MTD2A_binary_output

MTD2A: Model **Train Detection And Action – arduino library** https://github.com/MTD2A/MTD2A
Jørgen Bo Madsen / V1.3 / 28-06-2025

MTD2A_binary_output is an easy-to-use, advanced and functional C++ class for time-controlled handling of output for relays, LEDs, and more. 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_output process consists of 4 functions:

1. `MTD2A_binary_output object_name ("object_name" , outputTimeMS, beginTimeMS, endTimeMS, { BINARY | PWM }, pinBeginValue, pinEndValue);`

2. `object_name.initialize ( pinNumber, startPinValue );`
   Called in `void setup ();` and after `Serial.begin (9600);`

3. `object_name.activate();` Activate once and only work again when the process is completed (COMPLETE)

4. `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, parameters must be specified in ascending order. See example below:

```
MTD2A_binary_input object_name ();
MTD2A_binary_input object_name
 ("object_name");
 ("object_name" , outputTimeMS);
 ("object_name" , outputTimeMS, beginTimeMS);
 ("object_name" , outputTimeMS, beginTimeMS, endTimeMS);
 ("object_name" , outputTimeMS, beginTimeMS, endTimeMS, pinOutputMode);
 ("object_name" , outputTimeMS, beginTimeMS, endTimeMS, pinOutputMode, pinBeginVlaue);
 ("object_name" , outputTimeMS, beginTimeMS, endTimeMS, pinOutputMode, pinBeginVlaue, pinEndValue);
```

Defaults:

```
 ("Object name" , 0 , 0 , 0  , BINARY , HIGH  , LOW);
```

Example

```
// Two blinking LEDs. One with symmetric interval and another with asymetric interval.

#include <MTD2A.h>
using namespace MTD2A_const;

MTD2A_binary_output red_LED   ("Red LED",   400, 400); // 0.4 sec light, 0.4 sec no light
MTD2A_binary_output green_LED ("Green LED", 300, 700, 0, PWM, 96); // 0.3 / 0.7 sec PWM dimmed

void setup() {
  Serial.begin(9600);
  while (! Serial) { delay(10); } // ESP32 Serial Monitor ready delay

  byte RED_LED_PIN   = 9;   Arduino board pin number
  byte GREEN_LED_PIN = 10;  Arduino board pin number
  red_LED.initialize (RED_LED_PIN);
  green_LED.initialize (GREEN_LED_PIN);
  Serial.println("Two LED flashes");
}

void loop() {
  If (red_LED.get_processState()   == PENDING) {
    red_LED.activate();
  }
  if (green_LED.get_processState() == PENDING) {
    green_LED.activate();
  }

  MTD2A_loop_execute();
}
```

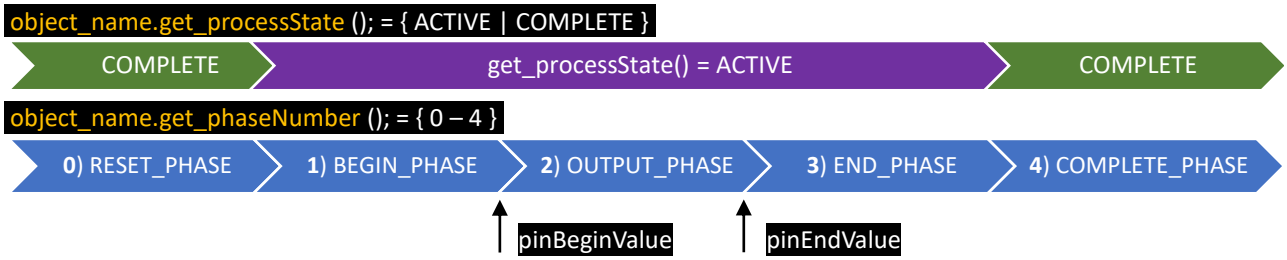
More examples and youtube video:

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

DEMO video: https://youtu.be/eyGRazX9Bko

Process phases

Depending on the current configuration, the process is carried out in between 1 and 5 stages.



