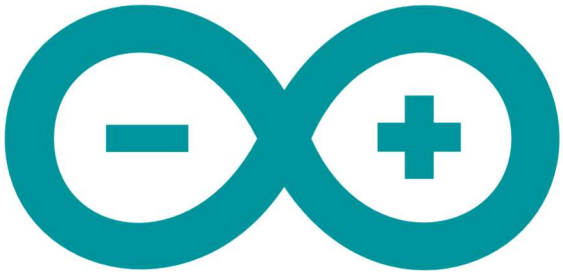
Dokumentacja techniczna

Biblioteka ARDUINO



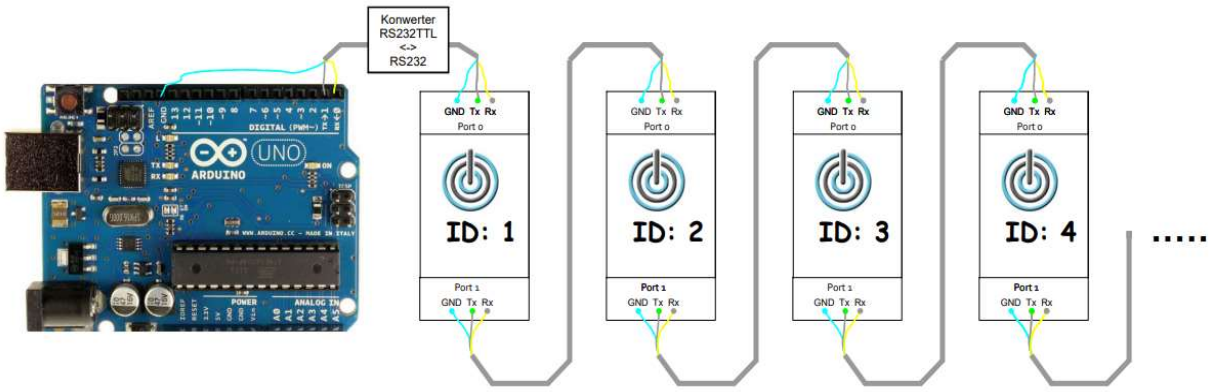
wersja: 1.18
08.2021

Autor:

Illia Kaidanov

1. Wstęp

Poniższy dokument opisuje protokół komunikacji pomiedzy Systemem IUVO a platwormą Arduino. Komunikacją pomiedzy systemem IUVO a płytkami ARDUINO odbywa się za pomocą portu RS232.



Biblioteka pozwala pobierać modułowi Arduino, informację o stanie systemu IUVO(stan wejść, wyjść, rolet…) oraz wydawać rozkazy do modułów IUVO.

2. Struktura programu

```
#include "iuvo_lib.h"

void SetupFunction(){ }

void ExecFunction(){ }
```

3. Funkcje obowiązkowe

3.1. SetupFunction

```
void SetupFunction(){}
```

Funkcja jest zadeklarowana w pliku nagłowkowym `"iuvo_lib.h"` *i jej implementacja musi byc w głównym pliku* `"*.ino"`.

Musi w sobie zawierać wywołanie funkcji `SetControllerQuantity`.

3.2. ExecFunction

```
void ExecFunction(){}
```

Funkcja jest zadeklarowana w pliku nagłowkowym `"iuvo_lib.h"` *i jej implementacja musi byc w głównym pliku* `"*.ino"`.

Zawiera w sobie stworzoną przez użytkownika logikę działań modułów.

Program musi być stworzony na tej zasadzie że ten program jest wykonywany cyklicznie. Dlatego wszystkie funkcje ***sterujące*** *muszą być wykonywane po sprawdzeniu funkcji* ***zdarzeniowych****.*

3.3. SetControllerQuantity

```
void SetControllerQuantity(byte quantity);
```

Opis:

Ustawia ilość modułów które będą monitorowane.
quantity = 1...20 *– ilość modułów.*

Przykład:

```
SetControllerQuantity(5);
```

3.4. SetControllerType

```
void SetControllerType(byte ControllerID, byte type);
```

Opis:

Rejestruje typ modułu

ControllerID = 1...20 *– ID modułu.*

type = 1...4/

1: IUVO Controller0806

2: IUVO Controller0806RTC

3: IUVO RollerShutter

4: IUVO Controller0806T

Jest możliwość wpisania wartości słownie:

```
#define IUVO_C0806 1
#define IUVO_C0806RTC 2
#define IUVO_RollerShutter 3
#define IUVO_C0806T 4
```

Przykład:

Rejestracja modułu 2 jako sterownika roletami

```
SetControllerType(2,IUVO_RollerShutter);
```

4. Funkcje informacyjne

4.1. GetInput

```
bool GetInput(byte ControllerID, byte InputID);
```

Opis:

Sprawdza czy wybrane wejście jest aktywne.
ControllerID = 1...20 *– ID modułu.*
InputID = 1...8 *– ID wejścia.*

Przykład:

Sprawdzenie wejścia 7 modułu 3.

```
if (GetInput(3,7))
```

4.2. GetOutput

```
bool GetInput(byte ControllerID, byte OutputID);
```

Opis:

Sprawdza czy wybrane wyjście jest aktywne.
ControllerID = 1...20 *– ID modułu.*
OutputID = 1...6 *– ID wyjścia.*

Przykład:

Sprawdzenie wyjścia 4 modułu 1.

```
if (GetOutput(1,4))
```

4.3. GetLamp

```
bool GetLamp(byte ControllerID, byte LampID);
```

Opis:

Sprawdza czy wybrane flaga/lampka jest aktywna.
ControllerID = 1...20 *– ID modułu.*
LampID = 1...8 *– ID flagi/lampki.*

Przykład:

Sprawdzenie flagi/lampki 4 modułu 2.

```
if (GetLamp(2,4))
```

4.4. GetTemperature

```
float GetTemperature(byte ControllerID, byte TempSensID);
```

Opis:

Zwraca wartość odczytu temperatury wybranego czujnika.
ControllerID = 1...20 *– ID modułu.*
TempSensID = 1...2 *– ID czujnika temperatury.*

Przykład:

Sprawdzenie czujnika 1 modułu 3.

```
if (GetTemperature(3,1))
```

4.5. GetRoller

```
byte GetRoller(byte ControllerID, byte RollerID);
```

Opis:

Zwraca stan wybranej rolety.
ControllerID = 1...20 *– ID modułu.*
RollerID = 1...4 *– ID rolety.*

Odpowiedź:

1 = `0b0001`***:*** *Roleta jest otwierana*
2 = `0b0010`***:*** *Roleta jest zamykana*
4 = `0b0100`***:*** *Roleta jest zatrzymana po otwarciu*
8 = `0b1000`***:*** *Roleta jest zatrzymana po zamknięciu*

```
#define OPENING     0b0001
#define CLOSING     0b0010
#define OPEN_STOP   0b0100
#define CLOSE_STOP  0b1000
```

Przykład:

Sprawdzenie czy roleta 2 modułu 4 jest zamykana.

```
if (GetRoller(4,2) == CLOSING)
```

Dodatkowe możliwości:

Bity 0 i 1 odpowiedzi sygnalizują o tym że roleta się porusza, natomiast bity 2 i 3 syngalizują że roleta jest zatrzymana. Dla takiego porównania wykorzystywany musi być operator ***&*** *(bitwise AND).*

```
#define STOPPED     0b1100
#define MOVING      0b0011
```

Przykład:

Sprawdzenie czy roleta 2 modułu 4 jest w ruchu.

```
if (GetRoller(4,2) & MOVING)
```

4.6. GetMask

```
byte GetMask(byte ControllerID, byte group, byte RollerState);
```

Opis:

Zwraca stany wejść/wyjść/lamp/rolet w formacie jednego bajtu.
Każdy element grupy jest reprezentowany przez odpowiadający bit.
ControllerID = 1...255 *– ID modułu.*
group = 0...4/

0: *Wejścia*
1: *Wyjścia*
2: *Flagi/Lampki*
3: *Rolety*

Jest możliwość wpisania wartości słownie:

```
#define INPUT 0
#define OUTPUT 1
#define LAMP 2
#define ROLLER 3
```

RollerState *– Stan rolet. Opcjonalny parametr funkcji.*

Przykłady:

Maska wejść modułu 1.

```
GetMask(1, INPUT)
```

Sprawdzenie czy wszysktie flagi/lampki modułu 2 są odznaczone.

```
if (GetMask(2, LAMP) == 0b00000000)
```

*Sprawdzenie czy jest włączone **tylko** wyjście 2 modułu 3.*

```
if (GetMask(3, OUTPUT) == 0b000010)
```

*Sprawdzenie czy jest włączone **przynajmniej** wejście 3 **lub** 8 modułu 4.*

```
if (GetMask(4, INPUT) & 0b10000100)
```

Maska rolet modułu 5 które są otwierane.

```
GetMask(5, ROLLER, OPENING)
```

Sprawdzenie czy wszysktie rolety modułu 6 są zatrzymane.

```
if (GetMask(6, ROLLER, STOPPED) == 0b1111)
```

5. Funkcje zdarzeniowe

5.1. GetFlag

```
bool GetFlag(byte FlagID);
```

Opis:

Sprawdza czy flaga jest ustawiona (flagi są resetowane na końcu każdego cyklu programu).

FlagID = 0...10 *– ID sprawzanej flagi.*

Przykład:

Sprawdzenie flagi nr 3.

```
if (GetFlag(3))
```

5.2. GetInputEvent

```
bool GetInputEvent(byte event, byte ControllerID, byte InputID);
```

Opis:

Sprawdza czy na wybranym wejście wystąpiło zdarzenie sygnału.

event = 1…3/

1: *Zbocze opadające lub zbocze narastające*

2: *Zbocze opadające*

3: *Zbocze narastające*

4: *Krótkie naciśnięcie*

8: *Długie naciśnięcie*

Jest możliwość wpisania wartości słownie:

```
#define CHANGE 0b0001
#define FALLING 0b0010
#define RISING 0b0011
#define SHORT_PRESS 0b0100
#define LONG_PRESS 0b1000
```

ControllerID = 1...20 *– ID modułu.*

InputID = 1...8 *– ID wejścia.*

Przykład:

Detekcja zbocza opadającego na wejściu 7 modułu 3.

```
if (GetEdge(FALLING,3,7))
```

Detekcja krótkiego naciśnięcia na wejściu 3 modułu 2.

```
if (GetEdge(SHORT_PRESS,2,3))
```

6. Funkcje sterujące

6.1. SetFlag

```
void SetFlag(byte FlagID, bool set);
```

Opis:

Ustawia flagę o wybranym numerze (flagi są resetowane na końcu każdego cyklu programu).

FlagID = 0...20 *– ID flagi.*

set = true/false *– ustawić/odzaczyć flagę. Opcjonalny parametr, domyślnie ustawia flagę.*

Przykład:

Ustawienie flagi 2

```
SetFlag(2);
```

Zerowanie flagi 3

```
SetFlag(3, false);
```

6.2. SetOut

```
void SetOut (byte ControllerID, byte OutputID, byte action);
```

Opis:

Wysyła do wybranego modułu komendę o ustawieniu wyjścia.

ControllerID = 1...20 *– ID modułu.*

OutputID = 1...6 *– ID wyjścia.*

action = 0...3 *– akcja:*

0: *Nic nie robić*

1: *Włączyć*

2: *Wyłączyć*

3: *Zmienić na przeciwne*

Jest możliwość wpisania wartości słownie:

```
#define NONE 0
#define ON 1
#define OFF 2
#define TOGGLE 3
```

Przykład:

Włączyć wyjście 2 moduła 1

```
SetOut(1, 2, ON);
```

Przełączyć wyjście 4 modułu 3

```
SetOut(3, 4, TOGGLE);
```

6.3. SetOutMulti

```
void SetOutMulti(byte ControllerID, byte action1, byte action2,
byte action3, byte action4, byte action5, byte action6);
```

Opis:

Wysyła do wybranego modułu komendę o ustawieniu wszystkich wyjść danego modułu według wybranych akcji dla każdego wyjścia.

