MTD2A_binary_input

Jørgen Bo Madsen / V1.0 / 31-03-2025

MTD2A_binary_input er en avanceret, ikke blokerende, logisk C++ objekt klasse til tidsstyret håndtering af input fra sensorer, knapper og programmet selv, der er let at anvende til avancerede løsninger.

Funktionsbeskrivelse

MTD2A_binary_input funktionen består af tre funktioner:

1. `MTD2A_binary_input object_name`
   `( "Object Name", delayTimeMS, {firstTrigger | lastTrigger}, {timeDelay | monoStable}, pinBlockTimeMS );`
   Klasseobjektet defineres som det første i programmet. **Standard værdier:**
   "Object navn" = "Object Name".
   delayTimeMS = 0 millisekunder (ingen tidsforsinkelse).
   firstTrigger = Første gang der er en HIGH->LOW input impuls.
   timeDelay = Tidsforsinkelse.
   pinBlockTimeMS = 0 millisekunder (ingen afsluttende blokering af benforbindelsen).

2. `object_name.init_setup ( pinNumber, {normal | inverted} );`
   Kaldes i `void setup()` og efter `Serial.begin("Hastighed")`
   Standard værdier:
   pinNumber = Benforbindelse nummer = 255. Dvs. ingen aflæsning af benforbindelsen.
   Normal = Standard input mode.

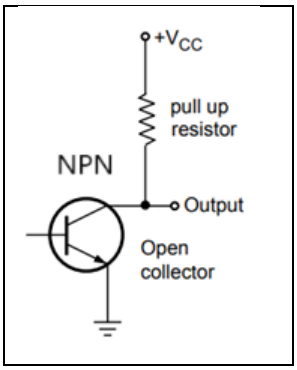
4. `object_name.loop_fast ();`
   Kaldes som det sidste I `void loop()` i den absolut sidste linje tilføjes `delay(10)`

Eksempel

```
MTD2A_binary_input FC_51_left ("FC-51 left", 5000, lastTrigger, timeDelay);
// "FC-51 left" = Sensor (object) navn, som vises sammen med tilstandsbeskeder.
// 5000 = Tidsforsinkelse i millisekunder (5 sekunder).
// lastTrigger = Tidspunkt for start på tidtagning. Her sidste impuls i tidsperioden (LOW->HIGH)
// timeDelay = Benyt tidsforsinkelse (timer funktion)

void setup () {
  Serial.begin (250000);  // Nødvendigt og først, hvis der skal vises tilstandsbeskeder.
  FC_51_left.init_setup (DigitalPin_2);
  // DigitalPin_2 = Arduino board ben nummer.
  FC_51_left.set_debugPrint (on);
  // on = Vis tilstandsbeskeder.
}

void loop () {
  FC_51_left.loop_fast();
  // Opdatere klasseobjektet. Typisk 10 millisekunder
  delay (10);
}
```

Input

Input fra digital benforbindelse på Arduino board er 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

Der er to mulige input til funktionen:

1. Input fra digital benforbindelse på Arduino board
   `pinState` => {HIGH | LOW} pin read only.
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 benforbindelse nummer der er specificeret i `object_name.init_setup` Kaldes funktionen ikke, bliver benforbindelsen ikke aflæst `pinRead = disable`

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 andet input gives.

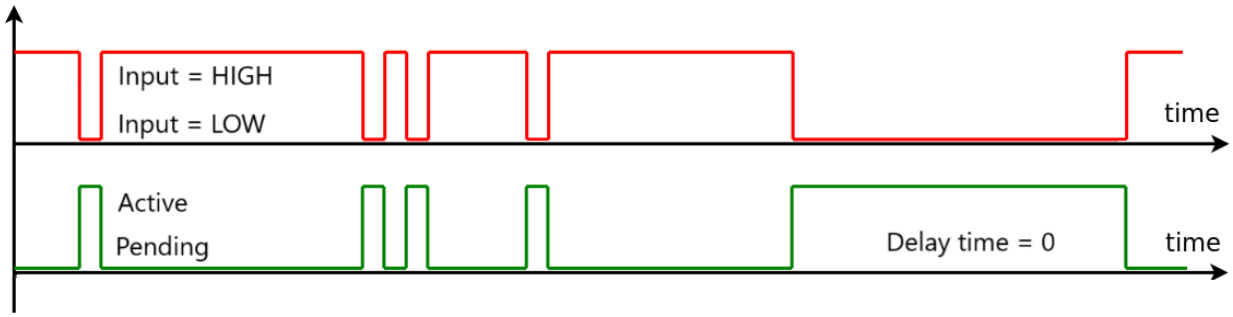
Simpel binær funktion

Spejler input til omvendt output. Når input går fra HIGH til LOW gør output præcis det modsatte.

