OpenServo RS485

1.0

Wygenerowano przez Doxygen 1.8.1

Cz, 31 maj 2012 11:03:46

Spis treści

1 Dokumentacja bibliotek modułu IMU 1

2 Indeks grup 1

2.1 Moduły . . . 1

3 Indeks struktur danych 1

3.1 Struktury danych . . . 1

4 Indeks plików 1

4.1 Lista plików . . . 1

5 Dokumentacja grup 3

5.1 OpenServo . . . 3

5.1.1 Opis szczegółowy . . . 3

5.1.2 Dokumentacja definicji . . . 3

5.1.3 Dokumentacja funkcji . . . 4

5.2 Biblioteka RS485 . . . 8

5.2.1 Opis szczegółowy . . . 9

5.2.2 Dokumentacja definicji . . . 10

5.2.3 Dokumentacja typów wyliczanych . . . 11

5.2.4 Dokumentacja funkcji . . . 12

6 Dokumentacja struktur danych 20

6.1 Dokumentacja struktury Frame . . . 20

6.1.1 Opis szczegółowy . . . 20

6.1.2 Dokumentacja pól . . . 20

6.2 Dokumentacja struktury motion_key . . . 21

6.2.1 Opis szczegółowy . . . 21

6.2.2 Dokumentacja pól . . . 21

7 Dokumentacja plików 21

7.1 Dokumentacja pliku adc.c . . . 21

7.1.1 Dokumentacja definicji . . . 22

7.1.2 Dokumentacja funkcji . . . 23

7.1.3 Dokumentacja zmiennych . . . 25

7.2 Dokumentacja pliku adc.h . . . 25

7.2.1 Dokumentacja funkcji . . . 26

7.2.2 Dokumentacja zmiennych . . . 28

7.3 Dokumentacja pliku config.h . . . 28

7.3.1 Dokumentacja definicji . . . 29

7.4 Dokumentacja pliku curve.c . . . 30

7.4.1 Dokumentacja funkcji . . . 31
7.4.2 Dokumentacja zmiennych . . . 32
7.5 Dokumentacja pliku curve.h . . . 33
7.5.1 Dokumentacja funkcji . . . 33
7.5.2 Dokumentacja zmiennych . . . 34
7.6 Dokumentacja pliku eeprom.c . . . 35
7.6.1 Dokumentacja funkcji . . . 35
7.7 Dokumentacja pliku eeprom.h . . . 36
7.7.1 Dokumentacja definicji . . . 37
7.7.2 Dokumentacja funkcji . . . 37
7.8 Dokumentacja pliku estimator.c . . . 38
7.9 Dokumentacja pliku estimator.h . . . 38
7.9.1 Dokumentacja funkcji . . . 38
7.10 Dokumentacja pliku ipd.c . . . 39
7.11 Dokumentacja pliku ipd.h . . . 39
7.11.1 Dokumentacja funkcji . . . 39
7.12 Dokumentacja pliku macros.h . . . 39
7.12.1 Dokumentacja definicji . . . 42
7.12.2 Dokumentacja funkcji . . . 46
7.12.3 Dokumentacja zmiennych . . . 46
7.13 Dokumentacja pliku main.c . . . 53
7.14 Dokumentacja pliku math.c . . . 53
7.15 Dokumentacja pliku math.h . . . 53
7.16 Dokumentacja pliku motion.c . . . 53
7.16.1 Dokumentacja definicji typów . . . 54
7.16.2 Dokumentacja funkcji . . . 54
7.16.3 Dokumentacja zmiennych . . . 58
7.17 Dokumentacja pliku motion.h . . . 58
7.17.1 Dokumentacja definicji . . . 59
7.17.2 Dokumentacja funkcji . . . 59
7.17.3 Dokumentacja zmiennych . . . 63
7.18 Dokumentacja pliku openservo.h . . . 63
7.18.1 Dokumentacja definicji . . . 63
7.18.2 Dokumentacja definicji typów . . . 65
7.19 Dokumentacja pliku pid.c . . . 65
7.19.1 Dokumentacja definicji . . . 65
7.19.2 Dokumentacja funkcji . . . 66
7.20 Dokumentacja pliku pid.h . . . 68
7.20.1 Dokumentacja funkcji . . . 68
7.21 Dokumentacja pliku power.c . . . 70

7.21.1 Dokumentacja funkcji . . . 70
7.22 Dokumentacja pliku power.h . . . 71
7.22.1 Dokumentacja funkcji . . . 71
7.23 Dokumentacja pliku pulsectl.c . . . 72
7.24 Dokumentacja pliku pulsectl.h . . . 72
7.24.1 Dokumentacja funkcji . . . 72
7.25 Dokumentacja pliku pwm.c . . . 73
7.25.1 Dokumentacja definicji . . . 73
7.25.2 Dokumentacja funkcji . . . 73
7.26 Dokumentacja pliku pwm.h . . . 76
7.26.1 Dokumentacja funkcji . . . 77
7.27 Dokumentacja pliku registers.c . . . 80
7.27.1 Dokumentacja funkcji . . . 80
7.27.2 Dokumentacja zmiennych . . . 82
7.28 Dokumentacja pliku registers.h . . . 82
7.28.1 Dokumentacja definicji . . . 84
7.28.2 Dokumentacja funkcji . . . 90
7.28.3 Dokumentacja zmiennych . . . 92
7.29 Dokumentacja pliku regulator.c . . . 92
7.30 Dokumentacja pliku regulator.h . . . 92
7.30.1 Dokumentacja funkcji . . . 93
7.31 Dokumentacja pliku rs485.c . . . 93
7.31.1 Dokumentacja definicji . . . 95
7.31.2 Dokumentacja funkcji . . . 96
7.31.3 Dokumentacja zmiennych . . . 99
7.32 Dokumentacja pliku rs485.h . . . 100
7.33 Dokumentacja pliku seek.c . . . 102
7.34 Dokumentacja pliku seek.h . . . 102
7.35 Dokumentacja pliku timer.c . . . 102
7.36 Dokumentacja pliku timer.h . . . 102
7.37 Dokumentacja pliku twi.c . . . 102
7.37.1 Dokumentacja definicji . . . 103
7.37.2 Dokumentacja funkcji . . . 103
7.38 Dokumentacja pliku twi.h . . . 105
7.38.1 Dokumentacja definicji . . . 105
7.38.2 Dokumentacja funkcji . . . 106
7.39 Dokumentacja pliku watchdog.c . . . 108
7.39.1 Dokumentacja funkcji . . . 108
7.40 Dokumentacja pliku watchdog.h . . . 109
7.40.1 Dokumentacja funkcji . . . 109

1 Dokumentacja bibliotek modułu IMU

Opis:

Projekt kontrolera serwomechanizmów wyposażonego w interfejs RS485 oparty na projekce OpenServo

Autor

Jarosław Toliński

2 Indeks grup

2.1 Moduły

Tutaj znajduje się lista wszystkich grup:

OpenServo 3

Biblioteka RS485 8

3 Indeks struktur danych

3.1 Struktury danych

Tutaj znajdują się struktury danych wraz z ich krótkimi opisami:

Frame 20

4 Indeks plików

4.1 Lista plików

Tutaj znajduje się lista wszystkich plików z ich krótkimi opisami:

config.h 28

main.c 53

rs485.c 93

rs485.h 100

5 Dokumentacja grup

5.1 OpenServo

Plik konfiguracji OpenServa.

Definicje

- #define DEFAULT_PID_PGAIN 0x047C

Konfiguracja P w PID dla TowerPro SG5010.

- #define DEFAULT_PID_DGAIN 0x1000

  Konfiguracja D w PID dla TowerPro SG5010.
- #define DEFAULT_PID_IGAIN 0x0001

  Konfiguracja I w PID dla TowerPro SG5010.
- #define DEFAULT_PID_DEADBAND 0x01

  Martwa strefa TowerPro SG5010.
- #define DEFAULT_MIN_SEEK 0x0060

  Minimalna pozycja.
- #define DEFAULT_MAX_SEEK 0x03A0

  Maksymalna pozycja.

