MTD2A_binary_output

MTD2A: Model **Train Detection And Action – arduino library** https://github.com/MTD2A/MTD2A
Jørgen Bo Madsen / V1.6 / 25-10-2025

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 );`
   Executed 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();
}
```

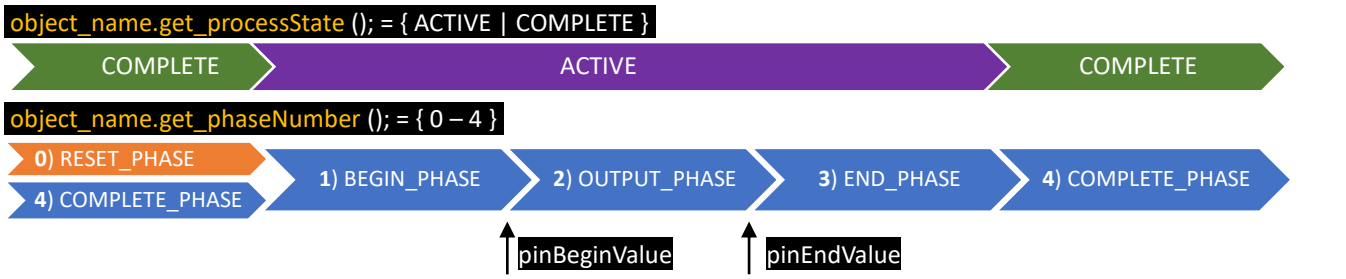
More examples and youtube video:

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 or 4) COMPLETE_PHASE.
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, and a warning is written.
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 reset (restart) timer

It is possible to set the new start time for the current timer process, and stop the current timer process prematurely.
RESET_TIMER Sets a new start time which is retrieved from the globally synchronized time globalSyncTimeMS
STOP_TIMER interrupts the time control process

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

The new start time is retrieved from the globally synchronized time and can be read with the function:
MTD2A::globalSyncTimeMS ();

The time period is set by default when the object is instantiated (activated). It is possible to define new time periods for all three time periods:

```
object_name.set_outputTimeMS ( {0 - 4294967295} );
object_name.set_beginDelayMS ( {0 - 4294967295} );
object_name.set_endDelayMS ( {0 - 4294967295} );
```

Initialization

object_name.initialize (pinNumber, startPinValue); Is executed in void setup (); and after Serial.begin (9600);
The output is written to the digital pin connection number specified in object_name.initialize (pinNumber);
If the function is not called, or an unsupported pin connection is specified, the pin connection will not be written to: pinWriteToggl = 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.
startPinValue is written **instantly** to pinNumber If the parameter is omitted, LOW is specified.
object_name.initialize (pinNumber, {**NORMAL** | INVERTED}, **startPinValue**);

If the pin connection number is defined 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_pinWriteToggl ({**ENABLE** | DISABLE});

It is also possible to write directly to the pin connection with the function:
object_name.set_PinWriteValue (PinValue); {HIGH | LOW} / PWM {0-255}
object_name.set_PinWriteValue (PinValue, {BINARY | P_W_M}); Optinal parameter. No default value.
This happens **instantly**, regardless of whether the process is active or not, and independently of MTD2A_loop_execute ();

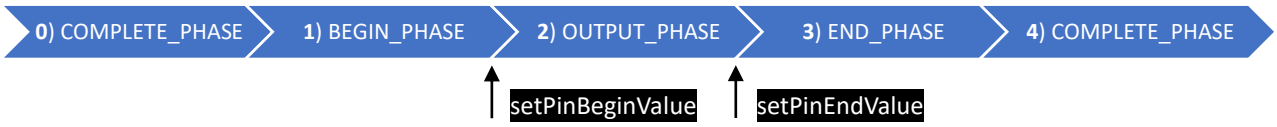
As a starting point, the pin connection is **undefined**. See Digital Pins | Arduino Documentation

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 **instantly**, and before MTD2A_loop_execute ();), and prepares for a fresh start. All functionally configured variables and default values are retained. The process phase switches to RESET_PHASE
If the pin connection is defined correctly, the startPinValue is written to the pin connection **instantly**.