0. The initial phase when the program starts and when the function `reset ();` is called.
1. Start delay. If set to 0 or not defined, the stage will be skipped.
2. Output time period. Starts with `pinBeginValue` and ends with `pinEndValue`.
   If set to 0 or not defined, the stage will be skipped.
3. End of delay. If set to 0 or not defined, the stage will be skipped.
4. The process is completed `COMPLETE` and ready for reactivation `object_name.activate ();`

Global number constants: `RESET_PHASE`, `BEGIN_PHASE`, `OUTPUT_PHASE`, `END_PHASE` & `COMPLETE_PHASE`

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

Process status

When transitioning to `BEGIN_PHASE`, `OUTPUT_PHASE`, or `END_PHASE`, ProcessState switches to `ACTIVE`.
When transitioning to `COMPLETE_PHASE` or `RESET_PHASE`, the processState switches to `COMPLETE`.

Timing

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

Stop and restart

It is possible to set the new start time for the current timer process, and stop the current timer process prematurely.

```
object_name.set_outputTimer (  {STOP_TIMER | RESTART_TIMER} );
object_name.set_beginTimer (  {STOP_TIMER | RESTART_TIMER} );
object_name.set_endTimer (  {STOP_TIMER | RESTART_TIMER} );
```

The new start time is retrieved from the globally synchronized time and can be read with the function:
`MTD2A_globalSyncTimeMS ();`

Initialization

The output is written to the digital pin connection number specified in object_name.initialize (pinNumber);
If the function is not called, the pin connection will not be written to. pinWrite=DISABLE and pinNumber = 255.

The output on the pin connection can be inverted (reversed) by setting the pinOutput to INVERTED
This means that HIGH and LOW are reversed and the PWM value is calculated to 255 – (minus) PWM value.
object_name.initialize (pinNumber, {NORMAL | **INVERTED**});

The pin connection is initialized with the value specified in the startPinValue. If the parameter is omitted, LOW is specified. object_name.initialize (pinNumber, {NORMAL | INVERTED}, **startPinValue**);

If the pin connection number is initialized correctly with object_name.initialize (); , it is possible to continuously control whether or not to write to the pin connection with the function:
object_name.set_pinWrite ({ENABLE | DISABLE});

It is also possible to write directly to the leg connection with the function:
object_name.set_setPinValue (setPinValue); {HIGH | LOW}/PWM {0-255}

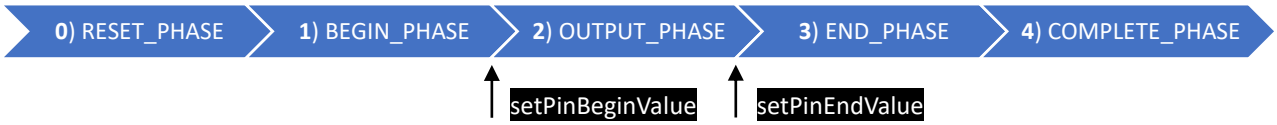
As a starting point, the pin connection is **undefined**. See [Digital Pins | Arduino Documentation]

Activation

The process is activated with the function: object_name.activate (); This switches the process State to ACTIVE
Subsequent activation has no effect as long as the process is active. As soon as the processState switches to COMPLETE, the process can be activated again.

The process can be reset at any time with the function: object_name.reset(); The function resets all control and process variables, and prepares for a fresh start. All functionally configured variables and default values are retained. In addition, the startPinValue is written for the pin connection if the pin connection is defined.
The process phase switches to RESET_PHASE

It is possible to write beginning values pinBeginValue and end values pinEndValue to the pin connection.



Activate functions use "function overloading". This means that the function can be called with none and up to 4 parameters. However, parameters must be specified in ascending order, starting from the first.