Funkcje

- int main (void)

5.1.1 Opis szczegółowy

Plik konfiguracji OpenServa. Plik main projektu OpenServo.

Dodano nastawy dla TowerPro SG5010.

Nota

Na podstawie projektu OpenServo

Autor

Michael P. Thompson mpthompson@gmail.com

Nota

W programie wykonano kilka zmian: usunięto obsługę TWI oraz dodano obsługę RS485. Odpowiednio zdefiniowano również porty
Na podstawie projektu OpenServo

Autor

Jaroslaw Toliński

5.1.2 Dokumentacja definicji

5.1.2.1 #define DEFAULT_MAX_SEEK 0x03A0

Maksymalna pozycja.

Definicja w linii 186 pliku config.h.

5.1.2.2 #define DEFAULT_MIN_SEEK 0x0060

Minimalna pozycja.

Definicja w linii 184 pliku config.h.

5.1.2.3 #define DEFAULT_PID_DEADBAND 0x01

Martwa strefa TowerPro SG5010.

Definicja w linii 179 pliku config.h.

5.1.2.4 #define DEFAULT_PID_DGAIN 0x1000

Konfiguracja D w PID dla TowerPro SG5010.

Definicja w linii 175 pliku config.h.

5.1.2.5 #define DEFAULT_PID_IGAIN 0x0001

Konfiguracja I w PID dla TowerPro SG5010.

Definicja w linii 177 pliku config.h.

5.1.2.6 #define DEFAULT_PID_PGAIN 0x047C

Konfiguracja P w PID dla TowerPro SG5010.

Definicja w linii 173 pliku config.h.

5.1.3 Dokumentacja funkcji

5.1.3.1 int main (void)

inicjalizacja pinu sterującego przepływem

inicjalizacja RS485

wykonywanie poleceń

Definicja w linii 104 pliku main.c.

Odwołuje się do RS485CMD(), UartInit() i UartInitRs485().

```
{
//  int i = 0;
    // Configure pins to the default states.
    config_pin_defaults();

    // Initialize the watchdog module.
    watchdog_init();

    // First, initialize registers that control servo operation.
    registers_init();

    // Initialize the PWM module.
    pwm_init();

    // Initialize the ADC module.
    adc_init();

#if ESTIMATOR_ENABLED
    // Initialize the state estimator module.
    estimator_init();
#endif

#if REGULATOR_MOTION_ENABLED
    // Initialize the regulator algorithm module.
    regulator_init();
#endif

#if PID_MOTION_ENABLED
    // Initialize the PID algorithm module.
    pid_init();
#endif

#if IPD_MOTION_ENABLED
    // Initialize the IPD algorithm module.
    ipd_init();
#endif

#if CURVE_MOTION_ENABLED
    // Initialize curve motion module.
    motion_init();
#endif

    // Initialize the power module.
    power_init();
```

```
#if PULSE_CONTROL_ENABLED
    pulse_control_init();
#endif

    UartInitRs485(&PORTD,PD2);
    UartInit();


    // Initialize the TWI slave module.
    //twi_slave_init(registers_read_byte(REG_TWI_ADDRESS));

    // Finally initialize the timer.
    timer_set(0);

    // Enable interrupts.
    sei();

    // Wait until initial position value is ready.
    while (!adc_position_value_is_ready());

#if CURVE_MOTION_ENABLED
    // Reset the curve motion with the current position of the servo.
    motion_reset(adc_get_position_value());
#endif

    // Set the initial seek position and velocity.
    registers_write_word(REG_SEEK_POSITION_HI, REG_SEEK_POSITION_LO,
      adc_get_position_value());
    registers_write_word(REG_SEEK_VELOCITY_HI, REG_SEEK_VELOCITY_LO, 0);

    // XXX Enable PWM and writing.  I do this for now to make development and
    // XXX tuning a bit easier.  Constantly manually setting these values to
    // XXX turn the servo on and write the gain values get's to be a pain.
    pwm_enable();
    registers_write_enable();

    // This is the main processing loop for the servo.  It basically looks
    // for new position, power or TWI commands to be processed.

    for (;;)
    {

        RS485CMD();




        // Is position value ready?
        if (adc_position_value_is_ready())
        {
            int16_t pwm;
            int16_t position;

#if PULSE_CONTROL_ENABLED
            // Give pulse control a chance to update the seek position.
            pulse_control_update();
#endif

#if CURVE_MOTION_ENABLED
            // Give the motion curve a chance to update the seek position and
       velocity.
            motion_next(10);
#endif

            // Get the new position value.
            position = (int16_t) adc_get_position_value();


#if ESTIMATOR_ENABLED
            // Estimate velocity.
            estimate_velocity(position);
#endif

#if PID_MOTION_ENABLED
            // Call the PID algorithm module to get a new PWM value.
            pwm = pid_position_to_pwm(position);
#endif

#if IPD_MOTION_ENABLED
            // Call the IPD algorithm module to get a new PWM value.
            pwm = ipd_position_to_pwm(position);
#endif

#if REGULATOR_MOTION_ENABLED
            // Call the state regulator algorithm module to get a new PWM
       value.
```

```
            pwm = regulator_position_to_pwm(position);
#endif

            // Update the servo movement as indicated by the PWM value.
            // Sanity checks are performed against the position value.
            pwm_update(position, pwm);
        }

        // Is a power value ready?
        if (adc_power_value_is_ready())
        {
            // Get the new power value.
            uint16_t power = adc_get_power_value();

            // Update the power value for reporting.
            power_update(power);
        }

        //UartPutChar('#');
  //      // Was a command recieved?
  //      if (twi_data_in_receive_buffer())
  //      {
  //          // Handle any TWI command.
  //          handle_twi_command();
  //      }

#if MAIN_MOTION_TEST_ENABLED
        // This code is in place for having the servo drive itself between
        // two positions to aid in the servo tuning process.  This code
        // should normally be disabled in config.h.
#if CURVE_MOTION_ENABLED
        if (motion_time_left() == 0)
        {
            registers_write_word(REG_CURVE_DELTA_HI, REG_CURVE_DELTA_LO, 2000);
            registers_write_word(REG_CURVE_POSITION_HI, REG_CURVE_POSITION_LO,
      0x0100);
            motion_append();
            registers_write_word(REG_CURVE_DELTA_HI, REG_CURVE_DELTA_LO, 1000);
            registers_write_word(REG_CURVE_POSITION_HI, REG_CURVE_POSITION_LO,
      0x0300);
            motion_append();
            registers_write_word(REG_CURVE_DELTA_HI, REG_CURVE_DELTA_LO, 2000);
            registers_write_word(REG_CURVE_POSITION_HI, REG_CURVE_POSITION_LO,
      0x0300);
            motion_append();
            registers_write_word(REG_CURVE_DELTA_HI, REG_CURVE_DELTA_LO, 1000);
            registers_write_word(REG_CURVE_POSITION_HI, REG_CURVE_POSITION_LO,
      0x0100);
            motion_append();
        }
#else
        {
            // Get the timer.
            uint16_t timer = timer_get();

            // Reset the timer if greater than 800.
            if (timer > 800) timer_set(0);

            // Look for specific events.
            if (timer == 0)
            {
                registers_write_word(REG_SEEK_HI, REG_SEEK_LO, 0x0100);
            }
            else if (timer == 400)
            {
                registers_write_word(REG_SEEK_HI, REG_SEEK_LO, 0x0300);
            }
        }
#endif
#endif
    }

    return 0;
}
```

5.2 Biblioteka RS485

Biblioteka RS485.