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.

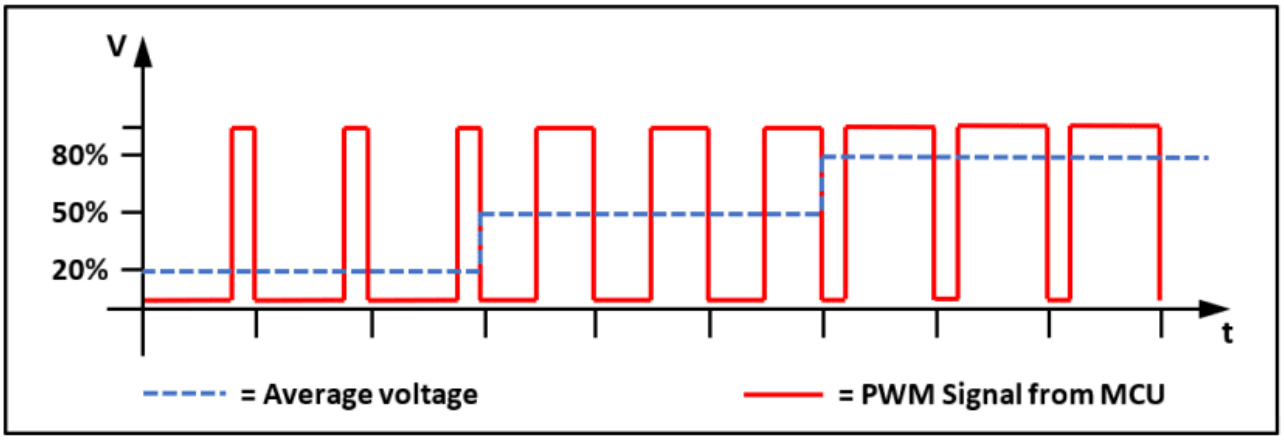
Default values are **not** used. All existing values remain the same unless values are specified as parameters in the function call. See example below:

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

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

PWM

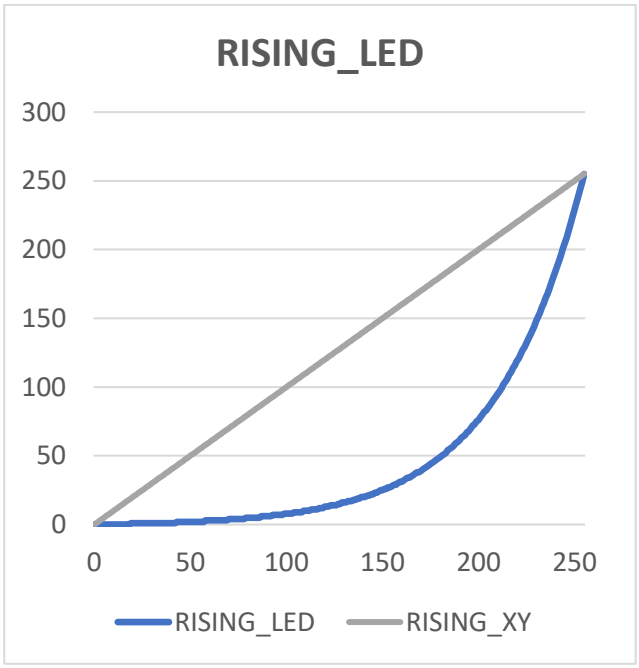
PWM (Pulse Width Modulation) is a technique used to control the average power delivered to an electrical device by varying the width of a series of pulses. Essentially, it's a digital method of creating an analog effect by rapidly switching a signal on and off

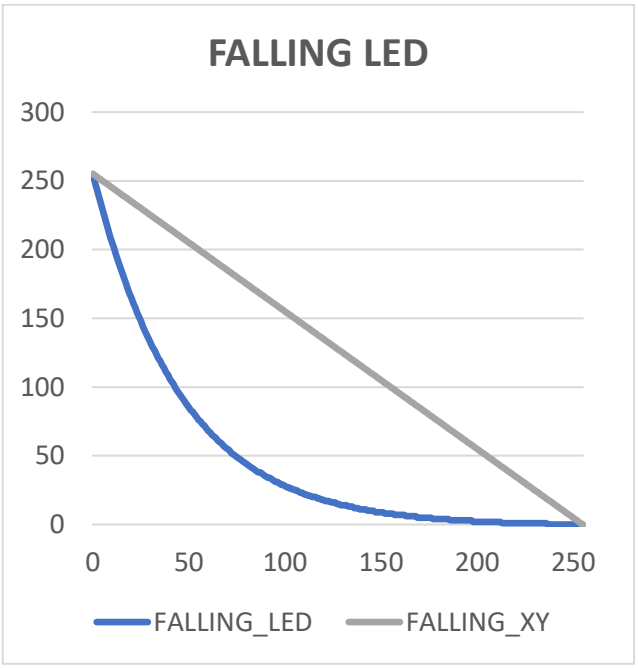


16 different math PWM curves are designed: 2 linear, 4 exponential and 10 power functions. The functions are intended to smooth out physical conditions that have a nonlinear relationship between time and function. E.g. LED fade in/fade out and starting and stopping DC motors (trains).

