time periods for both timers:

object_name.set_delayTimeMS ({0 - 4294967295});

object_name.set_pinBlockMS ({0 - 4294967295});

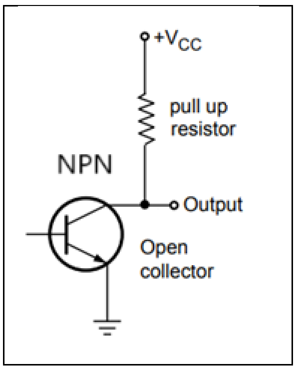
See the document MTD2A.PDF and the section "Cadence", "Synchronization" as well as "Execution speed".

Input detection and activation

MTD2A_binary_input read input from

1) Digital Pin connection on the Arduino Board
2) The program itself.

Input from digital pin connection on the Arduino board is configured with INPUT_PULLUP by default. This means that a resistor of typically 10 K ohms is connected between the input pin connection and + (plus) = HIGH. Activation is done by connecting the pin connection to – (minus) = LOW.
See more here: [INPUT | INPUT_PULLUP | OUTPUT | Arduino Documentation]()



All types of switches, relays and all kinds of circuits can be used with the Open Collector transistor NPN - binary and analogue. In analog circuitry, the status changes according to the voltage levels as described here: [HIGH | LOW | Arduino Documentation]()

If the input circuit also uses its own pullup resistance, it will generally work as it should. Otherwise, INPUT_PULLUP can be deselected by entering INPUT in the function:
`object_name.initialize ( pinNumber, {NORMAL | INVERTED}, {INPUT_PULLUP | INPUT } );.`

There are two possible inputs to the function:

1. Input from digital leg connection on Arduino board
   pinState => {HIGH | LOW} pin read only.
2. Input from the program itself
   inputState = {HIGH | LOW} write & read.

pinState	inputState	CurrState
HIGH	HIGH	HIGH
HIGH	LOW	LOW
LOW	HIGH	LOW
LOW	LOW	LOW

Input is read from the digital pin connection number specified in `object_name.initialize ( pinNumber );`
If the function is not called, the pin connection will not be read `pinReadToggl = disable` and `pinNumber` = 255.

If the pin connection is initialized correctly with the above function, it is possible to continuously control whether the pin connection should be read or not with the function:
`object_name.set_pinReadToggl ( {ENABLE | DISABLE} );`

Input can also come from the program itself:
`object_name.set_inputState ( {HIGH | LOW}, {PULSE | FIXED} );`

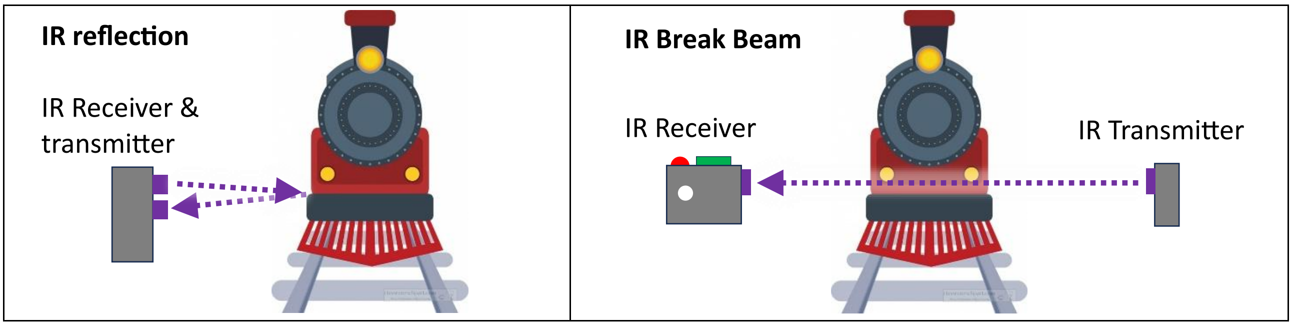
Pulse indicates a single pulse (short monostable) and fixed works permanently and until Pulse is specified.

Pin Input mode

There are two ways to read input:

1. NORMAL Trigger occurs at HIGH -> LOW. The output mirrors the input. For example, a reflection sensor.
2. INVERTED Trigger occurs at LOW -> HIGH. Output follows input. E.g. break beam sensor.