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

Det er muligt at input og output følges ad ved at vælge parameteren "inverted".

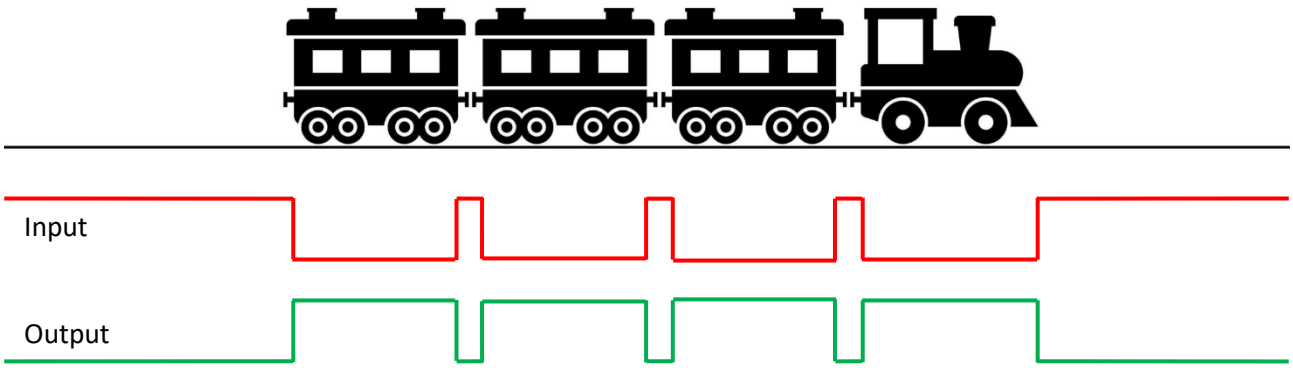
```
FC_51_sensor.init_setup (DigitalPin_2, inverted);
```



I de følgende eksempler tages der udgangspunkt i FC-51 binær refleksion sensor, der er placeret lodret under skinnerne. (kun den ene som vist på billedet).

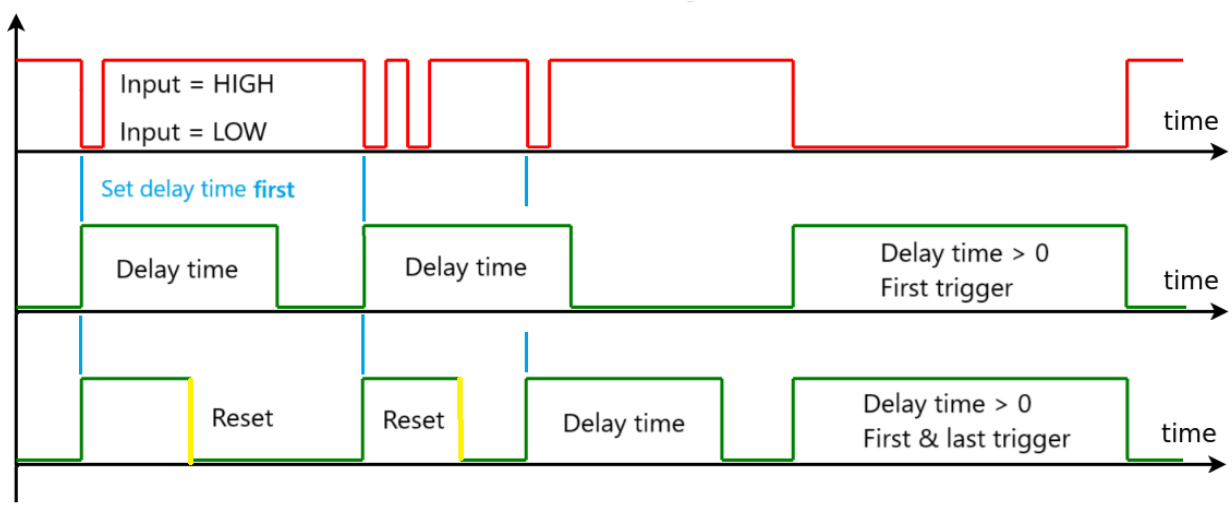
Sensordetektering af et togsæt i bevægelse vil uundgåeligt medføre et antal impulser grundet variationer i opbygning af togvognene og lokomotiv, samt "huller" ved sammenkoblinger med mere.