Default values are **not** used. All existing values remain the same unless values are specified as parameters in the function call. See example below:

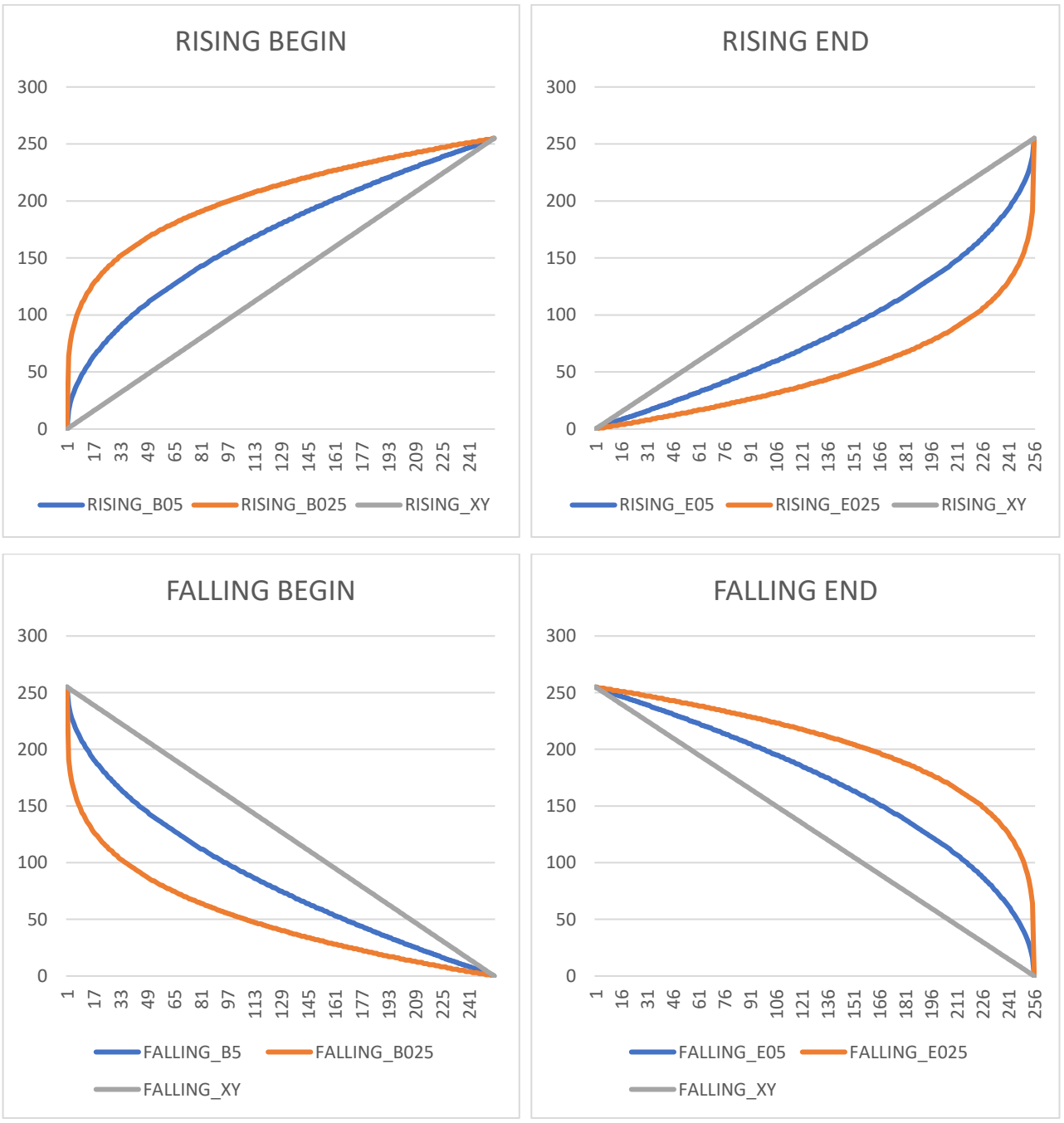
```
object_name.activate ();
object_name.activate (setPinBeginValue);
object_name.activate (setPinBeginValue, setPinEndValue);
object_name.activate (setPinBeginValue, setPinEndValue, setPWMcurveType);
object_name.activate (setPinBeginValue, setPinEndValue, setPWMcurveType, LoopFastOnce);
```