ControllerID = 1...20 *– ID modułu.*

action{x} = 0...3 *– akcja dla poszczególnego wyjścia o ID == {x}:*

0: *Nic nie robić*

1: *Włączyć*

2: *Wyłączyć*

3: *Zmienić na przeciwne*

Jest możliwość wpisania wartości słownie:

```
#define NONE 0
#define ON 1
#define OFF 2
#define TOGGLE 3
```

Przykład:

Moduł 4: włączyć wyjście 1 i 2, wyłączyć wyjście 3 i przełączyć wyjście 4, pozostawiając wyjścia 5 i 6 w poprzednim stanie.

```
SetOutMulti(4, ON, ON, OFF, TOGGLE, NONE, NONE);
```

6.4. SetOutGroup

```
void SetOutGroup(byte GroupID, byte action,
byte referrence_ControllerID, byte referrence_OutputID);
```

Opis:

Ustawia wszystkie wyjścia w wybranej grupie. Grupę trzeba stworzyć za pomocą funkcji ***AddToGroup****.*

GroupID = 1...4 *– ID grupy.*

action = 0...3 *– akcja*

0: *Nic nie robić*

1: *Włączyć*

2: *Wyłączyć*

3: *Zmienić na przeciwne*

Jest możliwość wpisania wartości słownie:

```
#define NONE 0
#define ON 1
#define OFF 2
#define TOGGLE 3
```

referrence_ControllerID = 1...20 *– ID modułu referencyjnego.*

referrence_OutputID = 1...6 *– ID wyjścia referencyjnego.*

Ostatnie dwa parametry są opcjonalne, ale są potrzebne tylko dla akcji „TOGGLE". Służą dla synchronizacji wyjść według jednego wyjścia.

Przykład:

Włączyć wszystkie wyjścia w grupie 2

```
SetOutGroup(2, ON);
```

Przełączyć wszystkie wyjścia w grupie 3, synchronizując ich z wyjściem 5 modułu 4

```
SetOutGroup(3, TOGGLE, 4, 5);
```

6.5. SetOutMask

```
void SetOutMask(byte ControllerID, byte mask, byte action);
```

Opis:

Wysyła do wybranego modułu komendę o ustawieniu wszystkich wyjść danego modułu według maski.
ControllerID = 1...20 *– ID modułu.*
mask = 0b000000...0b111111 *– wybiera jakie wyjścia zostaną zmodyfikowane. Każde wyjście jest reprezentowane przez odpowiadający bit.*
action = 0...4 *– akcja.*

0: *Nic nie robić*
1: *Włączyć*
2: *Wyłączyć*
3: *Zmienić na przeciwne*
4: *Maska (wyjścia zaznaczone jako „1” zostają* ***włączone****, a wyjścia zaznaczone jako „0” zostają* ***wyłączone****). Akcja domyślna.*

Jest możliwość wpisania wartości słownie:

```
#define NONE 0
#define ON 1
#define OFF 2
#define TOGGLE 3
#define MASK 4
```

Przykład:

Moduł 3: włączyć wyjścia 1, 2 i 6, pozostawiając wyjścia 3, 4 i 5 w poprzednim stanie.

```
SetOutMask(3, 0b100011, ON);
```

Moduł 4: przełączyć wyjścia 2, 4 i 6, pozostawiając wyjścia 1, 3 i 5 w poprzednim stanie.

```
SetOutMask(4, 0b101010, TOGGLE);
```

Moduł 5: włączyć wyjścia 3 i 5, wyłączyć wyjścia 1, 2, 4 i 6.

```
SetOutMask(5, 0b010100);
```

6.6. SetLamp

```
void SetLamp (byte ControllerID, byte LampID, byte action);
```

Opis:

Wysyła do wybranego modułu komendę o ustawieniu lampki/flagi.
ControllerID = 1...20 *– ID modułu.*
LampID = 1...6 *– ID wyjścia.*
action = 0...3 *– akcja:*

0: *Nic nie robić*
1: *Włączyć*
2: *Wyłączyć*
3: *Mrugnij (włącz na jeden cykl programu)*
Jest możliwość wpisania wartości słownie:

```
#define NONE 0
#define ON 1
#define OFF 2
#define BLINK 3
```

Przykład:

Włączyć lampkę/flagę 3 moduła 4

```
SetLamp(4, 3, ON);
```

6.7. SetLampMulti

```
void SetLampMulti(byte ControllerID, byte action1, byte action2,
byte action3, byte action4, byte action5,
byte action6byte action7,byte action8);
```

Opis:

Wysyła do wybranego modułu komendę o ustawieniu wszystkich lampek/flag danego modułu według wybranych akcji dla każdej lampki/flagi.