Struktury danych

- struct Frame

Definicje

- #define UART_DEFAULT_BAUD_RATE 19200
- #define UART_DEFAULT_DATA_BITS UART_DATA_BITS_8
- #define UART_DEFAULT_PARITY UART_PARITY_NONE
- #define UART_DEFAULT_STOP_BITS UART_STOP_BITS_1
- #define UART_DEFAULT_TRANSMIT_TIMEOUT_MILISECONDS 1000
- #define UART_DEFAULT_BUFFER_SIZE
- #define ReceiveNoError 0x00
- #define ReceiveParityE 0x01
- #define ReceiveFrameE 0x02
- #define ReceiveOverrunE 0x04
- #define FRAMECRCVALID 0x00
- #define FRAMECRCMISMATCH 0x01
- #define CMD_DIAG 0x01
- #define CMD_RESET 0x02
- #define CMD_READNORMALREG 0x03
- #define CMD_WRITENORMALREG 0x04
- #define CMD_NOTFOUND 0xF0

Wyliczenia

- enum UART_PARITY {
  UART_PARITY_NONE = 0x00,
  UART_PARITY_ODD = 0x30,
  UART_PARITY_EVEN = 0x20 }
- enum UART_DATA_BITS {
  UART_DATA_BITS_5 = 0x00,
  UART_DATA_BITS_6 = 0x02,
  UART_DATA_BITS_7 = 0x04,
  UART_DATA_BITS_8 = 0x06,
  UART_DATA_BITS_9 = 0x0E }
- enum UART_STOP_BITS {
  UART_STOP_BITS_1 = 0x00,
  UART_STOP_BITS_2 = 0x80 }

Funkcje

- bool UartInit (void)

  Inicjuje UART.
- bool UartSetBaud (uint32_t baudRate)

  Ustawia prędkość transmisji (BaudRate)
- void UartSetDataBits (UART_DATA_BITS dataBits)

  Ustawia ilość bitów danych.
- void UartSetParity (UART_PARITY parity)

Ustawia bity parzytości.

- void UartSetStopBits (UART_STOP_BITS stopBits)

  Ustawia ilość bitów stopu.
- bool UartSetBuffersSize (uint8_t size)
- void UartInitRs485 (volatile uint8_t *port, uint8_t pinConnectedToReDe)

  Inicjalizuje pin obsługujący kierunek przeływu danych na potrzeby half-duplexu RS485.
- uint16_t FrameCRC (volatile const Frame *f)

  Oblicza sumę kontrolną ramki.
- bool FrameCheckCRC (volatile Frame *f)

  Sprawdza poprawność sumy kontrolnej ramki.
- void FrameInit (volatile Frame *f, uint8_t a, uint8_t c, uint16_t d1, int16_t d2)

  Inicjuje strukturę ramki podanymi wartościami i oblicza jej sumę kontrolną
- void FrameCopy (volatile Frame *to, volatile Frame *from)

  Kopiuje ramkę
- bool SendFrame (Frame *f)

  Rozpoczyna wysyłanie ramki.
- bool GetFrame (volatile Frame *f)

  Sprawdza czy pobrano ramkę
- void RS485CMD ()

  Sprawdza czy przyszło jakieś polecenie, jeśli tak wykonuje je.

5.2.1 Opis szczegółowy

Biblioteka RS485.

```
#include <RS485.h>
```

Biblioteka służy wykonywaniu poleceń przesłanych przez interfejs RS485 (do portu UART mikrokontrolera). Docelowym urządzeniem biblioteki jest atmega168. Do komunikacji wykorzystywana jest ramka składająca się z 9 bajtów:

'<',device_address,cmd,data1MSB,data1LSB,data2MSB,data2LSB,CRCMSB,CRCLSB.

Program pominie wszystkie znaki aż do nadejścia bajtu '<'. Po jego odebraniu do kolejnych bajtów będą zapisywane kolejne odebrane znaki. W przypadku napotkania błędu odbioru (błąd zwracany przez uart) aktualna ramka zostanie porzucona i program rozpocznie pobieranie nowej ramki.

Po pobraniu ramki przechowywana jest ona w buforze (aktualnie jednoelementowym) oczekując na wykonie odbranej ramki (lub porzucenie jej). Po wywolaniu *RS485CMD()* program sprawdza czy zostala odebrana ramka, jesli tak, sprawdzana jest poprawnosc jej sumy CRC, następnie zgodność adresu z adresem urządzenia oraz poprawność komendy (czy istnieje).

Jeśli ramka przeszła weryfikację wykonywane jest zapisane w niej polecenie.

W przypadku błędu CRC lub niezgodności adresów ramka zostanie zignorowana.

W przypadku niepoprawnej komendy, program odeśle komunikat o błędzie zawierający błędne polecenie.

W przypadku braku gotowej ramki funkcja zakończy działanie.

Nota

Oparte na projekcie ZoSuperModiefied

Inicjalizacja i użycie

Do inicjalizacji układu wykorzytywana jest funkcja *UartInit()*:

```
UartInit();
```

Wykonanie odebranych komend:

```
RS485CMD();
```

Autor

Jaroslaw Toliński

5.2.2 Dokumentacja definicji

5.2.2.1 #define CMD_DIAG 0x01

Definicja w linii 95 pliku rs485.h.

5.2.2.2 #define CMD_NOTFOUND 0xF0

Definicja w linii 100 pliku rs485.h.

Odwołania w RS485CMD().

5.2.2.3 #define CMD_READNORMALREG 0x03

Definicja w linii 97 pliku rs485.h.

Odwołania w RS485CMD().

5.2.2.4 #define CMD_RESET 0x02

Definicja w linii 96 pliku rs485.h.

5.2.2.5 #define CMD_WRITENORMALREG 0x04

Definicja w linii 98 pliku rs485.h.

Odwołania w RS485CMD().

5.2.2.6 #define FRAMECRCMISMATCH 0x01

Definicja w linii 93 pliku rs485.h.

Odwołania w GetFrame().

5.2.2.7 #define FRAMECRCVALID 0x00

Definicja w linii 92 pliku rs485.h.

Odwołania w GetFrame() i RS485CMD().

5.2.2.8 #define ReceiveFrameE 0x02

Definicja w linii 90 pliku rs485.h.

Odwołania w ISR().

5.2.2.9 #define ReceiveNoError 0x00

Definicja w linii 88 pliku rs485.h.

Odwołania w ISR().

5.2.2.10 #define ReceiveOverrunE 0x04

Definicja w linii 91 pliku rs485.h.

Odwołania w ISR().

5.2.2.11 #define ReceiveParityE 0x01

Definicja w linii 89 pliku rs485.h.

Odwołania w ISR().

5.2.2.12 #define UART_DEFAULT_BAUD_RATE 19200

Definicja w linii 71 pliku rs485.h.

Odwołania w UartInit().

5.2.2.13 #define UART_DEFAULT_BUFFER_SIZE

Definicja w linii 76 pliku rs485.h.

5.2.2.14 #define UART_DEFAULT_DATA_BITS UART_DATA_BITS_8

Definicja w linii 72 pliku rs485.h.

Odwołania w UartInit().

5.2.2.15 #define UART_DEFAULT_PARITY UART_PARITY_NONE

Definicja w linii 73 pliku rs485.h.

Odwołania w UartInit().

5.2.2.16 #define UART_DEFAULT_STOP_BITS UART_STOP_BITS_1

Definicja w linii 74 pliku rs485.h.

Odwołania w UartInit().

5.2.2.17 #define UART_DEFAULT_TRANSMIT_TIMEOUT_MILISECONDS 1000

Definicja w linii 75 pliku rs485.h.

5.2.3 Dokumentacja typów wyliczanych

5.2.3.1 enum UART_DATA_BITS

Wartości wyliczeń:

UART_DATA_BITS_5

UART_DATA_BITS_6

UART_DATA_BITS_7

UART_DATA_BITS_8

UART_DATA_BITS_9

Definicja w linii 56 pliku rs485.h.

```
            {
        UART_DATA_BITS_5 = 0x00,
        UART_DATA_BITS_6 = 0x02,
        UART_DATA_BITS_7 = 0x04,
        UART_DATA_BITS_8 = 0x06,
        UART_DATA_BITS_9 = 0x0E
}UART_DATA_BITS;
```

5.2.3.2 enum UART_PARITY

Wartości wyliczeń:

UART_PARITY_NONE

UART_PARITY_ODD

UART_PARITY_EVEN

Definicja w linii 50 pliku rs485.h.

```
        {
        UART_PARITY_NONE = 0x00,
        UART_PARITY_ODD = 0x30,
        UART_PARITY_EVEN = 0x20
}UART_PARITY;
```

5.2.3.3 enum UART_STOP_BITS

Wartości wyliczeń:

UART_STOP_BITS_1

UART_STOP_BITS_2

Definicja w linii 64 pliku rs485.h.

```
        {
        UART_STOP_BITS_1=0x00,
        UART_STOP_BITS_2=0x80
}UART_STOP_BITS;
```

5.2.4 Dokumentacja funkcji

5.2.4.1 bool FrameCheckCRC (volatile Frame * f)

Sprawdza poprawność sumy kontrolnej ramki.