Eksempel på kørende tog



Time delay – first trigger

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



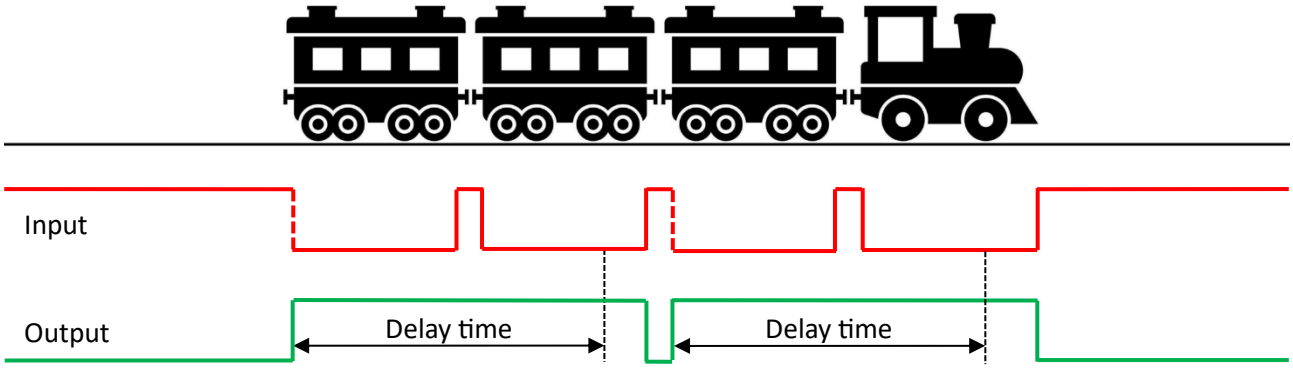

```
object_name.reset_timer ();
```

Nulstiller alle dynamiske variable og gør klar til ny start. Alle konfigurerede variable og standard værdier bibeholdes.

Eksempel på kørende tog

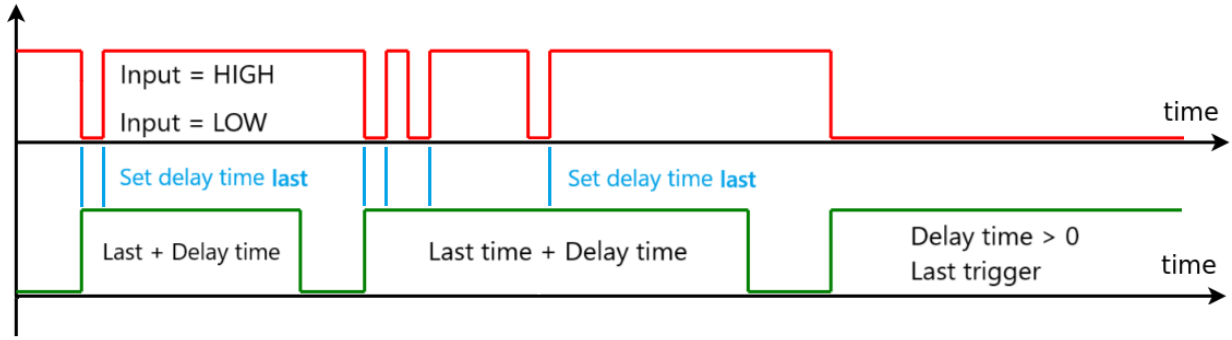
delayTimeMS = 5.000 millisekunder og triggerMode = firstTrigger.
12 sekund forsinkelse målt fra første detektering af tog.