LED fading

PWM is a good way to to control the brightness of LEDs. However, the relationship between the duty cycle and the perceived brightness is not linear at all. Humans perceive the brightness change non-linearly. The human eye responds to light in a logarithmic fashion and has a better sensitivity at low luminance than high luminance.

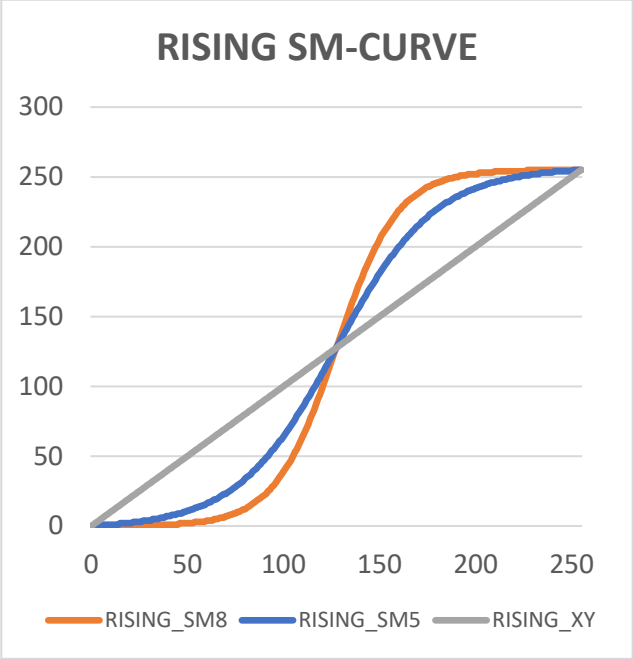


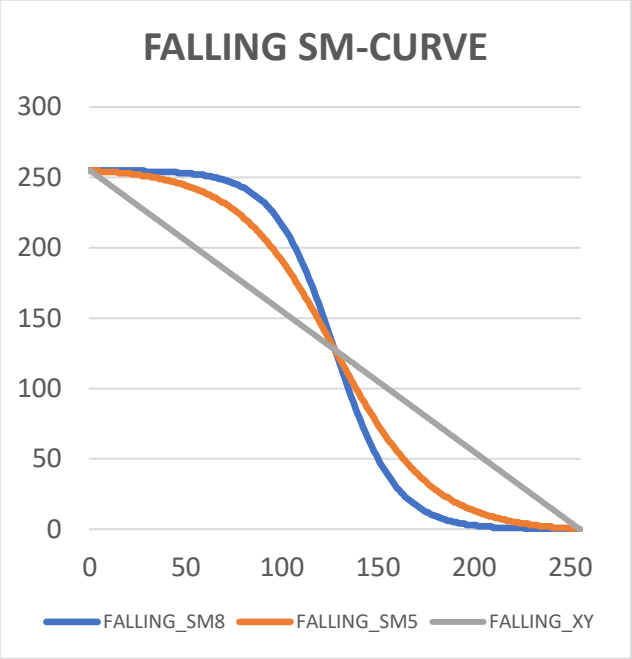


Arduino example: MTD2A/examples/math_fade_LED at main · MTD2A/MTD2A
DEMO video: https://youtu.be/8TV6nOdXBno

S-curves

Sigmoid S-curves can be used where soft acceleration and deceleration is preferable. For example, driving a (heavy loaded) servo motor from point a to b, or a servo motor that lowers/rasing booms up and down at a railroad crossing.





