OpenServo RS485

1.0

Wygenerowano przez Doxygen 1.8.1

Cz, 31 maj 2012 10:58:50

Spis treści

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:

3 Indeks struktur danych

3.1 Struktury danych

Tutaj znajdują się struktury danych wraz z ich krótkimi opisami:

4 Indeks plików

4.1 Lista plików

Tutaj znajduje się lista wszystkich plików z ich krótkimi opisami:

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.

Odwołania w pid_registers_defaults().

5.1.2.2 #define DEFAULT_MIN_SEEK 0x0060

Minimalna pozycja.

Definicja w linii 184 pliku config.h.

Odwołania w pid_registers_defaults().

5.1.2.3 #define DEFAULT_PID_DEADBAND 0x01

Martwa strefa TowerPro SG5010.

Definicja w linii 179 pliku config.h.

Odwołania w pid_registers_defaults().

5.1.2.4 #define DEFAULT_PID_DGAIN 0x1000

Konfiguracja D w PID dla TowerPro SG5010.

Definicja w linii 175 pliku config.h.

Odwołania w pid_registers_defaults().

5.1.2.5 #define DEFAULT_PID_IGAIN 0x0001

Konfiguracja I w PID dla TowerPro SG5010.

Definicja w linii 177 pliku config.h.

Odwołania w pid_registers_defaults().

5.1.2.6 #define DEFAULT_PID_PGAIN 0x047C

Konfiguracja P w PID dla TowerPro SG5010.

Definicja w linii 173 pliku config.h.

Odwołania w pid_registers_defaults().

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 adc_init(), estimate_velocity(), estimator_init(), ipd_init(), ipd_position_to_pwm(), motion_append(), motion_init(), motion_next(), motion_reset(), pid_init(), pid_position_to_pwm(), power_init(), power_update(), pulse-_control_init(), pulse_control_update(), pwm_init(), pwm_update(), REG_CURVE_DELTA_HI, REG_CURVE_DEL-TA_LO, REG_CURVE_POSITION_HI, REG_CURVE_POSITION_LO, REG_SEEK_POSITION_HI, REG_SEEK_-POSITION_LO, REG_SEEK_VELOCITY_HI, REG_SEEK_VELOCITY_LO, registers_init(), registers_write_word(), regulator_init(), regulator_position_to_pwm(), RS485CMD(), UartInit(), UartInitRs485() i watchdog_init().

```
{
//  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 odebranej 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, FALSE, FrameCRC() i TRUE.

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, FALSE, FrameCheckCRC(), FrameCopy(), FrameCRC(), FRAMECRCMISMATCH, F-RAMECRCVALID, FrameReceived, TRUE 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_WRITENO-RMALREG, Frame::data1, Frame::data2, eeprom_erase(), eeprom_restore_registers(), eeprom_save_registers(), FRAMECRCVALID, FrameInit(), GetFrame(), MAX_WRITE_PROTECT_REGISTER, motion_append(), motion_-reset(), REG_TWI_ADDRESS, registers_defaults(), registers_read_word(), registers_write_word(), SendFrame(), TWI_CMD_CURVE_MOTION_APPEND, TWI_CMD_CURVE_MOTION_DISABLE, TWI_CMD_CURVE_MOTION-_ENABLE, TWI_CMD_CURVE_MOTION_RESET, TWI_CMD_EEPROM_ERASE, TWI_CMD_PWM_DISABLE, T-WI_CMD_PWM_ENABLE, TWI_CMD_REGISTERS_DEFAULT, TWI_CMD_REGISTERS_RESTORE, TWI_CMD-_REGISTERS_SAVE, TWI_CMD_RESET, TWI_CMD_VOLTAGE_READ, TWI_CMD_WRITE_DISABLE, TWI_C-MD_WRITE_ENABLE, Frame::valid i watchdog_hard_reset().

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, FALSE, FrameInit(), FrameOverflows, FrameSend, TRUE 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(), TRUE, 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.

Odwołuje się do TRUE.

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.

Odwołuje się do FALSE i TRUE.

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

6.2 Dokumentacja struktury motion_key

Pola danych

- uint16_t delta
- float position
- float in_velocity
- float out_velocity

6.2.1 Opis szczegółowy

Definicja w linii 38 pliku motion.c.

6.2.2 Dokumentacja pól

6.2.2.1 uint16_t delta

Definicja w linii 40 pliku motion.c.

Odwołania w motion_append(), motion_init() i motion_reset().

6.2.2.2 float in_velocity

Definicja w linii 42 pliku motion.c.

Odwołania w motion_append(), motion_init() i motion_reset().

6.2.2.3 float out_velocity

Definicja w linii 43 pliku motion.c.

Odwołania w motion_append(), motion_init() i motion_reset().

6.2.2.4 float position

Definicja w linii 41 pliku motion.c.

Odwołania w motion_append(), motion_init() i motion_reset().

Dokumentacja dla tej struktury została wygenerowana z pliku:

- motion.c

7 Dokumentacja plików

7.1 Dokumentacja pliku adc.c