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



Time delay – last trigger

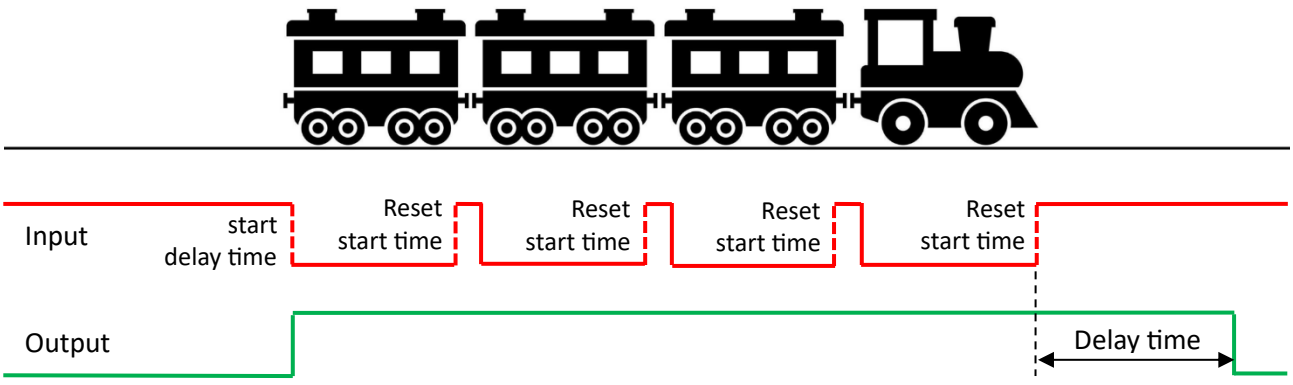
Når input går fra HIGH til LOW fastholdes output HIGH ind til at den definerede tidsperiode ophører. Hver gang input går fra LOW til HIGH forskydes starten for tidsperioden til det nye tidspunkt. Er input LOW ved tidsperiodens ophør, forbliver output HIGH, ind til at input går fra LOW til HIGH.



Eksempel på kørende tog

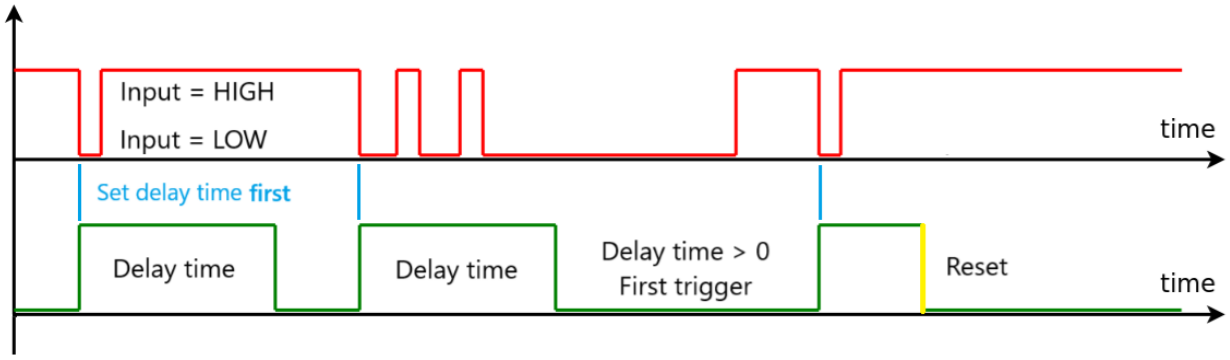
delayTimeMS = 5.000 millisekunder og triggerMode = lastTrigger.
5 sekunder forsinkelse målt fra sidste detektering af tog (LOW til HIGH).