Parametry

f	wskaźnik na ramkę

Zwracane wartości

FALSE	gdy ramka ma niepoprawną sumę kontrolną
TRUE	gdy ramka

Definicja w linii 173 pliku rs485.c.

Odwołuje się do Frame::crc i FrameCRC().

Odwołania w GetFrame().

```
{
    /*if(f->crc==FrameCRC(f))
    {
        f->valid=FRAMECRCVALID;
        return TRUE;
    }
    else
    {
        f->valid|=FRAMECRCMISMATCH;
        return TRUE;
    }*/
    return (f->crc==FrameCRC(f))?TRUE:FALSE;
}
```

5.2.4.2 void FrameCopy (volatile Frame * to, volatile Frame * from)

Kopiuje ramkę

Parametry

to	ramka docelowa
from	kopiowana ramka

Definicja w linii 284 pliku rs485.c.

Odwołuje się do Frame::address, Frame::cmd, Frame::crc, Frame::data1 i Frame::data2.

Odwołania w GetFrame().

```
{
    f->address=from->address;
    f->cmd=from->cmd;
    f->data1=from->data1;
    f->data2=from->data2;
    f->crc=from->crc;
}
```

5.2.4.3 uint16_t FrameCRC (volatile const Frame * *f*)

Oblicza sumę kontrolną ramki.

Parametry

f	wskaźnik na ramkę

Zwraca

suma kontrolna ramki

Definicja w linii 157 pliku rs485.c.

Odwołuje się do Frame::address, Frame::cmd, Frame::data1 i Frame::data2.

Odwołania w FrameCheckCRC(), FrameInit() i GetFrame().

```
{
    uint16_t crc = 0xffff;
    crc = _crc16_update(crc,f->address);
    crc = _crc16_update(crc,f->cmd);

    //for(; i<f->length; ++i)
        //crc = _crc16_update(crc, f->data[i]);
    crc = _crc16_update(crc, f->data1>>8);
    crc = _crc16_update(crc, f->data1&0x00FF);
    crc = _crc16_update(crc, f->data2>>8);
    crc = _crc16_update(crc, f->data2&0x00FF);

    return crc;
}
```

5.2.4.4 void FrameInit (volatile Frame * *f*, uint8_t *a*, uint8_t *c*, uint16_t *d1*, int16_t *d2*)

Inicjuje strukturę ramki podanymi wartościami i oblicza jej sumę kontrolną

Parametry

f	inicjowana ramka
a	adres
c	polecenie (komenda)
d1	pierwszy bit danych
d2	drugi bit danych

Definicja w linii 276 pliku rs485.c.

Odwołuje się do Frame::address, Frame::cmd, Frame::crc, Frame::data1, Frame::data2 i FrameCRC().

Odwołania w RS485CMD(), SendFrame() i UartInit().

```
{
    f->address=a;
    f->cmd=c;
    f->data1=d1;
    f->data2=d2;
    f->crc=FrameCRC(f);
}
```

5.2.4.5 bool GetFrame (volatile Frame * f)

Sprawdza czy pobrano ramkę

Parametry

out	*f*	wskaźnik na ramkę do której zostanie zapisana odebrana ramka

Zwracane wartości

FALSE	oczekiwanie na dane (ramka f nie została zmodyfikowana)
TRUE	odebrano ramkę (ramka f zawiera odebrane dane)

Definicja w linii 393 pliku rs485.c.

Odwołuje się do Frame::crc, FrameCheckCRC(), FrameCopy(), FrameCRC(), FRAMECRCMISMATCH, FRAMECRCVALID, FrameReceived i Frame::valid.

Odwołania w RS485CMD().

```
{
    if(!FrameReceived)//if no frame  received
        return FALSE;//waiting for data , try later
    FrameCopy(f,&InFrame);//
      InFrame.address,InFrame.cmd,InFrame.data1,InFrame.data2);
    FrameCheckCRC(f);
    if(f->crc!=FrameCRC(f))
        f->valid=FRAMECRCMISMATCH;
    else
        f->valid=FRAMECRCVALID;
    FrameReceived=FALSE;//received frame has been taken care of
                        //wait for new data
    return TRUE;
}
```

5.2.4.6 void RS485CMD ( )

Sprawdza czy przyszło jakieś polecenie, jeśli tak wykonuje je.

Definicja w linii 408 pliku rs485.c.

Odwołuje się do Frame::address, Frame::cmd, CMD_NOTFOUND, CMD_READNORMALREG, CMD_WRITENORMALREG, Frame::data1, Frame::data2, FRAMECRCVALID, FrameInit(), GetFrame(), SendFrame() i Frame::valid.

Odwołania w main().

```
{
    if(GetFrame(&cmd))//wait for cmd (without crc errors)
                if(cmd.valid==FRAMECRCVALID&&cmd.
      address==registers_read_byte(REG_TWI_ADDRESS))
                {
                    switch(cmd.cmd)
                    {
                        case TWI_CMD_RESET:

                            // Reset the servo.
                            watchdog_hard_reset();

                            break;
```

```
            case TWI_CMD_PWM_ENABLE:

                // Enable PWM to the servo motor.
                pwm_enable();

                break;

            case TWI_CMD_PWM_DISABLE:

                // Disable PWM to the servo motor.
                pwm_disable();

                break;

            case TWI_CMD_WRITE_ENABLE:

                // Enable write to read/write protected registers.
                registers_write_enable();

                break;

            case TWI_CMD_WRITE_DISABLE:

                // Disable write to read/write protected registers.
                registers_write_disable();

                break;

            case TWI_CMD_REGISTERS_SAVE:

                // Save register values into EEPROM.
                eeprom_save_registers();

                break;

            case TWI_CMD_REGISTERS_RESTORE:

                // Restore register values into EEPROM.
                eeprom_restore_registers();

                break;

            case TWI_CMD_REGISTERS_DEFAULT:

                // Restore register values to factory defaults.
                registers_defaults();
                break;

            case TWI_CMD_EEPROM_ERASE:

                // Erase the EEPROM.
                eeprom_erase();

                break;

            case TWI_CMD_VOLTAGE_READ:

                // Request a voltage reading.
                adc_read_voltage();

                break;

#if CURVE_MOTION_ENABLED
            case TWI_CMD_CURVE_MOTION_ENABLE:

                // Enable curve motion handling.
                motion_enable();

                break;

            case TWI_CMD_CURVE_MOTION_DISABLE:

                // Disable curve motion handling.
                motion_disable();

                break;

            case TWI_CMD_CURVE_MOTION_RESET:

                // Reset the motion to the current position.
                motion_reset(adc_get_position_value());

                break;

            case TWI_CMD_CURVE_MOTION_APPEND:
```

```
                        // Append motion curve data stored in the
     registers.
                        motion_append();

                        break;
                #endif
                        case CMD_READNORMALREG:
                            if( (cmd.data1>>8)<=
    MAX_WRITE_PROTECT_REGISTER)
                            {
                                FrameInit(&response,0x00,cmd
    .cmd,cmd.data1,registers_read_word(cmd.data1>>8, cmd.data1
    &0x00FF));
                                SendFrame(&response);
                            }
                            break;
                        case CMD_WRITENORMALREG:
                            if((cmd.data1>>8)<=
    MAX_WRITE_PROTECT_REGISTER)
                                registers_write_word(cmd.data1>>8, cmd
    .data1&0x00FF, cmd.data2);
                            break;
                        default:
                            FrameInit(&response,0x00,
    CMD_NOTFOUND,cmd.cmd,0);
                            SendFrame(&response);
                            // Ignore unknown command.
                            break;
                    }
                }
                //else SendFrame(&erro);
}
```

5.2.4.7 bool SendFrame (Frame * f)

Rozpoczyna wysyłanie ramki.