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

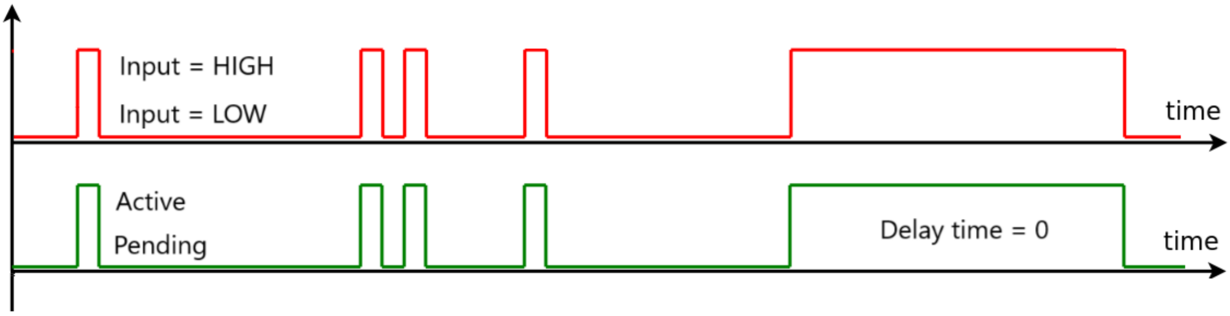


The following examples are based on the FC-51 binary break beam sensor, where the transmitter is placed on one side of the train, and the receiver is placed on the other side.

Simple binary function

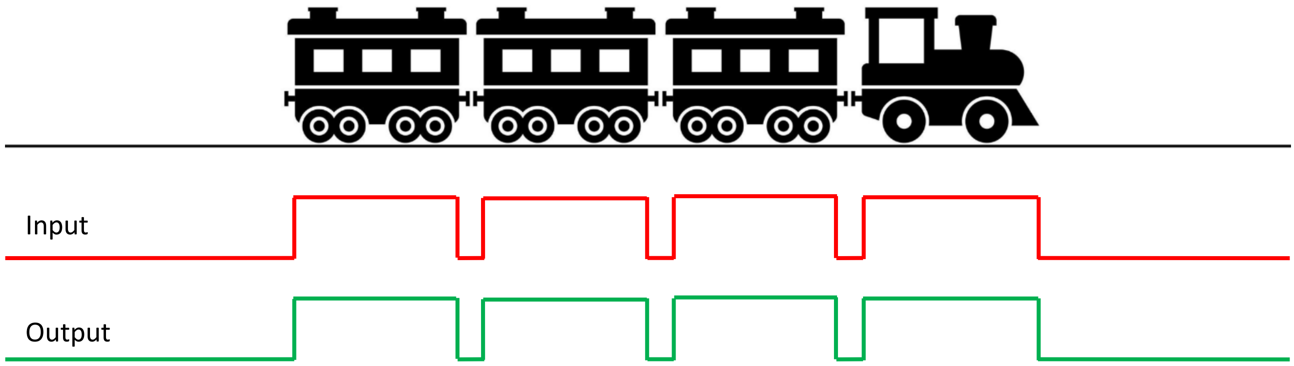
```
MTD2A_binary_input FC_51_sensor ("FC_51_sensor");
```

When the input goes from LOW to HIGH, the output does exactly the same, and vice versa.



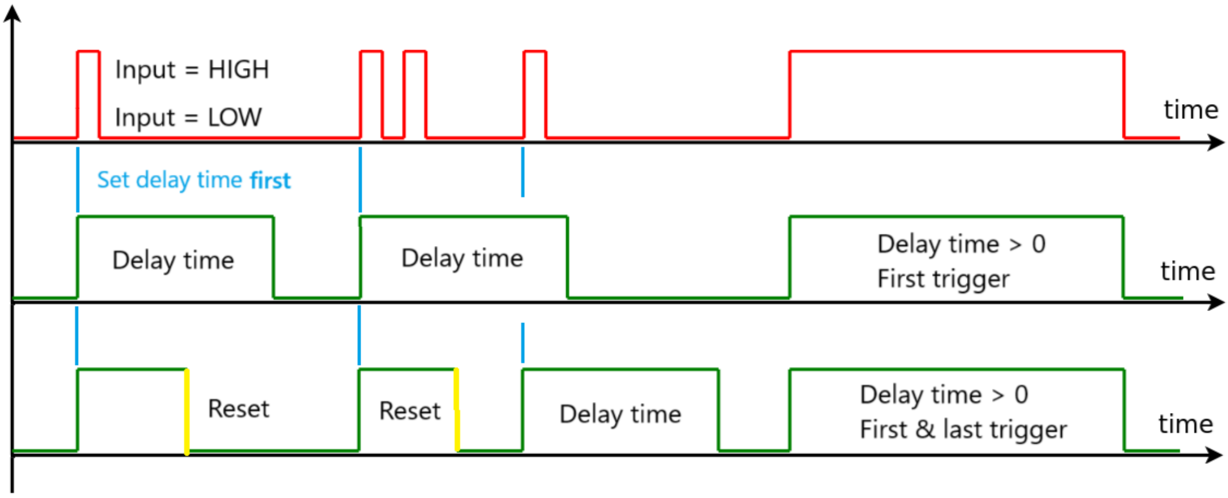
Sensor detection of a moving train set will inevitably cause a number of pulses due to variations in the construction of the train cars and locomotive, as well as "holes" at connections and more. These variations can cause unforeseen reactivation of functions and errors in the subsequent logic process.