Arduino example: MTD2A/examples/servo_math_curve at main · MTD2A/MTD2A
DEMO video: https://youtu.be/rhQtu0iKFl8

Power functions

Power function is for smoothing acceleration and deceleration of trains.

For example, using power functions is suitable for shuttle service with model trains. The locomotive accelerates smotthly up to speed, runs at a fixed speed for a while, deaccelerates smoothly down to speed and comes to a complete stop. After a short pause, the process is repeated in the opposite direction.

Inertia and friction

Starting and stopping of train (locomotive and cars) involves both inertia and friction.

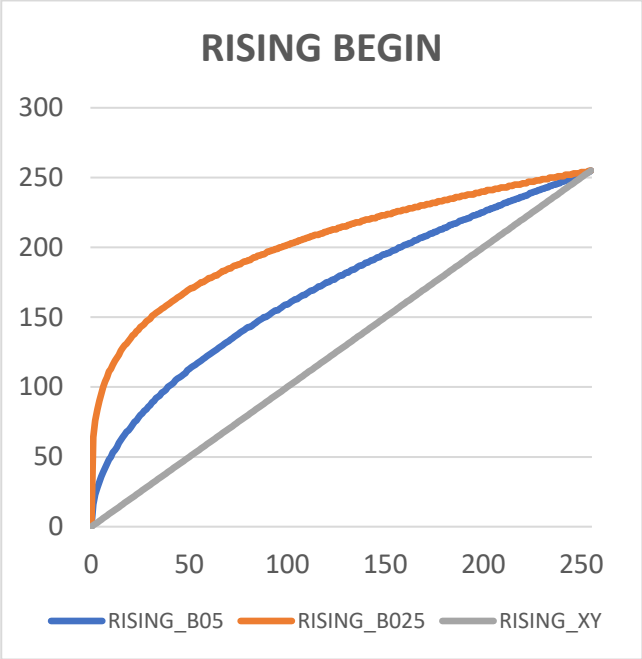
Inertia is the property of an object that causes it to resist changes to its state of motion. To start or stop motion, an unbalanced force is needed to overcome inertia and, in most real-world scenarios, friction.

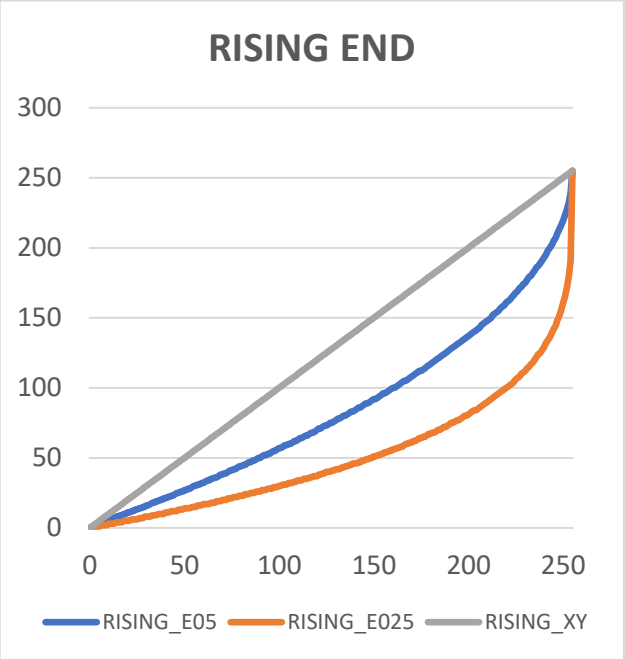
- A car accelerating requires a force to overcome the inertia of the car, which wants to remain stationary.
- A rolling ball eventually stops due to friction, but without friction, it would continue rolling indefinitely.