Parametry

f	wskaźnik na wysyłaną ramkę

Zwracane wartości

FALSE	jeśli wcześniejsza ramka nie została jeszcze wysłana
TRUE	jeśli rozpoczęto wysyłanie ramki

Definicja w linii 377 pliku rs485.c.

Odwołuje się do Frame::address, Frame::cmd, Frame::data1, Frame::data2, FrameInit(), FrameOverflows, FrameSend i UartStartTx().

Odwołania w RS485CMD() i UartInit().

```
{
    if(FrameSend)//if line busy
    {
        ++FrameOverflows;
        return FALSE;//line busy
    }
    //FrameSend=FALSE;

    FrameInit(&OutFrame,f->address,f->cmd,f->data1
      ,f->data2);
    FrameSend=TRUE;
    UartStartTx();
    return TRUE;//frame going out;

}
```

5.2.4.8 bool UartInit (void)

Inicjuje UART.

Zwracane wartości

FALSE	w przypadku błedu
TRUE	w przeciwnym wypadku

Definicja w linii 67 pliku rs485.c.

Odwołuje się do FrameInit(), SendFrame(), UART_DEFAULT_BAUD_RATE, UART_DEFAULT_DATA_BITS, UART_DEFAULT_PARITY, UART_DEFAULT_STOP_BITS, UartSetBaud(), UartSetDataBits(), UartSetParity() i UartSetStopBits().

Odwołania w main().

```
{
    FrameInit(&erro,0x41,0x44,0x4352,0x4345);
    FrameInit(&response,00,0x53,0x5441,0x5254);
    SendFrame(&response);
    UCSR0A |= _BV(U2X0);  //dual speed
               //double speed mode
    UCSR0A &= ~_BV(MPCM0);//multiprocessor mode off
                          //no multiprocessor
    UCSR0C &= ~(_BV(UMSEL01)|_BV(UMSEL00));//asynchronous USART mode

    UartSetBaud(UART_DEFAULT_BAUD_RATE);
    UartSetDataBits(UART_DEFAULT_DATA_BITS
      );
    UartSetParity(UART_DEFAULT_PARITY);
    UartSetStopBits(UART_DEFAULT_STOP_BITS
      );

    UCSR0B |= _BV(RXEN0)|_BV(RXCIE0)  ;    //enable receive and receive
       interrupt, transmit and transmit interrupt are enabled when data for transmission are
       present.
    UCSR0B &= ~_BV(TXEN0);                                  //be sure the tx is
       disabled.
    DDRD &= ~_BV(PD1);                                              //put tx
       pin in high impendance mode in order to allow others to communicate

    sei();

   return TRUE;
}
```

5.2.4.9 void UartInitRs485 (volatile uint8_t * *port*, uint8_t *pinConnectedToReDe*)

Inicjalizuje pin obsługujący kierunek przeływu danych na potrzeby half-duplexu RS485.

Parametry

port	port na którym znajduje się używany pin
pinConnectedToReDe	numer używanego pinu (liczony od 0)

Definicja w linii 121 pliku rs485.c.

Odwołania w main().

```
{
        Rs485Used = TRUE;
        Rs485ReDePort = port;
        Rs485ReDePin = pinConnectedToReDe;

        *(port-1) |= _BV(pinConnectedToReDe);   //configure DDR register
        *port &= ~_BV(pinConnectedToReDe);      //configure PORT register
                                                //reset reDePin -> receive mode
        UCSR0B |= _BV(TXCIE0);                  // Enable Transmit Complete
       Interrupt
}
```

5.2.4.10 bool UartSetBaud (uint32_t *baudRate*)

Ustawia prędkość transmisji (BaudRate)

Parametry

baudRate	docelowa prędkość transmisji (BaudRate)

Zwracane wartości

FALSE	w przypadku błedu
TRUE	w przeciwnym wypadku

Definicja w linii 90 pliku rs485.c.

Odwołania w UartInit().

```
{
        uint32_t ubrrReg = 0;

        //configure baud rate
        ubrrReg = (F_CPU/baudRate/8 - 1);
        if( ( ubrrReg > 65535) || ( ubrrReg < 1 ) )
                return FALSE;

        UBRR0H = (uint8_t)((ubrrReg >> 8) & 0x00FF);                //baud rate
       divisor high byte
        UBRR0L = (uint8_t)(ubrrReg & 0x00FF);                               //
      baud rate divisor low byte

        return TRUE;
}
```

5.2.4.11 bool UartSetBuffersSize (uint8_t *size*)

5.2.4.12 void UartSetDataBits (UART_DATA_BITS *dataBits*) [inline]

Ustawia ilość bitów danych.

Parametry

dataBits	ilość bitów danych (wartość musi należeć do UART_DATA_BITS)

Zobacz również

UART_DATA_BITS

Definicja w linii 105 pliku rs485.c.

Odwołuje się do DATA_BITS_MASK_UCSR0B i DATA_BITS_MASK_UCSR0C.

Odwołania w UartInit().

```
{
        UCSR0C = (UCSR0C & ~DATA_BITS_MASK_UCSR0C) | (
      dataBits & DATA_BITS_MASK_UCSR0C);
        UCSR0B = (UCSR0B & ~DATA_BITS_MASK_UCSR0B) | ((
      dataBits>>1) & DATA_BITS_MASK_UCSR0B);
}
```

5.2.4.13 void UartSetParity (UART_PARITY *parity*) [inline]

Ustawia bity parzytości.

Parametry

parity	rodzaj bitu parzystości (wartość musi należeć do UART_PARITY)

Zobacz również

UART_PARITY

Definicja w linii 111 pliku rs485.c.

Odwołuje się do PARITY_BITS_MASK.

Odwołania w UartInit().

```
{
        UCSR0C = (UCSR0C & ~PARITY_BITS_MASK)|parity;
}
```

5.2.4.14 void UartSetStopBits (UART_STOP_BITS *stopBits*) [inline]

Ustawia ilość bitów stopu.

Parametry

stopBits	ilość bitów stopu (wartość musi należeć do UART_STOP_BITS)

Zobacz również

UART_STOP_BITS

Definicja w linii 116 pliku rs485.c.

Odwołuje się do STOP_BITS_MASK.

Odwołania w UartInit().

```
{
        UCSR0C = (UCSR0C & ~STOP_BITS_MASK)|stopBits;
}
```

6 Dokumentacja struktur danych

6.1 Dokumentacja struktury Frame

`#include <rs485.h>`

Pola danych

- uint8_t address
- uint8_t cmd
- uint16_t data1
- uint16_t data2
- uint16_t crc
- uint8_t valid

6.1.1 Opis szczegółowy

Definicja w linii 78 pliku rs485.h.

6.1.2 Dokumentacja pól

6.1.2.1 uint8_t address

Definicja w linii 80 pliku rs485.h.

Odwołania w FrameCopy(), FrameCRC(), FrameInit(), RS485CMD(), SendFrame(), UartGetFrameISR() i UartPutFrameISR().

6.1.2.2 uint8_t cmd

Definicja w linii 81 pliku rs485.h.

Odwołania w FrameCopy(), FrameCRC(), FrameInit(), RS485CMD(), SendFrame(), UartGetFrameISR() i UartPutFrameISR().

6.1.2.3 uint16_t crc

Definicja w linii 84 pliku rs485.h.

Odwołania w FrameCheckCRC(), FrameCopy(), FrameInit(), GetFrame(), UartGetFrameISR() i UartPutFrameISR().

6.1.2.4 uint16_t data1

Definicja w linii 82 pliku rs485.h.

Odwołania w FrameCopy(), FrameCRC(), FrameInit(), RS485CMD(), SendFrame(), UartGetFrameISR() i UartPutFrameISR().

6.1.2.5 uint16_t data2

Definicja w linii 83 pliku rs485.h.

Odwołania w FrameCopy(), FrameCRC(), FrameInit(), RS485CMD(), SendFrame(), UartGetFrameISR() i UartPutFrameISR().

6.1.2.6 uint8_t valid

Definicja w linii 85 pliku rs485.h.

Odwołania w GetFrame() i RS485CMD().