ControllerID = 1...20 *– ID modułu.*

action{x} = 0...3 *– akcja dla poszczególnej lampki/flagi o ID == {x}:*

0: *Nic nie robić*

1: *Włączyć*

2: *Wyłączyć*

3: *Mrugnij (włącz na jeden cykl programu)*

Jest możliwość wpisania wartości słownie:

```
#define NONE 0
#define ON 1
#define OFF 2
#define BLINK 3
```

Przykład:

Moduł 4: włączyć lampki 1 i 2, wyłączyć lampki 3 i 4, mrugnąć lampkami 5 i 6, pozostawiając lampki 7 i 8 w poprzednim stanie.

```
SetLampMulti(4, ON, ON, OFF, OFF, BLINK, BLINK, NONE, NONE);
```

6.8. SetLampMask

```
void SetLampMask(byte ControllerID, byte mask);
```

Opis:

Wysyła do wybranego modułu komendę o ustawieniu wszystkich lampek danego modułu według maski.
ControllerID = 1...20 *– ID modułu.*
mask = 0b00000000...0b11111111 *– wybiera jakie lampki zostaną zmodyfikowane. Każda lampka jest reprezentowana przez odpowiadający bit.*

Przykład:

Moduł 3: włączyć lampki 1, 2 i 6, pozostawiając lampki 3, 4, 5, 7 i 8 w poprzednim stanie.

```
SetOutMask(3, 0b00100011, ON);
```

Moduł 4: włączyć lampki 1, 4, 6 i 7, wyłączyć lampki 2, 3, 5 i 8.

```
SetOutMask(4, 0b01101001);
```

6.9. SetRol

```
void SetRol(byte ControllerID, uint16_t timer, byte RollerID,
byte action);
```

Opis:

Wysyła do wybranego modułu komendę o sterowaniu wybraną roletą.
ControllerID = 1...20 *– ID modułu.*
timer = 1...600 *– czas przez który roleta będzie się poruszała (wyrażony w wielokrotnościach 0.1 sekundy, czyli 50 oznacza 5 sekund). Nie gra roli jeżeli akcja jest „Stop”.*
RollerID = 1...4 *– ID rolety.*
action = 0...4 *– akcja:*

0: *Nic nie robić*
1: *Otworzyć*
2: *Zamknąć*
3: *Zatrzymać*
4: *„Krok” – każde kolejne wysłanie wywołuje następny krok ruchu rolet: Otwieranie->Stop->Zamykanie->Stop*
Jest możliwość wpisania wartości słownie:

```
#define NONE 0
#define CLOSE 1
#define OPEN 2
#define STOP 3
#define STEP 4
```

Przykład:

Moduł 1: zamykać roletę 2 przez 30 sekund.

```
SetRol(1, 300, 2, CLOSE);
```

Moduł 4: zatrzymać roletę 5.

```
SetRol(4, 300, 5, STOP);
```

6.10. SetRolMulti

```
void SetRolMulti(byte ControllerID, uint16_t timer, byte action1,
byte action2, byte action3, byte action4)
```

Opis:

Wysyła do wybranego modułu komendę o sterowaniu wszystkimi roletami tego modułu.
ControllerID = 1...20 *– ID modułu.*
timer = 1...600 *– czas przez który roleta będzie się poruszała (wyrażony w wielokrotnościach 0.1 sekundy, czyli 50 oznacza 5 sekund). Nie gra roli jeżeli akcja jest „Stop”.*
action{x} = 0...4 *– akcja dla rolety o numerze {x}:*

0: *Nic nie robić*
1: *Otworzyć*
2: *Zamknąć*
3: *Zatrzymać*
4: *„Krok” – każde kolejne wysłanie wywołuje następny krok ruchu rolet: Otwieranie->Stop->Zamykanie->Stop*
Jest możliwość wpisania wartości słownie:

```
#define NONE 0
#define CLOSE 1
#define OPEN 2
#define STOP 3
#define STEP 4
```