Example of moving trains



Time delay – first trigger

When the input goes from LOW to HIGH, the output HIGH is maintained until the defined time period ends. If the input is HIGH at the end of the time period, the output remains HIGH until the input goes from HIGH to LOW.



object_name.reset ();

Resets all control and process variables and prepares for a fresh start. All functionally configured variables and default values are retained. The process phase switches to RESET_PHASE

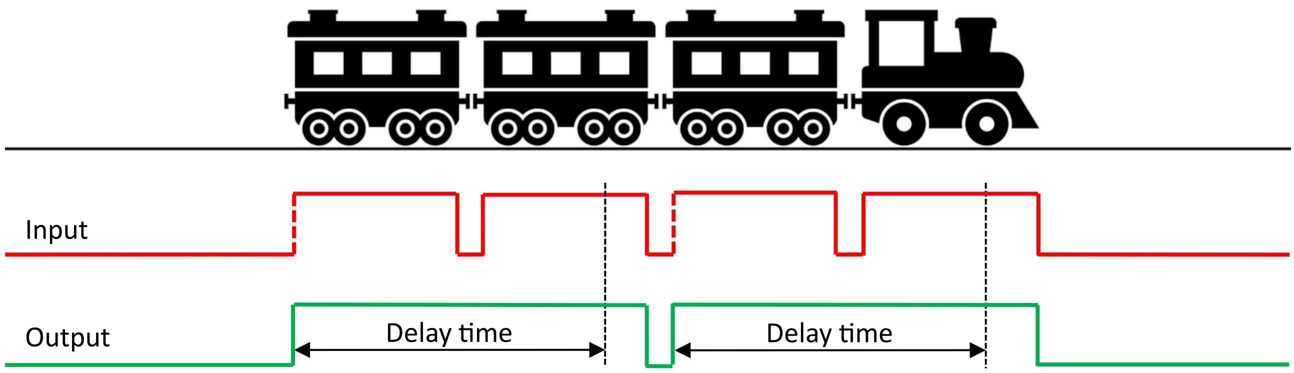
object_name.set_stopDelayTimer ( ); Instantly stops the delay period and moves on to the next stage.

Example of moving trains

To avoid inappropriate reactivation, delayTimeMS must be long to ensure that it also works on slow-moving trains. In the example below, delayTimeMS is too short, which results in two activations instead of one.

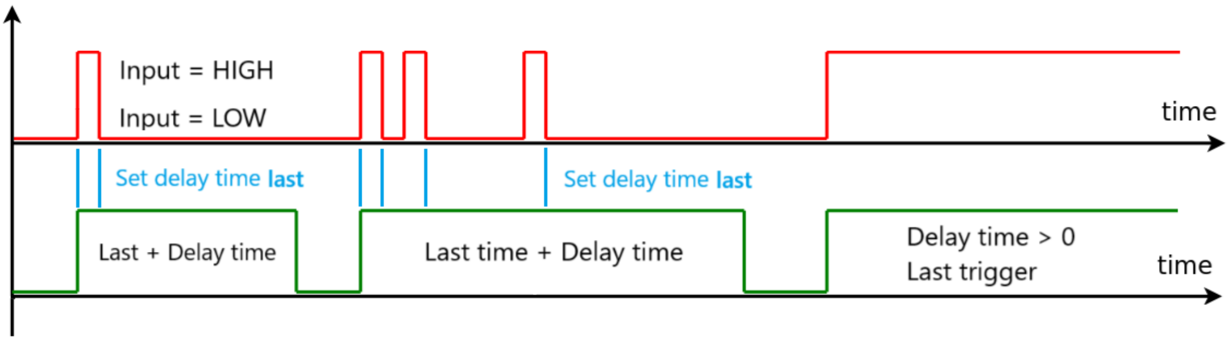
delayTimeMS = 5,000 milliseconds and triggerMode = **FIRST**_TRIGGER.
5 second delay measured from first detection of trains.