PWM (Pulse Width Modulation) – coming in the next version

setPWMcurveType specifies which curve to follow in the time period outputTimeMS The curve starts with the value setPinBeginValue and ends with the value setPinEndValue

As an example, it is possible – slowly – to turn up the light on an LED (fade in) and subsequently turn down the light (fade out) on the same LED. A more advanced example is shuttle service with model trains. The locomotive accelerates up to speed, runs at a fixed speed for a while, deaccelerates down to speed and comes to a complete stop. After a short pause, the process is repeated in the opposite direction.

Mathematical PWM curves



Clarification: Pulse-width modulation - Wikipedia, the free encyclopedia

The various curves are named global constants (MTD2A_const.h):

NO_CURVE
RISING_XY
RISING_B05
FALLING_B05

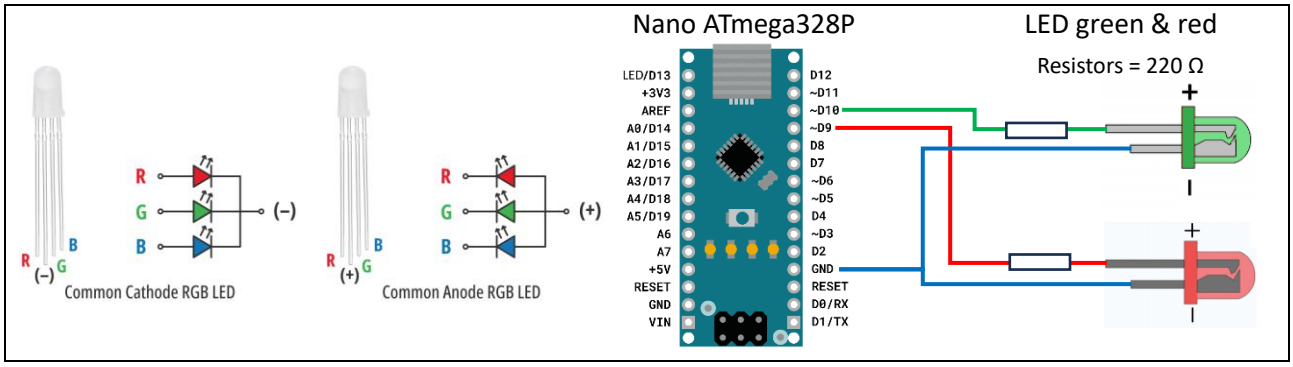
FALLING_XY
RISING_B025
FALLING_B025

RISING_E05
FALLING_E05

RISING_E025
FALLING_E025

Examples of configuration

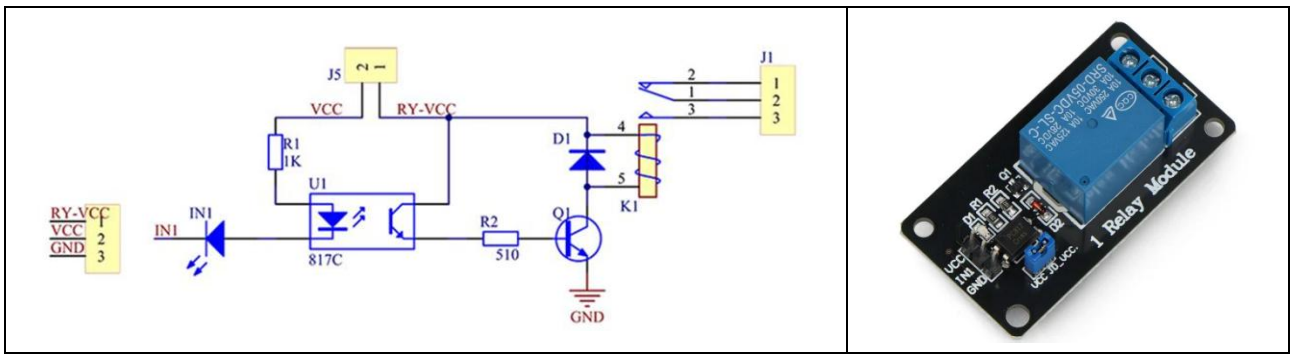
A multi-coloured LED with a common **cathode** is activated by changing from LOW to HIGH. PWM 0 -> 255.
A multi-colored LED with a common **anode** is activated by changing from HIGH to LOW. PWM 255 -> 0.