Przykład:

Moduł 1: zamykać wszystkie rolety oprócz rolety 4 przez 30 sekund.

```
SetRolMulti(1, 300, CLOSE, CLOSE, CLOSE, STOP);
```

Moduł 4: zatrzymać rolety 1 i 2, otwierać rolety 3 i 4 przez 45 sekund.

```
SetRolMulti(1, 450, STOP, STOP, OPEN, OPEN);
```

Moduł 6: Wykonywać następny krok dla rolet 1, 2 i 4 przez 32.8 sekund.

```
SetRolMulti(1, 328, STEP, STEP, NONE, STEP);
```

Ostatni przykład nie jest zalecany z przyczyny niemożliwości synchronizacji rolet. Dla synchronizacji kilku rolet działania istnieje funkcja ***SetRolGroup****, opisana na następnej stronie.*

6.11. SetRolGroup

```
void SetRolGroup(byte GroupID, uint16_t timer,
byte ref_ControllerID,byte ref_RollerID, bool unsafe)
```

Opis:

Synchronizuje wszystkie rolety w wybranej grupie według wybranej rolety, imitując komendę „STEP”. Grupę trzeba stworzyć za pomocą funkcji ***AddToGroup****.*
GroupID = 1...4 *– ID grupy.*
timer = 1...600 *– czas przez który roleta będzie się poruszała (wyrażony w wielokrotnościach 0.1 sekundy, czyli 50 oznacza 5 sekund).*

ref_ControllerID = 1...20 *– ID modułu referencyjnego.*
ref_RollerID = 1...4 *– ID rolety referencyjnego.*
Te dwa parametry służą dla synchronizacji rolet według wybranej rolety wybranego modułu. ***Roleta referencyjna musi być dodana do grupy.*** *W przeciwnym przypadku funckja nie zadziała.*

unsafe = true/false *– parametr bezpieczeństwa. Jest opcjonalny, domyślnie jest ustawiony jako „false” i oznacza to że jeśli przynajmniej jedna z rolet w grupie jest obecnie w ruchu – to pierwsze wywołanie tej funkcji najpierw zatrzyma wszystkie rolety w grupie i dopiero po drugim wywołaniu rolety zaczną ruch. Ustawienie tego parametru na „true” wyłączy to sprawdzenie. Nie jest zalecane ustawiać ten parametr.*

Przykład:

Grupa 1: synchronizować rolety według rolety 2 modułu 3, czas – 25 sekund.

```
SyncRolGroup(1, 250, 3, 2);
```

7. Funkcje pomocnicze

7.1. AddToGroup

```
void AddToGroup(byte GroupID, byte ControllerID, byte mask);
```

Opis:

Dodaje wybrane wyjścia lub rolety wybranego modułu do grupy. Wyjścia/rolety są wybierane za pomocą maski, czyli każde wyjście/roleta jest reprezentowana przez odpowiadający bit.

GroupID = 1...4 *– ID grupy. Standardowo ilość grup jest ograniczona do 4.*
ControllerID = 1...20 *– ID modułu referencyjnego.*
mask *– wybiera jakie wyjścia/rolety będą dodane do grupy.*
mask = 0b000000...0b111111 *– dla wyjść.*
mask = 0b0000...0b1111 *– dla rolet.*

Przykłady:

Tworzenie grupy 1 z wyjść 3 i 4 modułu 2 oraz wyjść 1 i 4 modułu 3.

```
AddToGroup(1, 2, 0b001100);
AddToGroup(1, 3, 0b001001);
```

Ponowne wywołanie funkcji z tym samym numerem grupy i numerem modułu zamieni starą maskę na nową. Czyli:

```
AddToGroup(2, 4, 0b001111);
AddToGroup(2, 5, 0b100001);
```

Teraz grupa 2 mieści w sobie wyjścia 1,2,3 i 4 modułu 4 i wyjścia 1 i 6 modułu 5. Kolejne wywołanie:

```
AddToGroup(2, 4, 0b010001);
```

Spowoduje że teraz grupa 2 mieści w sobie wyjścia 1 i 5 modułu 4 i wyjścia 1 i 6 modułu 5.