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



Monostable – first trigger

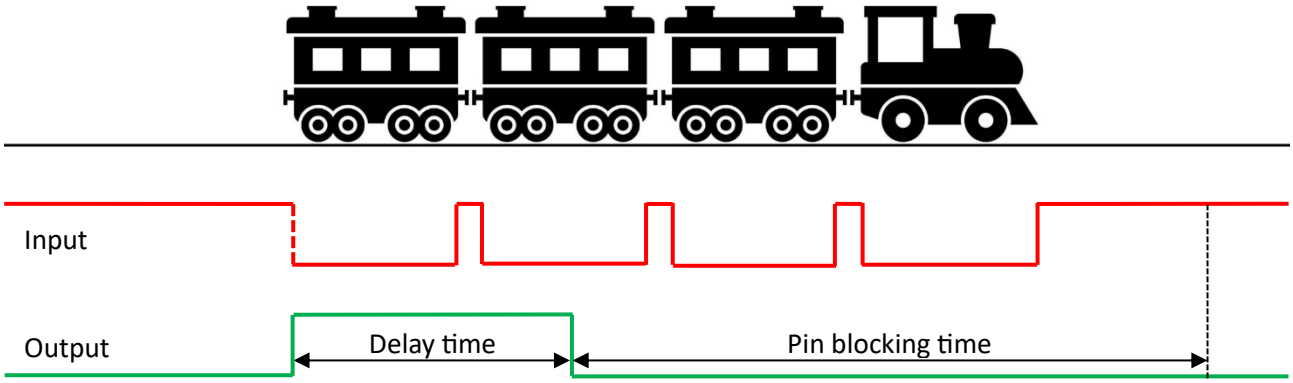
Monostabilt fastholder altid den definerede tidsperiode, uanset om input skifter mellem LOW og HIGH i tidsperioden, og forbliver input enten LOW eller HIGH, ændre det ikke tidsperioden.



Eksempel på kørende tog

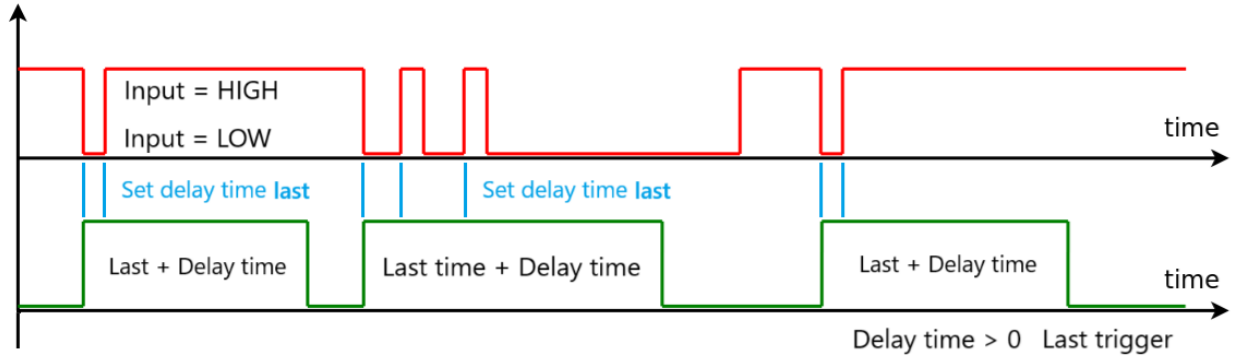
delayTimeMS =5.000 millisekunder, triggerMode = firstTrigger, timerMode = monoStable, pinBlockMS = 12.000 millisekunder (input fra benforbindelsen blokeres/ ignoreres ved monostabil afslutning).
5 sekunder forsinkelse målt fra første detektering af tog (HIGH til LOW).

```
MTD2A_binary_input FC_51_sensor ("FC_51_sensor", 5000, firstTrigger, monoStable, 12000);
```