```
#include <inttypes.h>
#include <avr/io.h>
#include <avr/interrupt.h>
#include "openservo.h"
#include "config.h"
#include "adc.h"
#include "timer.h"
```

Definicje

- #define ADC_CHANNEL_POWER 0
- #define ADC_CHANNEL_POSITION 1
- #define ADC_CHANNEL_VOLTAGE 2
- #define ADPS ((1<<ADPS2) | (1<<ADPS1) | (0<<ADPS0))
- #define CSPS ((1<<CS02) | (0<<CS01) | (1<<CS00))
- #define CRVALUE 78

Funkcje

- void adc_init (void)

Zmienne

- volatile uint8_t adc_channel
- volatile uint8_t adc_power_ready
- volatile uint16_t adc_power_value
- volatile uint8_t adc_position_ready
- volatile uint16_t adc_position_value
- volatile uint8_t adc_voltage_needed

7.1.1 Dokumentacja definicji

7.1.1.1 #define ADC_CHANNEL_POSITION 1

Definicja w linii 78 pliku adc.c.

Odwołania w adc_init().

7.1.1.2 #define ADC_CHANNEL_POWER 0

Definicja w linii 77 pliku adc.c.

7.1.1.3 #define ADC_CHANNEL_VOLTAGE 2

Definicja w linii 79 pliku adc.c.

7.1.1.4 #define ADPS ((1<<ADPS2) | (1<<ADPS1) | (0<<ADPS0))

Definicja w linii 83 pliku adc.c.

Odwołania w adc_init().

7.1.1.5 #define CRVALUE 78

Definicja w linii 91 pliku adc.c.

Odwołania w adc_init().

7.1.1.6 #define CSPS ((1<<CS02) | (0<<CS01) | (1<<CS00))

Definicja w linii 87 pliku adc.c.

Odwołania w adc_init().

7.1.2 Dokumentacja funkcji

7.1.2.1 void adc_init (void)

Definicja w linii 103 pliku adc.c.

Odwołuje się do adc_channel, ADC_CHANNEL_POSITION, adc_position_ready, adc_position_value, adc_power_ready, adc_power_value, adc_voltage_needed, ADPS, CRVALUE i CSPS.

Odwołania w main().

```
{
    // Read from position first.
    adc_channel = ADC_CHANNEL_POSITION;

    // Initialize flags and values.
    adc_power_ready = 0;
    adc_power_value = 0;
    adc_position_ready = 0;
    adc_position_value = 0;
    adc_voltage_needed = 1;

    //
    // Initialize ADC registers to yield a 125KHz clock.
    //

#if defined(__AVR_ATtiny45__) || defined(__AVR_ATtiny85__)
    // Make sure port PB4 (ADC3) and PB5 (ADC0) are set as input.
    PORTB &= ~((1<<PB4) | (1<<PB5));

    // Disable digital input for ADC3 and ADC0 to reduce power consumption.
    DIDR0 |= (1<<ADC3D) | (1<<ADC0D);

    // Set the ADC multiplexer selection register.
    ADMUX = (0<<REFS2) | (0<<REFS1) | (0<<REFS0) |          // Select VCC as
       voltage reference.
            (0<<MUX3) | (0<<MUX2) | (1<<MUX1) | (1<<MUX0) | // Select ADC3
       (PB3), no gain.
            (0<<ADLAR);                                     // Keep high bits
       right adjusted.

    // Set the ADC control and status register B.
    ADCSRB = (0<<BIN) |                                     // Gain working in
       unipolar mode.
             (0<<IPR) |                                     // No input
       polarity reversal.
             (0<<ADTS2) | (1<<ADTS1) | (1<<ADTS0);          // Timer/Counter0
       Compare Match A.

    // Set the ADC control and status register A.
    ADCSRA = (1<<ADEN) |                                    // Enable ADC.
             (0<<ADSC) |                                    // Don’s start yet,
       will be auto triggered.
             (1<<ADATE) |                                   // Start auto
       triggering.
             (1<<ADIE) |                                    // Activate ADC
       conversion complete interrupt.
             ADPS;                                          // Prescale --
       see above.
#endif // __AVR_ATtiny45__ || __AVR_ATtiny85____

#if defined(__AVR_ATmega8__)
    // Make sure ports PC0 (ADC0), PC1 (ADC1) and PC2 (ADC2) are set low.
    PORTC &= ~((1<<PC2) | (1<<PC1) | (1<<PC0));

    // Set the ADC multiplexer selection register.
    ADMUX = (0<<REFS1) | (1<<REFS0) |                       // Select AVCC as
       voltage reference.
            (0<<MUX3) | (0<<MUX2) | (1<<MUX1) | (0<<MUX0) | // Select ADC2
       (PC2) as analog input.
            (0<<ADLAR);                                     // Keep high bits
       right adjusted.