- Pushing a box: You need to push with enough force to overcome static friction and get the box moving. Once moving, you need to continue applying force to overcome kinetic friction and maintain the speed.

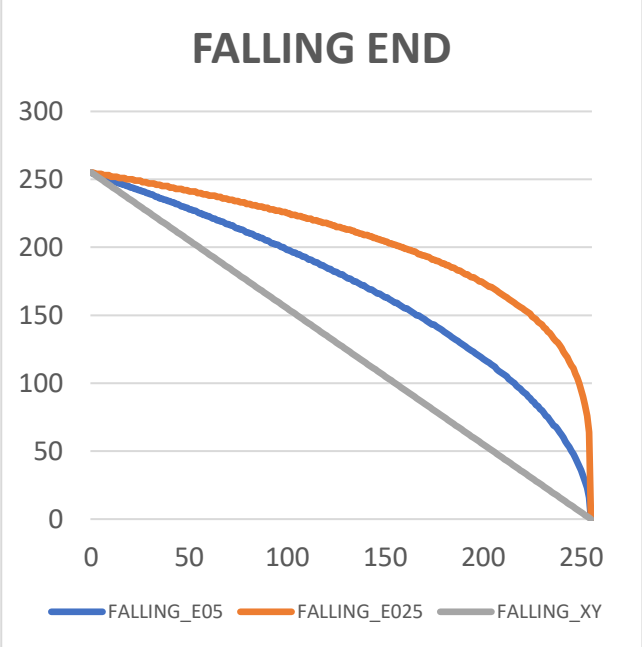
Inertia and friction are fundamental concepts in physics that explain why objects move (or don't move) and how motion can be started and stopped, and the relationship between time and function is **nonlinear**.

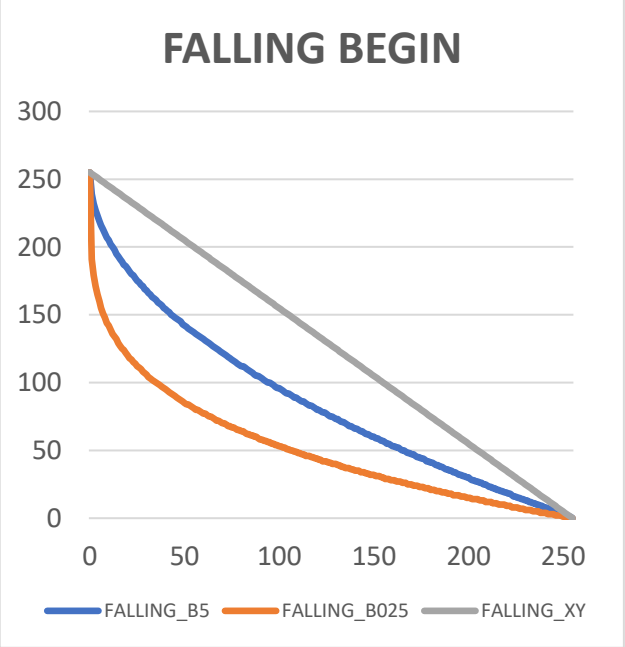
Furthermore there can be an effect in DC brushed motors called "armature reaction." Depending on how the motor is manufactured there can be an asymmetry between forward and reverse rotation.

The following 8 power functions will result in more smooth train acceleration and deceleration.