Dokumentacja dla tej struktury została wygenerowana z pliku:

- rs485.h

7 Dokumentacja plików

7.1 Dokumentacja pliku config.h

Definicje

- #define TWI_CHECKED_ENABLED 0
- #define PID_MOTION_ENABLED 1
- #define IPD_MOTION_ENABLED 0
- #define REGULATOR_MOTION_ENABLED 0
- #define ESTIMATOR_ENABLED (REGULATOR_MOTION_ENABLED)
- #define FIXED_MATH_ENABLED (ESTIMATOR_ENABLED || REGULATOR_ENABLED)
- #define CURVE_MOTION_ENABLED 1
- #define MAIN_MOTION_TEST_ENABLED 0
- #define PULSE_CONTROL_ENABLED 0
- #define SWAP_PWM_DIRECTION_ENABLED 0
- #define HARDWARE_TYPE_UNKNOWN 0
- #define HARDWARE_TYPE_FUTABA_S3003 1
- #define HARDWARE_TYPE_HITEC_HS_311 2
- #define HARDWARE_TYPE_HITEC_HS_475HB 3
- #define HARDWARE_TYPE_TOWERPRO_SG5010 4
- #define HARDWARE_TYPE HARDWARE_TYPE_TOWERPRO_SG5010
- #define DEFAULT_PID_PGAIN 0x047C

  Konfiguracja P w PID dla TowerPro SG5010.
- #define DEFAULT_PID_DGAIN 0x1000

  Konfiguracja D w PID dla TowerPro SG5010.
- #define DEFAULT_PID_IGAIN 0x0001

  Konfiguracja I w PID dla TowerPro SG5010.
- #define DEFAULT_PID_DEADBAND 0x01

  Martwa strefa TowerPro SG5010.
- #define DEFAULT_MIN_SEEK 0x0060

  Minimalna pozycja.
- #define DEFAULT_MAX_SEEK 0x03A0

  Maksymalna pozycja.
- #define DEFAULT_PWM_FREQ_DIVIDER 0x0010

7.1.1 Dokumentacja definicji

7.1.1.1 #define CURVE_MOTION_ENABLED 1

Definicja w linii 99 pliku config.h.

7.1.1.2 #define DEFAULT_PWM_FREQ_DIVIDER 0x0010

Definicja w linii 189 pliku config.h.

7.1.1.3 #define ESTIMATOR_ENABLED (REGULATOR_MOTION_ENABLED)

Definicja w linii 88 pliku config.h.

7.1.1.4 #define FIXED_MATH_ENABLED (ESTIMATOR_ENABLED || REGULATOR_ENABLED)

Definicja w linii 93 pliku config.h.

7.1.1.5 #define HARDWARE_TYPE HARDWARE_TYPE_TOWERPRO_SG5010

Definicja w linii 147 pliku config.h.

7.1.1.6 #define HARDWARE_TYPE_FUTABA_S3003 1

Definicja w linii 139 pliku config.h.

7.1.1.7 #define HARDWARE_TYPE_HITEC_HS_311 2

Definicja w linii 140 pliku config.h.

7.1.1.8 #define HARDWARE_TYPE_HITEC_HS_475HB 3

Definicja w linii 141 pliku config.h.

7.1.1.9 #define HARDWARE_TYPE_TOWERPRO_SG5010 4

Definicja w linii 142 pliku config.h.

7.1.1.10 #define HARDWARE_TYPE_UNKNOWN 0

Definicja w linii 138 pliku config.h.

7.1.1.11 #define IPD_MOTION_ENABLED 0

Definicja w linii 70 pliku config.h.

7.1.1.12 #define MAIN_MOTION_TEST_ENABLED 0

Definicja w linii 105 pliku config.h.

7.1.1.13 #define PID_MOTION_ENABLED 1

Definicja w linii 60 pliku config.h.

7.1.1.14 #define PULSE_CONTROL_ENABLED 0

Definicja w linii 111 pliku config.h.

7.1.1.15 #define REGULATOR_MOTION_ENABLED 0

Definicja w linii 80 pliku config.h.

7.1.1.16 #define SWAP_PWM_DIRECTION_ENABLED 0

Definicja w linii 118 pliku config.h.

7.1.1.17 #define TWI_CHECKED_ENABLED 0

Definicja w linii 52 pliku config.h.

7.2 Dokumentacja pliku main.c

```
#include <inttypes.h>
#include <avr/interrupt.h>
#include <avr/io.h>
#include "openservo.h"
#include "config.h"
#include "adc.h"
#include "eeprom.h"
#include "estimator.h"
#include "motion.h"
#include "pid.h"
#include "power.h"
#include "pwm.h"
#include "seek.h"
#include "timer.h"
#include "rs485.h"
#include "watchdog.h"
#include "registers.h"
```

Funkcje

- int main (void)

7.3 Dokumentacja pliku rs485.c

```
#include <avr/io.h>
#include <avr/interrupt.h>
#include <util/crc16.h>
#include <util/delay.h>
#include "rs485.h"
#include "twi.h"
#include "watchdog.h"
#include "registers.h"
```

Definicje

- #define MAX_uint8_t 255
- #define MAX_uint16_t 65535
- #define MAX_U32 4294967295
- #define MIN_int8_t -128
- #define MAX_int8_t 127
- #define MIN_int16_t -32768
- #define MAX_int16_t 32767
- #define MIN_S32 -2147483648
- #define MAX_S32 2147483647
- #define DATA_BITS_MASK_UCSR0C 0x06
- #define DATA_BITS_MASK_UCSR0B 0x04
- #define PARITY_BITS_MASK 0x30
- #define STOP_BITS_MASK 0x80
- #define FRAMELENGTH 9
- #define WAITONSTART 0
- #define WAITONADDRESS 1

- #define WAITONCMD 2
- #define WAITONDATA1 3
- #define WAITONDATA2 4
- #define WAITONDATA3 5
- #define WAITONDATA4 6
- #define WAITONCRC1 7
- #define WAITONCRC2 8

Funkcje

- bool UartInit (void)

  Inicjuje UART.
- bool UartSetBaud (uint32_t baudRate)

  Ustawia prędkość transmisji (BaudRate)
- void UartSetDataBits (UART_DATA_BITS dataBits)

  Ustawia ilość bitów danych.
- void UartSetParity (UART_PARITY parity)

  Ustawia bity parzytości.
- void UartSetStopBits (UART_STOP_BITS stopBits)

  Ustawia ilość bitów stopu.
- void UartInitRs485 (volatile uint8_t ∗port, uint8_t pinConnectedToReDe)

  Inicjalizuje pin obsługujący kierunek przeływu danych na potrzeby half-duplexu RS485.
- void UartStartTx (void)
- void UartRxFlush (void)
- uint16_t FrameCRC (volatile const Frame ∗f)

  Oblicza sumę kontrolną ramki.
- bool FrameCheckCRC (volatile Frame ∗f)

  Sprawdza poprawność sumy kontrolnej ramki.
- uint8_t UartGetFrameISR (volatile Frame ∗f, volatile uint8_t byte)
- uint8_t UartPutFrameISR (volatile Frame ∗f)
- void FrameInit (volatile Frame ∗f, uint8_t a, uint8_t c, uint16_t d1, int16_t d2)

  Inicjuje strukturę ramki podanymi wartościami i oblicza jej sumę kontrolną
- void FrameCopy (volatile Frame ∗f, volatile Frame ∗from)

  Kopiuje ramkę
- ISR (USART_RX_vect)
- ISR (USART_UDRE_vect)
- ISR (USART_TX_vect)
- bool SendFrame (Frame ∗f)

  Rozpoczyna wysyłanie ramki.
- bool GetFrame (volatile Frame ∗f)

  Sprawdza czy pobrano ramkę
- void RS485CMD ()

  Sprawdza czy przyszło jakieś polecenie, jeśli tak wykonuje je.

Zmienne

- volatile uint8_t FrameBytesCount = 0
- volatile uint8_t FrameSendBytesCount = 0
- volatile bool FrameReceived = FALSE
- volatile bool FrameSend = FALSE
- volatile uint16_t FrameErrors = 0
- volatile uint16_t FrameOverflows = 0

- volatile uint16_t ReceiveErrors = 0
- volatile Frame InFrame
- volatile Frame OutFrame
- Frame response
- Frame cmd
- Frame erro

7.3.1 Dokumentacja definicji

7.3.1.1 #define DATA_BITS_MASK_UCSR0B 0x04

Definicja w linii 26 pliku rs485.c.