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



Time delay – last trigger

When the input goes from LOW to HIGH, the output HIGH is maintained until the defined time period ends. Each time the input goes from HIGH to LOW, the start of the time period is shifted to the new time. If the input is HIGH at the end of the time period, the output remains HIGH until the input goes from HIGH to LOW.

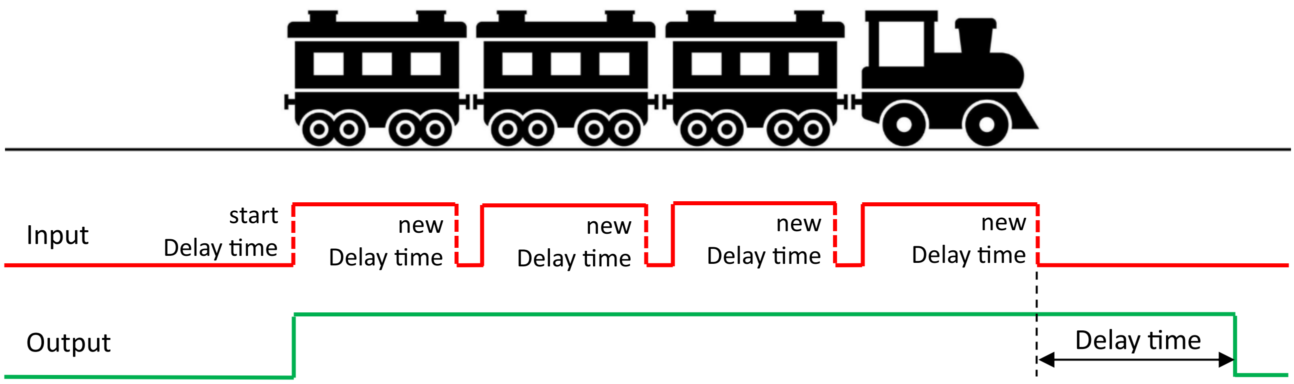


Example of moving trains

This method is best suited for detecting fast and slow-moving trains without inappropriate reactivations.

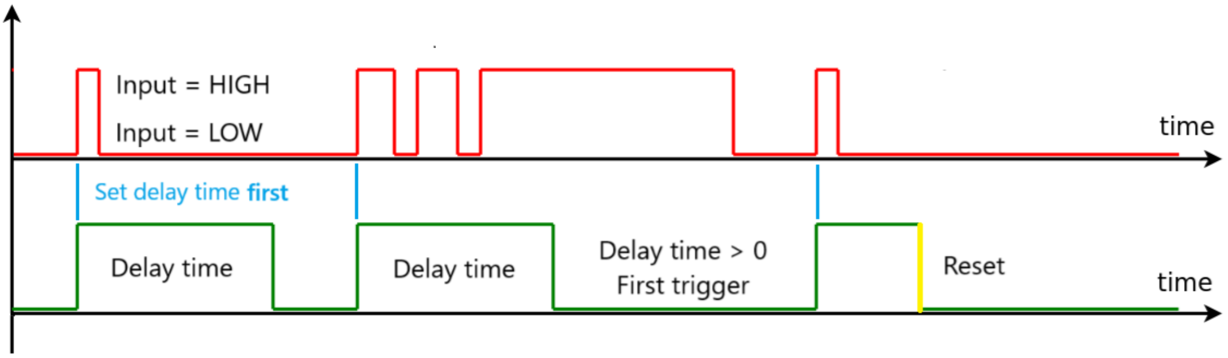
delayTimeMS = 5,000 milliseconds and triggerMode = **LAST**_TRIGGER.
5 seconds delay measured from the last detection of trains (HIGH to LOW).