Uwaga: ta funkcja nie widzi różnicy między maską wyjść i maską rolet. Jest możliwość dodania modułów różnego typu do jednej grupy. Funkcje które wykorzystują systemę grup mają zabezpieczenie według typu modułów do których oni są skierowane, więc mieszane grupy nie stworzą problemów, ale jednak nie są zalecane.

7.2. Timer

```
bool Timer(byte TimerID, bool condition, long SetTime,
bool retriggerable);
```

Opis:

Funkcja tajmerowa. Ma dwa przeznaczenia – ustawienie oraz odczyt tajmera.

TimerID = 1...10 *– ID tajmera. Standardowo ilość tajmerów jest ograniczona do 10.*
condition *– warunek startu tajmera. Przyjmuję wartość boolean, która może być otrzymana z funkcji informacyjnych opisanych wyżej na stronach 123124.*
SetTime *– czas który będzie odliczany przez wybrany tajmer. Wyrażony w milisekundach. Nie jest zalecane ustawiać tajmer na więcej niż 10 godzin.*
retriggerable *– odpowiada za możliwość restartowania tajmeru przed końcem odliczania czasu. Opcjonalny parametr. Domyślnie ustawiony jako „false”. Ustawiemie „true” spowoduje że ponowne pojawienie się warunku zrestartuje tajmer ze wskazanym parametrem czasu.*

Przykłady:

Wykonanie działania po 5 sekundach po pojawieniu się zbocza narastającego na wejściu 2 modułu 3, z wykorzystaniem tajmeru 4.

```
if (Timer(4, GetInputEvent(RISING,3,2), 5000))
{/**/}
```

Wykonanie działania po 3.5 sekundach po krótkim naciśnięciu wejścia 7 modułu 5, z wykorzystaniem tajmeru 6, z możliwością restartu.

```
if (Timer(6, GetInputEvent(PRESS_SHORT,5,7), 3500, true))
{/**/}
```

Wykonanie działania po 8 sekundach po pojawieniu się flagi 9, z wykorzystaniem tajmeru 6.

```
if (Timer(6, GetFlag(9), 8000))
{/**/}
```

Okresowe wykonanie działania co 10 sekund, - tajmer 8.

```
if (Timer(8, true, 10000))
{/**/}
```

7.3. RapidPress

```
bool RapidPress (byte TimerID, bool condition, long SetTime,
int target);
```

Opis:

Funkcja detekcji wielokrotnego wystąpienia pewnego zdarzenia. Naprzykład potrójne naciśnięcie przycisku. Wykorzystuje systemę tajmerów.

***TimerID = 1...10** – ID tajmera. Standardowo ilość tajmerów jest ograniczona do 10.*
***condition** – zdarzenie które będzie detektowane*
***SetTime** – czas w ciągu którego musi się odbyć wielokrotne zdarzenie. Wyrażony w milisekundach.*
***target** – ilość zdarzeń które są odliczane.*

Przykłady:

Wykonanie działania po trzech krótkich naciśnięciach na wejściu 6 modułu 5 w ciągu 2 sekund, z wykorzystaniem tajmeru 4.

```
if (RapidPress(4,GetInputEvent(SHORT_PRESS,5,6),2000,3))
{/**/}
```

Wykonanie działania po pięciu wystąpieniach flagi 9 w ciągu 3 sekund, z wykorzystaniem tajmeru 7.

```
if (RapidPress(7,GetFlag(9),3000,5))
{/**/}
```

7.4. ResetController

```
void ResetController(byte ControllerID);
```

Opis:

Wysyła do wybranego modułu komendę o restarcie.

***ControllerID = 0...20** – ID modułu. Wpisanie wartości „0” wyślę komendę do wszystkich modułow w systemie.*

Przykłady:

Zrestartowanie modułu 3.

```
ResetController(3);
```

Zrestartowanie wszystkich modułów.

```
ResetController(0);
```

7.5. ReinstateOutputs

```
void ReinstateOutputs();
```

Opis:

*Wysyła do wszystkich modułów komendy o ustawieniu wyjść według ostatnich wiadomych wartości. Jest Automatycznie wysyłana za 10 sekund po restarcie modułów komendą **ResetController** lub przychodzącą komendą **AT+RST.***