Odwołania w UartSetDataBits().

7.3.1.2 #define DATA_BITS_MASK_UCSR0C 0x06

Definicja w linii 25 pliku rs485.c.

Odwołania w UartSetDataBits().

7.3.1.3 #define FRAMELENGTH 9

Definicja w linii 36 pliku rs485.c.

7.3.1.4 #define MAX_int16_t 32767

Definicja w linii 21 pliku rs485.c.

7.3.1.5 #define MAX_int8_t 127

Definicja w linii 19 pliku rs485.c.

7.3.1.6 #define MAX_S32 2147483647

Definicja w linii 23 pliku rs485.c.

7.3.1.7 #define MAX_U32 4294967295

Definicja w linii 14 pliku rs485.c.

7.3.1.8 #define MAX_uint16_t 65535

Definicja w linii 13 pliku rs485.c.

7.3.1.9 #define MAX_uint8_t 255

Definicja w linii 12 pliku rs485.c.

7.3.1.10 #define MIN_int16_t -32768

Definicja w linii 20 pliku rs485.c.

7.3.1.11 #define MIN_int8_t -128

Definicja w linii 18 pliku rs485.c.

7.3.1.12 #define MIN_S32 -2147483648

Definicja w linii 22 pliku rs485.c.

7.3.1.13 #define PARITY_BITS_MASK 0x30

Definicja w linii 27 pliku rs485.c.

Odwołania w UartSetParity().

7.3.1.14 #define STOP_BITS_MASK 0x80

Definicja w linii 28 pliku rs485.c.

Odwołania w UartSetStopBits().

7.3.1.15 #define WAITONADDRESS 1

Definicja w linii 38 pliku rs485.c.

Odwołania w UartGetFrameISR() i UartPutFrameISR().

7.3.1.16 #define WAITONCMD 2

Definicja w linii 39 pliku rs485.c.

Odwołania w UartGetFrameISR() i UartPutFrameISR().

7.3.1.17 #define WAITONCRC1 7

Definicja w linii 44 pliku rs485.c.

Odwołania w UartGetFrameISR() i UartPutFrameISR().

7.3.1.18 #define WAITONCRC2 8

Definicja w linii 45 pliku rs485.c.

Odwołania w UartGetFrameISR() i UartPutFrameISR().

7.3.1.19 #define WAITONDATA1 3

Definicja w linii 40 pliku rs485.c.

Odwołania w UartGetFrameISR() i UartPutFrameISR().

7.3.1.20 #define WAITONDATA2 4

Definicja w linii 41 pliku rs485.c.

Odwołania w UartGetFrameISR() i UartPutFrameISR().

7.3.1.21 #define WAITONDATA3 5

Definicja w linii 42 pliku rs485.c.

Odwołania w UartGetFrameISR() i UartPutFrameISR().

7.3.1.22 #define WAITONDATA4 6

Definicja w linii 43 pliku rs485.c.

Odwołania w UartGetFrameISR() i UartPutFrameISR().

7.3.1.23 #define WAITONSTART 0

Definicja w linii 37 pliku rs485.c.

Odwołania w UartGetFrameISR() i UartPutFrameISR().

7.3.2 Dokumentacja funkcji

7.3.2.1 ISR (USART_RX_vect)

Definicja w linii 293 pliku rs485.c.

Odwołuje się do FrameBytesCount, FrameErrors, FrameOverflows, FrameReceived, ReceiveErrors, ReceiveFrameE, ReceiveNoError, ReceiveOverrunE, ReceiveParityE i UartGetFrameISR().

```
{
    uint8_t volatile c;
    uint8_t errors=ReceiveNoError;
    if ((UCSR0A & _BV(FE0)) != 0x00)  //is there a frame error?
            errors|=ReceiveFrameE;

    if ((UCSR0A & _BV(UPE0)) != 0x00) //is there a parity error?
            errors|=ReceiveParityE;

    if ( (UCSR0A & _BV(DOR0)) != 0x00 ) //Is there data overrun?
            errors|=ReceiveOverrunE;

    //Above three bits are cleared automatically when UDR0 is read.
                                                                                //
    c  = UDR0;
    if(errors==ReceiveNoError)
        if(!FrameReceived)
        {
            if(!UartGetFrameISR(&InFrame,c))
                ++FrameErrors;
        }
        else
            ++FrameOverflows;
    else
    {
        ++ReceiveErrors;
        //if(FrameBytesCount<7)//if only crc left
        FrameBytesCount=0;//drop frame, start receiving new
    }

}
```

7.3.2.2 ISR (USART_UDRE_vect)

Definicja w linii 325 pliku rs485.c.

Odwołuje się do FrameSend i UartPutFrameISR().

```
{

    //PORTB&=~_BV(PORTB3);//|_BV(PORTB5));//wlacz PB3 - kierunek wysylanie
    //PORTB|=_BV(PORTB4);//wlacz PB4 - kierunek wysylanie
    //if(!BufferIsEmpty(&TxBuffer))
    //PORTB&=~_BV(PORTB4);//wlacz PB4 - kierunek odbieranie
    //*Rs485ReDePort |= _BV(Rs485ReDePin);
    if(FrameSend)
    {


            //_delay_us(5);

            UCSR0B |= _BV(TXEN0);                   //enable transmitter we are
       about to send data on the bus
            //UCSR0B |= _BV(TXCIE0);

            //PORTB&=~_BV(PORTB4);//wlacz PB4 - kierunek odbieranie
            //_delay_us(10);
            volatile uint8_t c=UartPutFrameISR(&OutFrame
      );
            //if(Rs485Used)
            UDR0=c;
            //_delay_us(5);
            //UDR0 =BufferPopISR(&TxBuffer);
    }
    else
    {
        UCSR0B&=~_BV(UDRIE0);// Buffer empty, Disable Tx interrupts

    }
    //*Rs485ReDePort &= ~_BV(Rs485ReDePin);
    //PORTB|=_BV(PORTB4);//wlacz PB4 - kierunek wysylanie
```

```
    //PORTB|=_BV(PORTB3);//|_BV(PORTB5);//wylacz PB3 - kierunek odbieranie
    //PORTB|=_BV(PORTB4);//wylacz PB3 - kierunek odbieranie
}
```

7.3.2.3 ISR (USART_TX_vect)

Definicja w linii 361 pliku rs485.c.

```
{

        UCSR0B &= ~_BV(TXEN0);                          //disable transmitter,
       allow other nodes on the uart bus to communicate
        DDRD &= ~_BV(PD1);                                      //put tx pin in
       high impendance mode in order to allow others to communicate


        *Rs485ReDePort &= ~_BV(Rs485ReDePin);   // Clear RS485 Pin for receive
       mode
        //_delay_us(10);
        //_delay_us(10);
        PORTB|=_BV(PORTB4);//wlacz PB4 - kierunek wysylanie
        UCSR0B &=~_BV(TXCIE0);                                          // Disable
       trasnmit complete interrupt
        UCSR0B |= _BV(RXCIE0);
}
```

7.3.2.4 uint8_t UartGetFrameISR (volatile Frame * *f*, volatile uint8_t *byte*)

Definicja w linii 196 pliku rs485.c.

Odwołuje się do Frame::address, Frame::cmd, Frame::crc, Frame::data1, Frame::data2, FrameBytesCount, FrameReceived, WAITONADDRESS, WAITONCMD, WAITONCRC1, WAITONCRC2, WAITONDATA1, WAITONDATA2, WAITONDATA3, WAITONDATA4 i WAITONSTART.