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



Monostable – first trigger

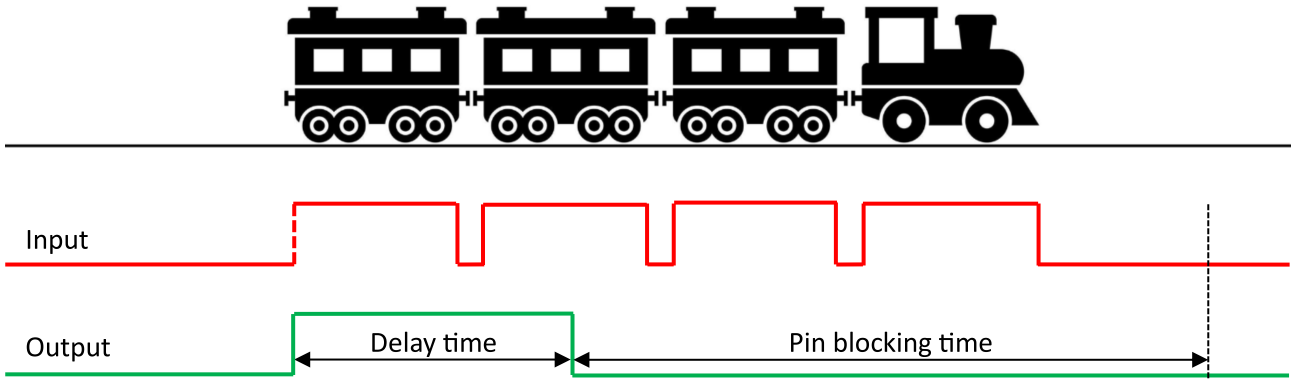
Monostable always maintains the defined time period, regardless of whether the input changes between HIGH and LOW during the time period, and if the input remains either HIGH or LOW, it does not change the time period.



Example of moving trains

delayTimeMS = 5,000 milliseconds, triggerMode = **FIRST**_TRIGGER, timerMode = MONO_STABLE, pinBlockMS = 12,000 milliseconds (time period where input from the pin connection is blocked from delayTimeMS termination until monostable termination).
5 seconds delay measured from first detection of trains (LOW to HIGH).

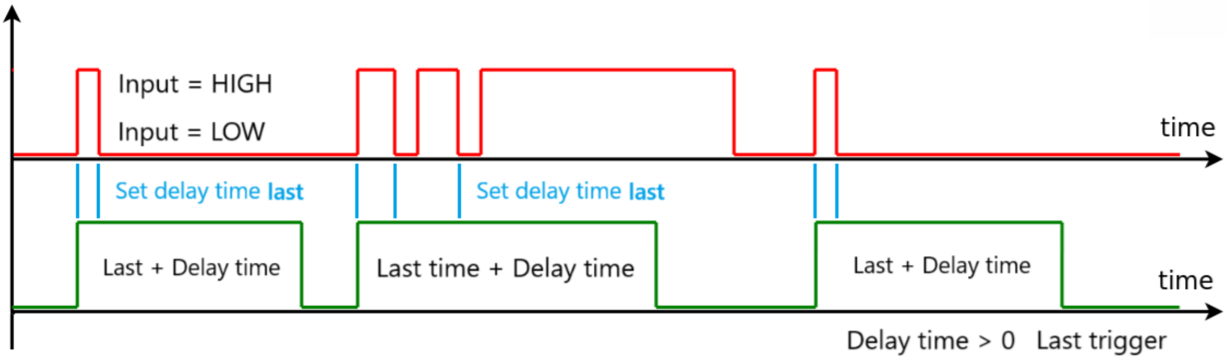
```
MTD2A_binary_input FC_51_sensor ("FC_51_sensor", 5000, FIRST_TRIGGER, MONO_STABLE, 12000);
```



`object_name.set_stopBlockTimer ( );` Instantly stops the delay period and moves on to the next stage.

Monostable – last trigger

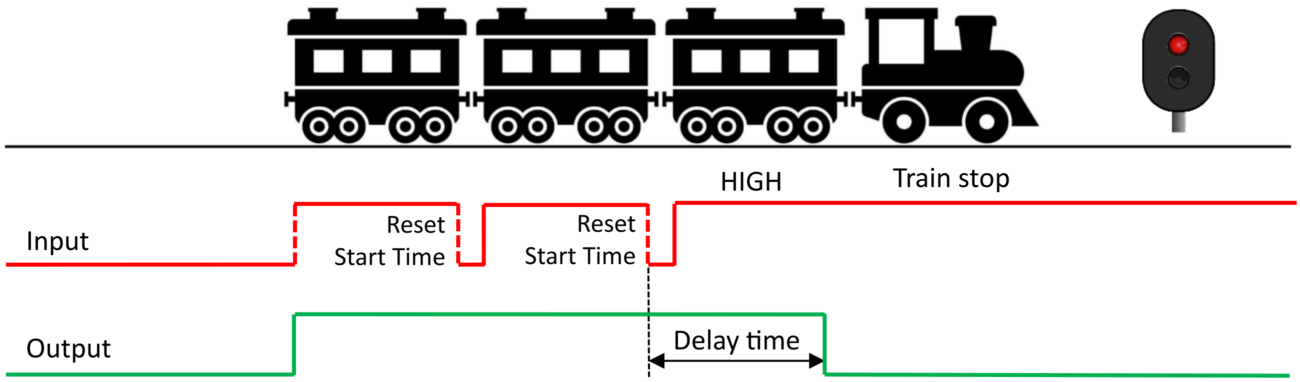
Monostable always maintains the defined time period, but switching inputs between HIGH and LOW during the time period, the time period is extended each time. After the total time period, the output will change to LOW regardless of whether the input is either HIGH or LOW.