Monostable – last trigger

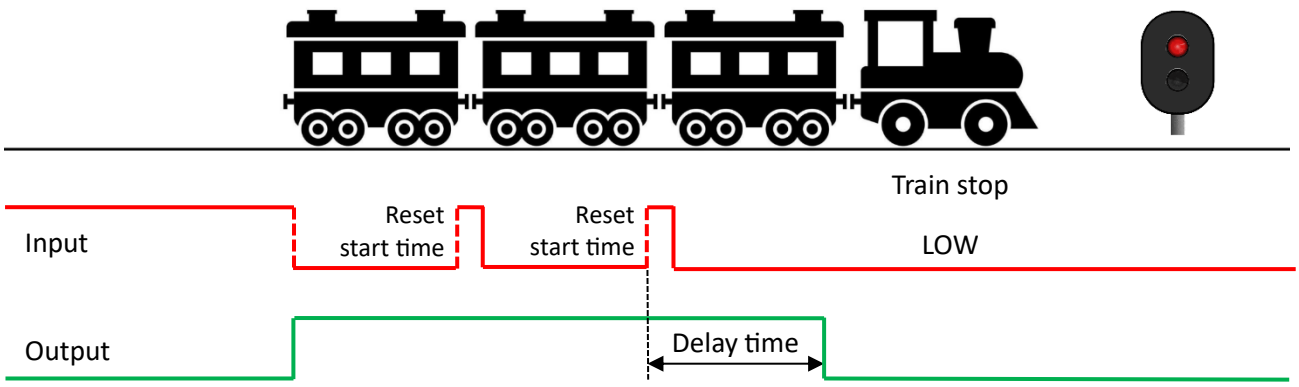
Monostabilt fastholder altid den definerede tidsperiode, men skifter input i mellem LOW og HIGH i tidsperioden, forlænges tidsperioden hver gang. Efter den samlede tidsperiode skifter output til LOW, uanset om input enten er LOW eller HIGH.



Eksempel på kørende tog der standser over sensor

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

```
MTD2A_binary_input FC_51_sensor ("FC_51_sensor", 3000, lastTrigger, monoStable);
```



Eksempler

Stop for rødt lys og afspil en besked til passagererne.

```
// Stop light and sound message: The train brakes and temporarily stops at a red light.

MTD2A_binary_input FC_51_stop ("Stop",  10000, lastTrigger, timeDelay
MTD2A_binary_input wait_time  ("Sound",  5000, firstTrigger, monoStable);
MTD2A_binary_input sound_time ("Time",   3000, firstTrigger, monoStable);

int soundPin = 9; // Danish and English speech from https://ttsmaker.com/

void setup() {
  Serial.begin(250000);
  FC_51_stop.init_setup (2);  // input pin 2
  // Debug information
  FC_51_stop.set_debugPrint  (on);
  wait_time.set_debugPrint (on);
  sound_time.set_debugPrint (on);
  // Output
  pinMode (soundPin, OUTPUT);

  bool trainSoundState = false;
}

void loop() {
  if (FC_51_stop.get_outputState() == active ) {
    // Continous activation (no drop outs)
    if (FC_51_stop.get_phaseChange() == true  &&  FC_51_stop.get_phaseNumber() == BIactive) {
      wait_time.set_inputState(LOW);
      wait_time.loop_fast();  // updtate setting (active)
    }
    // Waiting time before sound activation
    if (wait_time.get_phaseChange() == true  &&  wait_time.get_phaseNumber() == BIpending) {
      digitalWrite(soundPin, HIGH);  // write once
      Serial.println("Start sound track");
      sound_time.set_inputState(LOW);
    }
    // Sound duration
    if (sound_time.get_phaseChange() == true  &&  sound_time.get_phaseNumber() == BIpending) {
      digitalWrite(soundPin, LOW);  // write once
      Serial.println("End sound track");
    }
  }

  FC_51_stop.loop_fast();
  wait_time.loop_fast();
  sound_time.loop_fast();
  delay (10);
} // Stop light and sound message: The train brakes and temporarily stops at a red light.
```