Odwołania w ISR().

```
{
    if(!FrameReceived)
    {
        switch(FrameBytesCount)
        {
            case WAITONSTART:
                if(byte!='<')
                    return FALSE;
                break;
            case WAITONADDRESS:
                f->address=byte;
                break;
            case WAITONCMD:
                f->cmd=byte;
                break;
            case WAITONDATA1:
                f->data1=(uint16_t)byte<<8;
                break;
            case WAITONDATA2:
                f->data1+=byte& 0xFF;
                break;
            case WAITONDATA3:
                f->data2=(uint16_t)byte<<8;
                break;
            case WAITONDATA4:
                f->data2+=byte& 0xFF;
                break;
            case WAITONCRC1:
                f->crc=(uint16_t)byte<<8;
                break;
            case WAITONCRC2:
            {
                f->crc+=byte& 0xFF;
                FrameReceived=TRUE;
                FrameBytesCount=0;
                return TRUE;
            }
                break;
        }
        ++FrameBytesCount;
    }
    return TRUE;
}
```

7.3.2.5 uint8_t UartPutFrameISR (volatile Frame * f)

Definicja w linii 241 pliku rs485.c.

Odwołuje się do Frame::address, Frame::cmd, Frame::crc, Frame::data1, Frame::data2, FrameSend, FrameSendBytesCount, WAITONADDRESS, WAITONCMD, WAITONCRC1, WAITONCRC2, WAITONDATA1, WAITONDATA2, WAITONDATA3, WAITONDATA4 i WAITONSTART.

Odwołania w ISR().

```
{
    //if(FrameSend)
    //{
        //++
        switch(FrameSendBytesCount)
        {
            case WAITONSTART:
                return '<';//send frame start sign
            case WAITONADDRESS:
                return f->address;
            case WAITONCMD:
                return f->cmd;
            case WAITONDATA1:
                return (f->data1)>>8;
            case WAITONDATA2:
                return (f->data1)& 0xFF;
            case WAITONDATA3:
                return (f->data2)>>8;
            case WAITONDATA4:
                return (f->data2)& 0xFF;
            case WAITONCRC1:
                return (f->crc)>>8;
            case WAITONCRC2:
            {
                FrameSend=FALSE;//clear "sending frame" flag, ready
       for new frame to send
                FrameSendBytesCount=0;
                return (f->crc)& 0xFF;
            }
            //default:
            //  return FrameSend=FALSE;
        }
        ++FrameSendBytesCount;
}
```

7.3.2.6 void UartRxFlush (void)

Definicja w linii 148 pliku rs485.c.

```
{
    uint8_t  dummy;

    while (bit_is_set(UCSR0A, RXC0))
         dummy = UDR0;

}
```

7.3.2.7 void UartStartTx (void)

Definicja w linii 134 pliku rs485.c.

Odwołuje się do FrameSend.

Odwołania w SendFrame().

```
{
    enterCritical();
    PORTB&=~_BV(PORTB4);//wlacz PB4 - kierunek odbieranie
    *Rs485ReDePort |= _BV(Rs485ReDePin);
    if(FrameSend)                                    // See if
       this is the first character
    {
        UCSR0B |= _BV(UDRIE0);                            // Yes, Enable
       Tx interrupts
        UCSR0B |=_BV(TXCIE0);                               // Disable
       trasnmit complete interrupt
        UCSR0B &=~ _BV(RXCIE0);
```

```
    }
    exitCritical();
}
```

7.3.3 Dokumentacja zmiennych

7.3.3.1 Frame cmd

Definicja w linii 64 pliku rs485.c.

7.3.3.2 Frame erro

Definicja w linii 65 pliku rs485.c.

7.3.3.3 volatile uint8_t FrameBytesCount = 0

Definicja w linii 51 pliku rs485.c.

Odwołania w ISR() i UartGetFrameISR().

7.3.3.4 volatile uint16_t FrameErrors = 0

Definicja w linii 56 pliku rs485.c.

Odwołania w ISR().

7.3.3.5 volatile uint16_t FrameOverflows = 0

Definicja w linii 57 pliku rs485.c.

Odwołania w ISR() i SendFrame().

7.3.3.6 volatile bool FrameReceived = FALSE

Definicja w linii 53 pliku rs485.c.

Odwołania w GetFrame(), ISR() i UartGetFrameISR().

7.3.3.7 volatile bool FrameSend = FALSE

Definicja w linii 54 pliku rs485.c.

Odwołania w ISR(), SendFrame(), UartPutFrameISR() i UartStartTx().

7.3.3.8 volatile uint8_t FrameSendBytesCount = 0

Definicja w linii 52 pliku rs485.c.

Odwołania w UartPutFrameISR().

7.3.3.9 volatile Frame InFrame

Definicja w linii 60 pliku rs485.c.

7.3.3.10 volatile Frame OutFrame

Definicja w linii 61 pliku rs485.c.

7.3.3.11 volatile uint16_t ReceiveErrors = 0

Definicja w linii 58 pliku rs485.c.

Odwołania w ISR().

7.3.3.12 Frame response

Definicja w linii 63 pliku rs485.c.

7.4 Dokumentacja pliku rs485.h

```
#include <stdint.h>
#include "openservo.h"
#include <avr/io.h>
#include <avr/interrupt.h>
```

Struktury danych

- struct Frame

Definicje

- #define UART_DEFAULT_BAUD_RATE 19200
- #define UART_DEFAULT_DATA_BITS UART_DATA_BITS_8
- #define UART_DEFAULT_PARITY UART_PARITY_NONE
- #define UART_DEFAULT_STOP_BITS UART_STOP_BITS_1
- #define UART_DEFAULT_TRANSMIT_TIMEOUT_MILISECONDS 1000
- #define UART_DEFAULT_BUFFER_SIZE
- #define ReceiveNoError 0x00
- #define ReceiveParityE 0x01
- #define ReceiveFrameE 0x02
- #define ReceiveOverrunE 0x04
- #define FRAMECRCVALID 0x00
- #define FRAMECRCMISMATCH 0x01
- #define CMD_DIAG 0x01
- #define CMD_RESET 0x02
- #define CMD_READNORMALREG 0x03
- #define CMD_WRITENORMALREG 0x04
- #define CMD_NOTFOUND 0xF0

Wyliczenia

- enum UART_PARITY {
  UART_PARITY_NONE = 0x00,
  UART_PARITY_ODD = 0x30,
  UART_PARITY_EVEN = 0x20 }
- enum UART_DATA_BITS {
  UART_DATA_BITS_5 = 0x00,
  UART_DATA_BITS_6 = 0x02,
  UART_DATA_BITS_7 = 0x04,
  UART_DATA_BITS_8 = 0x06,
  UART_DATA_BITS_9 = 0x0E }
- enum UART_STOP_BITS {
  UART_STOP_BITS_1 = 0x00,
  UART_STOP_BITS_2 = 0x80 }

Funkcje

- bool UartInit (void)

  Inicjuje UART.
- bool UartSetBaud (uint32_t baudRate)

  Ustawia prędkość transmisji (BaudRate)
- void UartSetDataBits (UART_DATA_BITS dataBits)

  Ustawia ilość bitów danych.
- void UartSetParity (UART_PARITY parity)

  Ustawia bity parzytości.
- void UartSetStopBits (UART_STOP_BITS stopBits)

  Ustawia ilość bitów stopu.
- bool UartSetBuffersSize (uint8_t size)
- void UartInitRs485 (volatile uint8_t ∗port, uint8_t pinConnectedToReDe)

  Inicjalizuje pin obsługujący kierunek przeływu danych na potrzeby half-duplexu RS485.
- uint16_t FrameCRC (volatile const Frame ∗f)

  Oblicza sumę kontrolną ramki.
- bool FrameCheckCRC (volatile Frame ∗f)

  Sprawdza poprawność sumy kontrolnej ramki.
- void FrameInit (volatile Frame ∗f, uint8_t a, uint8_t c, uint16_t d1, int16_t d2)

  Inicjuje strukturę ramki podanymi wartościami i oblicza jej sumę kontrolną
- void FrameCopy (volatile Frame ∗to, volatile Frame ∗from)

  Kopiuje ramkę
- bool SendFrame (Frame ∗f)

  Rozpoczyna wysyłanie ramki.
- bool GetFrame (volatile Frame ∗f)

  Sprawdza czy pobrano ramkę
- void RS485CMD ()

  Sprawdza czy przyszło jakieś polecenie, jeśli tak wykonuje je.