Arduino example:
DEMO video:

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

MIN_PWM_VALUE

MAX_PWM_VALUE

NO_CURVE
RISING_XY
RISING_LED
RISING_SM8
RISING_B05
FALLING_B05

FALLING_XY
FALLING_LED
RISING_SM5
RISING_B025
FALLING_B025

FALLING_SM8
RISING_E05
FALLING_E05

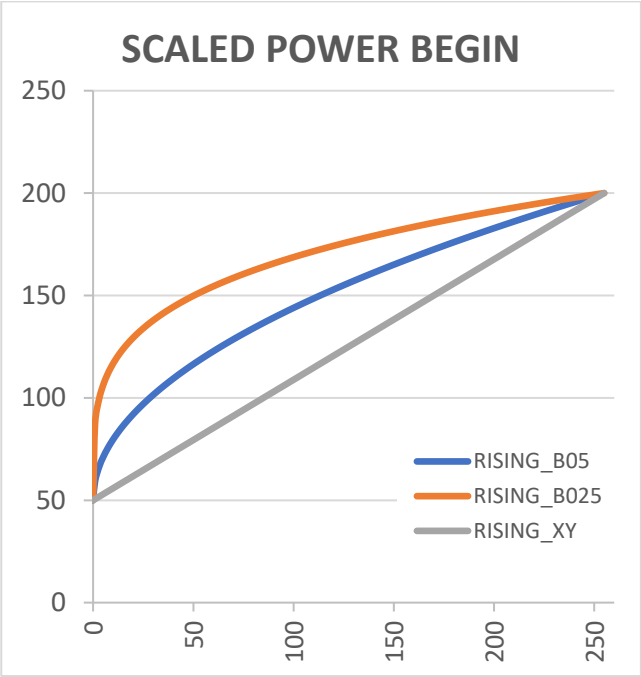
FALLING_SM5
RISING_E025
FALLING_E025

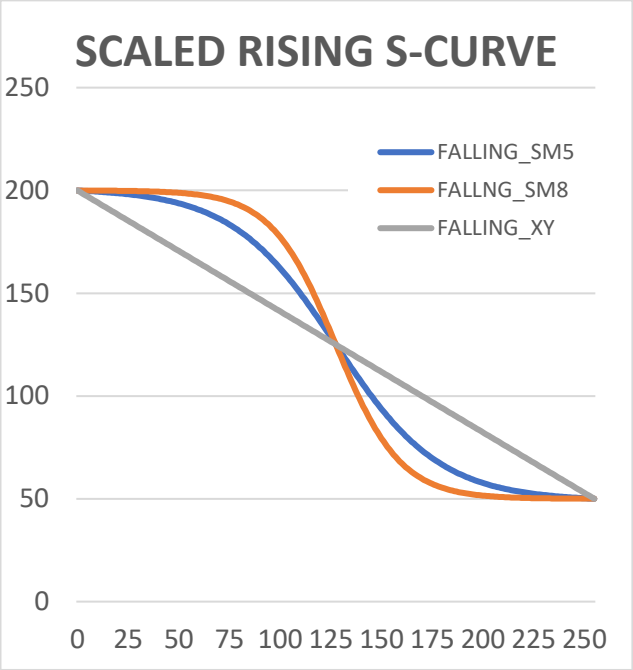
All curves can be scaled. If the value setPinBeginValue and the value setPinEndValue are defined, the entire curve is scaled within the time period outputTimeMS

Examples of scaling:

```
motor.activate (50, 200, RISING_XY);
motor.activate (50, 200, RISING_B05);
motor.activate (50, 200, RISING_B025);
```

```
servo.activate (200, 50, RISING_XY);
servo.activate (200, 50, RISING_SM8);
servo.activate (200, 50, RISING_SM5);
```

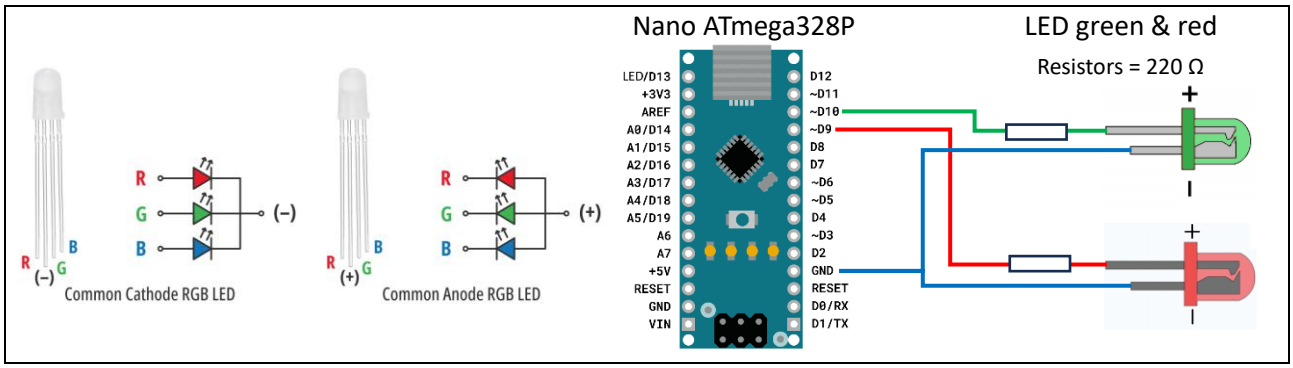




The individual calculated points on the curve can be read with the function: object_name.getOuputValue ();

Examples of configuration

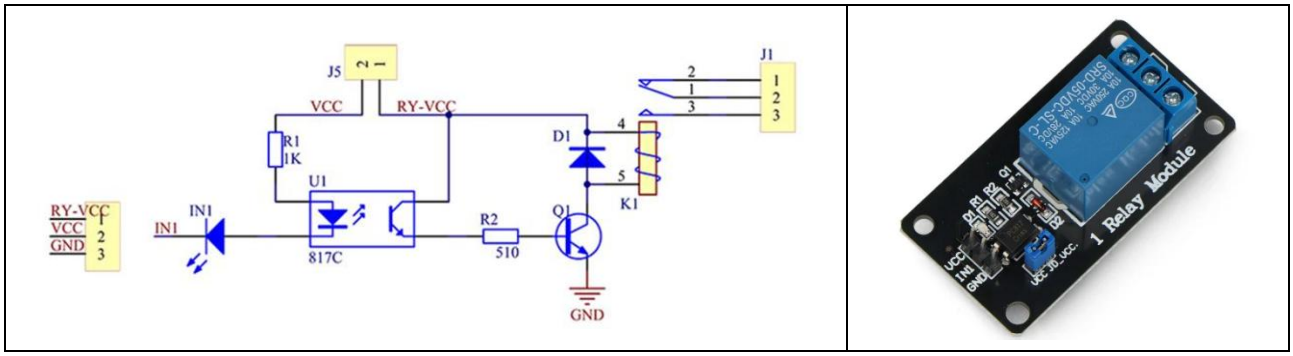
A multi-coloured LED with a common **cathode** is activated by changing from LOW to HIGH. PWM 0 -> 255.
A multi-colored LED with a common **anode** is activated by changing from HIGH to LOW. PWM 255 -> 0.



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 ();
```

Set get functions

Set functions Green is default when no parameter	Comment