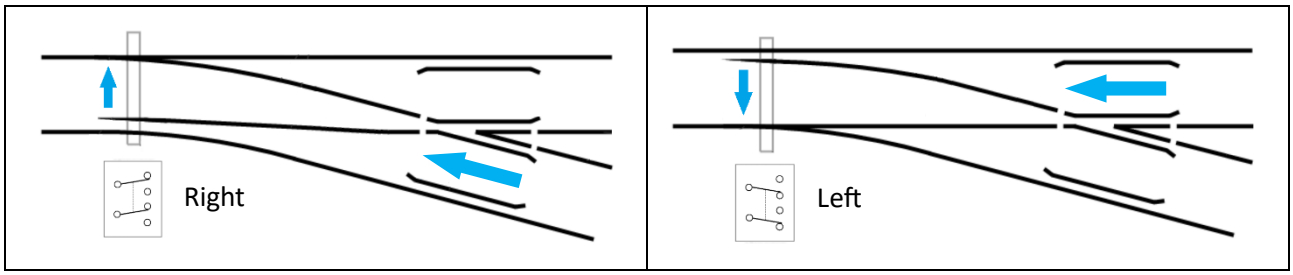
Bestemmelse af køreretning

```
// Example ---------------------------------------------------------------------------------

// Determining the direction of travel of the train

MTD2A_binary_input FC_51_left  ("Left", 5000, lastTrigger, timeDelay);
MTD2A_binary_input FC_51_right ("Right", 5000, lastTrigger, timeDelay);

bool directionActive = false;
int redLEDpin   = 9;
int greenLEDpin = 10;

void setup() {
  Serial.begin(250000);
  FC_51_left.init_setup  (2);  // input pin 2 (FC-51 left)
  FC_51_right.init_setup (3);  // input pin 3 (Fc-51 right)
  FC_51_left.set_debugPrint  (on);
  FC_51_right.set_debugPrint (on);
  pinMode (redLEDpin, OUTPUT);
  pinMode (greenLEDpin, OUTPUT);
}

void loop() {
  if (FC_51_left.get_outputState() == active  &&  FC_51_right.get_outputState() == pending) {
    // From left;
    if (directionActive == false) {
      digitalWrite(redLEDpin, HIGH);  // write once
      Serial.println("Moving left");
      directionActive = true;
    }
  }
  else
    if (FC_51_left.get_outputState() == pending  &&  FC_51_right.get_outputState() == active) {
      // From right;
      if (directionActive == false) {
        digitalWrite(greenLEDpin, HIGH);  // write once
        Serial.println("Moving rigth");
        directionActive = true;
      }
    }

 if (FC_51_left.get_outputState() == pending  &&  FC_51_right.get_outputState() ==
pending  &&  directionActive == true) {
    // Keep output activated for 5 seconds
    Serial.println("Turn off red and green LED");
    digitalWrite(redLEDpin, LOW);
    digitalWrite(greenLEDpin, LOW);
    directionActive = false;
```

```
  }

  FC_51_left.loop_fast();
  FC_51_right.loop_fast();
  delay (10);
} // Determining the direction of travel of the train
```

Skift automatisk spor efter køreretning

Hvis skiftesporet ikke passer med køreretningen, skifter sporet.
Kort efter at toget har passeret skiftesporet, skifter sporet tilbage til den oprindelige position.




```
// Example ----------------------------------------------------------------------------

// Automatic switch to the direction the train is coming from (to avoid derailment)
// Standard LGB switch with position switches for feedback is used

// Track sensor pin input
MTD2A_binary_input FC_51_left     ("Left track sensor",  10000, lastTrigger, timeDelay
MTD2A_binary_input FC_51_right    ("Right track sensor", 10000, lastTrigger, timeDelay
// Send 0,5 seconds pulse to switch motor
MTD2A_binary_input motor_left     ("Left switch motor", 500, firstTrigger, monoStable
MTD2A_binary_input motor_right    ("Right swith motor", 500, firstTrigger, monoStable);
// read current switch position
MTD2A_binary_input position_left  ("Left feedback position");
MTD2A_binary_input position_right ("Right feedback position");

int RailSwitchLeft   = 9;
int RailSwitchRight = 10;

bool trackShiftState = false;

void setup() {
  Serial.begin(250000);
  FC_51_left.init_setup     (2);  // input pin 2
  FC_51_right.init_setup    (3);  // input pin 3
  position_left.init_setup  (4);  // input pin 4
  position_right.init_setup (5);  // input pin 5

  // Debug print - show what is going on
  FC_51_left.set_debugPrint     (on);
  FC_51_right.set_debugPrint    (on);
```

```
  motor_left.set_debugPrint     (on);
  motor_right.set_debugPrint    (on);
  position_left.set_debugPrint  (on);
  position_right.set_debugPrint (on);
  // Switch motor
  pinMode (RailSwitchLeft, OUTPUT);
  pinMode (RailSwitchRight, OUTPUT);
  digitalWrite (RailSwitchLeft, LOW);
  digitalWrite (RailSwitchRight, LOW);
}

void loop() {
  if (FC_51_left.get_outputState() == active) {
    if (position_right.get_outputState() == active) {
      // Track switch is currently right
      trackShiftState = true;
      motor_left.set_inputState(LOW); // Activate track switch motor for 500 milliseconds
      // Write only once (a loop) to avoid "flicker"
      if (motor_left.get_outputState() == active  &&  motor_left.get_phaseChange() == true) {
        digitalWrite (RailSwitchLeft, HIGH);
        Serial.println("Move track switch to the left");
      }
      // Write only once (a loop) to avoid "flicker"
      if (motor_left.get_outputState() == pending  &&  motor_left.get_phaseChange() == true)
        digitalWrite (RailSwitchLeft, LOW);
    }
  }
  else {
    if (trackShiftState == true) {
      motor_right.set_inputState(LOW); // Activate track switch motor for 500 millisecond
      // Write only once (a loop) to avoid "flicker"
      if (motor_right.get_outputState() == active  &&  motor_right.get_phaseChange() == true) {
        digitalWrite (RailSwitchRight, HIGH);
        Serial.println("Move track switch back to the right");
      }
      // Write only once (a loop) to avoid "flicker"
      if (motor_right.get_outputState() == pending  &&  motor_right.get_phaseChange() == true) {
        digitalWrite (RailSwitchRight, LOW);
        trackShiftState = false;  // last command in the process
      }
    }
  }

  // Track Switch to the right is identical to Track Switch on the left, just mirrored.

  FC_51_left.loop_fast     ();
  FC_51_right.loop_fast    ();
  motor_left.loop_fast     ();
  motor_right.loop_fast    ();
  position_left.loop_fast  ();
```

```
  position_right.loop_fast ();

  delay (10);
} // Automatic track switch to the direction the train is coming from
```