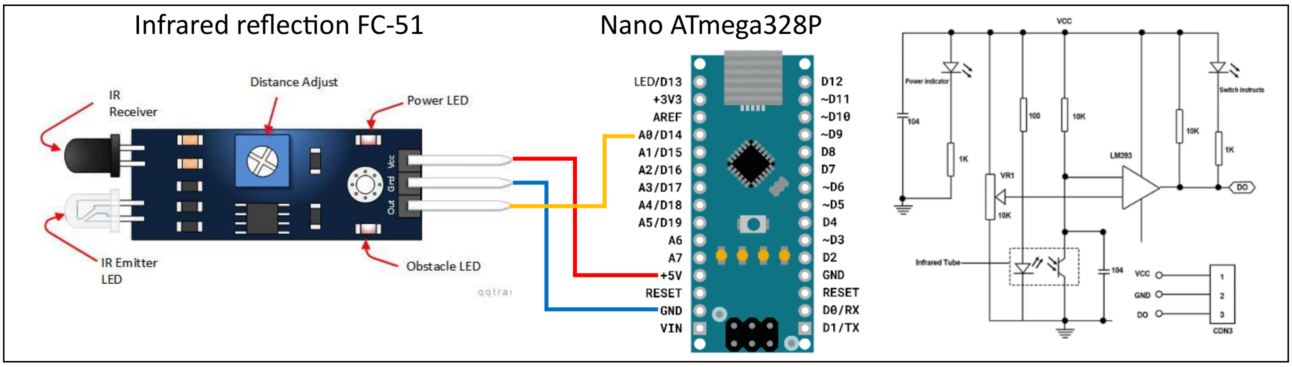
Example of moving train stopping over sensor

delayTimeMS = 3,000 milliseconds, triggerMode = LAST_TRIGGER, timerMode = MONO_STABLE.
3 seconds delay measured from first detection of trains (LOW to HIGH).

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



Examples of 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 ();

Get and set functions

Set functions	Comment
set_pinReadToggl ({ENABLE | DISABLE});	Enable or disable pin reading
set_pinReadMode ({NORMAL | INVERTED});	Configure pin trigger input
set_InputState ({HIGH | LOW}, {PULSE | FIXED});	Activate input state and set input mode
set_delayTimeMS ({0- 4294967295});	Set new delay time after instantiation
set_pinBlockMS ({0- 4294967295});	Set new blocking time after instantiation
set_stopDelayTimer ();	Stop first and last timer process immediately.
set_stopBlockTimer ();	Stop blocking timer process immediately.
set_debugPrint ({ENABLE | DISABLE});	Enable print phase number and text
set_errorPrint ({ENABLE | DISABLE});	Enable error messages

Get functions	Comment
get_processtState (); return bool {ACTIVE | COMPLETE}	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

print_conf();

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

```
MTD2A_binary_input:
-------------------
  objectName    : Left
  processState : ACTIVE
  phaseText     : [2] Last time
  debugPrint    : ENABLE
  globalDebugPr: DISABLE
  errorPrint    : DISABLE
  GlobalErrorPr: ENABLE
  errorNumber   : 0 OK
  triggerMode   : LAST_TRIGGER
  TimerMode     : TIME_DELAY
  pinNumber     : 2
  pinType       : INPUT_PULLUP
  pinReadToggl : ENABLE
  pinReadMode   : NORMAL
  inputMode     : PULSE
  delayTimeMS   : 10000
  firstTimeMS   : 4517
  lastTimeMS    : 7919
  endTimeMS     : 0
  blockTimeMS   : 0
  pinState      : HIGH
  inputState    : HIGH
```