set_PinOutputMode ({ **BINARY** | P_W_M });	Select binary og PWM output
set_pinWriteToggl ({**ENABLE** | DISABLE});	Enable or disable pin writing
set_pinWriteMode ({**NORMAL** | INVERTED});	Invert all output (pin writing)
set_pinWriteValue ({BINARY {HIGH | LOW} / PWM {0-255}) , {BINARY | P_W_M});	Write directly to output pin (if enabled) **Optinal** parameter. No default
set_outputTimer ({ STOP_TIMER | RESTART_TIMER})	Stop timer process or reset
set_beginTimer ({ STOP_TIMER | RESTART_TIMER})	Stop timer process or reset
set_endTimer ({ STOP_TIMER | RESTART_TIMER})	Stop timer process or reset
set_debugPrint ({**ENABLE** | DISABLE});	Enable print phase number and text
set_errirPrint ({**ENABLE** | DISABLE});	Enable error messages

Get functions	Comment
get_processtState (); return bool {ACTIVE | COMPLETE}	Process state.
get_pinWriteToggl (); return bool {ENABLE | DISABLE}	Write to pin is enabled or disabled
get_pinWriteMode (); return bool {NORMAL | INVERTED}	Output is normal or inverted
get_pinOutputValue() return uint8_t {HIGH | LOW} / {0- 255}	Current output value
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: P_W_M
  PWMcurveType  : 0
  pinBeginValue: 10
  pinEndValue   : 0
  pinNumber     : 9
  pinWriteToggl: ENABLE
  pinOutput     : NORMAL
  startPinValue: 0
  PinWriteValue: 0
  setBeginMS    : 0
  setOutputMS   : 2014
  setEndMS      : 4028
```