Example of a green LED that waits for half a second and then lights up for half a second.

```
1. MTD2A_binary_output green_LED ("Green LED", 500, 500);
2. green_LED.initialize (10);
3. green_LED.activate ();
4. MTD2A_loop_execute ();
```

A standard rail with optocoupler that is activated by going from HIGH to LOW.



Example of an optocoupled relay that waits for half a second and is then activated for half a second.
All default values are the inverse of what is the default. Therefore, all parameters must be specified.

```
1. MTD2A_binary_output opto_relay ("Opto relay",  500, 500, 0,  BINARY, LOW, HIGH);
2. opto_relay.initialize (12, NORMAL, HIGH);
3. opto_relay.activate ();
4. MTD2A_loop_execute ();
```

Alternatively, INVERTED, which inverts (reverses) all pin output values.

```
1. MTD2A_binary_output opto_relay ("Opto relay",  500, 500);
2. opto_relay.initialize(12, INVERTED);
3. opto_relay.activate ();
4. MTD2A_loop_execute ();
```

Other features

Set functions	Comment
set_pinWrite ({**ENABLE** | DISABLE});	Enable or disable pin writing
Set_pinOutput ({**NORMAL** | INVERTED});	Invert all output (pin writing)
set_setPinValue ({BINARY {HIGH | LOW}/PWM {0-255})	Write directly to output pin (if enabled).
set_outputTimer({ STOP_TIMER | RESTART_TIMER})	Stop timer process immediately or restart
set_beginTimer ({ STOP_TIMER | RESTART_TIMER})	Stop timer process immediately or restart
set_endTimer ({ STOP_TIMER | RESTART_TIMER})	Stop timer process immediately or restart
set_debugPrint ({**ENABLE** | DISABLE});	Activate print phase number and text

Common to all set functions is an additional parameter: LoopFastOnce = {ENABLE | **DISABLE**} disable is the default.

Get functions	Comment
get_processtState (); return bool {ACTIVE | COMPLETE}	Process state.
get_pinWrite (); return bool {ENABLE | DISABLE}	Write to pin is enabled or disabled
get_phaseChange (); return bool {true | false}	Momentarily phase change (one loop time)
get_phaseNumber (); return uint8_t {0 - 4}	Reset = 0, begin = 1, output = 2, end = 3, complete = 4.
get_setBeginMS (); return uint32_t milliseconds.	Start time for **begin** process
get_setOutputMS (); return uint32_t milliseconds.	Start time for **output** process
get_setEndMS (); return uint32_t milliseconds.	Start hour by **end** process
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 setOutputMS_1 > setOutputMS _2
object_name_1 << object_name_2	bool setOutputMS_1 < setOutputMS _2

print_conf();

object_name.print_conf ();

```
MTD2A_binary_output:
--------------------
  objectName    : LED_1
  processState  : ACTIVE
  phaseText     : [3] End delay
  debugPrint    : DISABLE
  globalDebugPr: DISABLE
  errorPrint    : DISABLE
  globalErrorPr: ENABLE
  errorNumber   : 0 OK
  outputTimeMS  : 2000
  beginDelayMS  : 0
  endDelayMS    : 2000
  pinOutputMode: PWM
  pinBeginValue: 10
  pinEndValue   : 0
  pinNumber     : 9
  pinWrite      : ENABLE
  pinOutput     : NORMAL
  startPinValue: 0
  setPinValue   : 0
  setBeginMS    : 0
  setOutputMS   : 2014
  setEndMS      : 4028
```