Funktioner

Set functions	Comment
object_name.set_pinRead ({enable | disable});	Enable and disable pin reading.
object_name.set_pinInput ({normal | inverted});	Configure pin trigger input.
object_name.set_pinState ({HIGH | LOW}, {pulse | fixed});	Activate input state and set input mode.
object_name.set_debugPrint ({on | off});	Activate print state information (Active, etc.)

Get functions	Comment
object_name.get_outputState (); return {active | pending}	Output state.
object_name.get_phaseChange (); return {true | false}	Phase change.
object_name.get_phaseNumber (); return {0 - 4}	Initialize and reset = 0, active = 1, Set last time = 2, Pin block = 3, Pending = 4
object_name.get_firstTimeMS (); return milliseconds	First time trigger time (falling edge)
object_name.get_lastTimeMS (); return milliseconds	Last time trigger time (rising edge)
object_name.get_endTimeMS (); return milliseconds	End time (total delay time)
object_name.get_inputGoLow (); return {true | false}	Falling edge detected
object_name.get_inputGoHigh (); return {true | false}	Rising edge detected
object_name.get_reset_error (); return {0-255}	Get error/warning number and reset number Error {0 - 127} and warning {128 – 255} number

Operator overloading	Function
object_name_1 == object_name_2	bool outputState_1 == outputState_2
object_name_1 != object_name_2	bool outputState_1 != outputState_2
object_name_1 > object_name_2	uint32_t firstTimeMS_1 > firstTimeMS_2
object_name_1 < object_name_2	uint32_t firstTimeMS_1 > firstTimeMS_2
object_name_1 >> object_name_2	uint32_t lastTimeMS_1 >> lastTimeMS_2
object_name_1 << object_name_2	uint32_t lastTimeMS_1 << lastTimeMS_2

object_name.print_conf ();

```
MTD2A_binary_input:
-------------------
  objectName  : Left switch motor
  triggerMode : First
  timerMode   : Mono
  pinNumber   : 255 (Program input)
  pinType     : INPUT_PULLUP
  pinRead     : Disable
  pinInput    : Normal
  inputMode   : Pulse
  errorNumber : 0
  debugPrint  : On
  delayTimeMS : 500
  firstTimeMS : 21478
  lastTimeMS  : 21478
  endTimeMS   : 21997
  BlockTimeMS : 0
  pinState    : High
  inputState  : Low
  outputState : Pending
  phaseChange : False
  phaseNumber : [4] Pending
```