    // Set the ADC control and status register A.
    ADCSRA = (1<<ADEN) |                                    // Enable ADC.
             (1<<ADSC) |                                    // Start the first
       conversion.
             (0<<ADFR) |                                    // Free running
       disabled.
             (1<<ADIF) |                                    // Clear any
       pending interrupt.
             (1<<ADIE) |                                    // Activate ADC
       conversion complete interrupt.
```

```
             ADPS;                                         // Prescale --
       see above.

    // Reset the counter value to initiate another ADC sample at the specified
       time.
    TCNT0 = 256 - CRVALUE;
#endif // __AVR_ATmega8____

#if defined(__AVR_ATmega88__) || defined(__AVR_ATmega168__)
    // Make sure ports PC0 (ADC0), PC1 (ADC1) and PC2 (ADC2) are set low.
    PORTC &= ~((1<<PC0) | (1<<PC1) | (1<<PC2));

    // Disable digital input for ADC0, ADC1 and ADC2 to reduce power
       consumption.
    DIDR0 |= (1<<ADC2D) | (1<<ADC1D) |(1<<ADC0D);

    // Set the ADC multiplexer selection register.
    ADMUX = (0<<REFS1) | (1<<REFS0) |                      // Select AVCC as
       voltage reference.
            (0<<MUX3) | (0<<MUX2) | (1<<MUX1) | (0<<MUX0) | // Select ADC2
       (PC2) as analog input.
            (0<<ADLAR);                                    // Keep high bits
       right adjusted.

    // Set the ADC control and status register B.
    ADCSRB = (0<<ADTS2) | (1<<ADTS1) | (1<<ADTS0);          // Timer/Counter0
       Compare Match A.

    // Set the ADC control and status register A.
    ADCSRA = (1<<ADEN) |                                   // Enable ADC.
             (0<<ADSC) |                                   // Don’s start yet,
       will be auto triggered.
             (1<<ADATE) |                                  // Start auto
       triggering.
             (1<<ADIE) |                                   // Activate ADC
       conversion complete interrupt.
             ADPS;                                         // Prescale --
       see above.
#endif // __AVR_ATmega88__ || __AVR_ATmega168__

#if defined(__AVR_ATtiny45__) || defined(__AVR_ATtiny85__)
    // Set timer/counter0 control register A.
    TCCR0A = (0<<COM0A1) | (0<<COM0A0) |                   // Disconnect OCOA.
             (0<<COM0B1) | (0<<COM0B0) |                   // Disconnect OCOB.
             (1<<WGM01) | (0<<WGM00);                      // Mode 2 - clear
       timer on compare match.

    // Set timer/counter0 control register B.
    TCCR0B = (0<<FOC0A) | (0<<FOC0B) |                     // No force output
       compare A or B.
             (0<<WGM02) |                                  // Mode 2 - clear
       timer on compare match.
             CSPS;                                         // Timer clock
       prescale -- see above.

    // Set the timer/counter0 interrupt masks.
    TIMSK = (1<<OCIE0A) |                                  // Interrupt on
       compare match A.
            (0<<OCIE0B) |                                  // No interrupt on
       compare match B.
            (0<<TOIE0);                                    // No interrupt on
       overflow.

    // Set the compare match A value which initiates an ADC sample.
    OCR0A = CRVALUE;
#endif // __AVR_ATtiny45__ || __AVR_ATtiny85__

#if defined(__AVR_ATmega8__)
    // Set timer/counter0 control register.
    TCCR0 = CSPS;                                          // Timer clock
       prescale -- see above.

    // Clear any pending interrupt.
    TIFR |= (1<<TOV0);                                     // Interrupt on
       overflow.

    // Set the timer/counter0 interrupt masks.
    TIMSK |= (1<<TOIE0);                                   // Interrupt on
       overflow.
#endif // __AVR_ATmega8____

#if defined(__AVR_ATmega88__) || defined(__AVR_ATmega168__)
    // Set timer/counter0 control register A.
    TCCR0A = (0<<COM0A1) | (0<<COM0A0) |                   // Disconnect OCOA.
             (0<<COM0B1) | (0<<COM0B0) |                   // Disconnect OCOB.
             (1<<WGM01) | (0<<WGM00);                      // Mode 2 - clear
       timer on compare match.
    // Set timer/counter0 control register B.
    TCCR0B = (0<<FOC0A) | (0<<FOC0B) |                      // No force output
       compare A or B.
             (0<<WGM02) |                                   // Mode 2 - clear
       timer on compare match.
             CSPS;                                          // Timer clock
       prescale -- see above.

    // Set the timer/counter0 interrupt masks.
    TIMSK0 = (1<<OCIE0A) |                                  // Interrupt on
       compare match A.
             (0<<OCIE0B) |                                  // No interrupt on
       compare match B.
             (0<<TOIE0);                                    // No interrupt on
       overflow.

    // Set the compare match A value which initiates an ADC sample.
    OCR0A = CRVALUE;
#endif // __AVR_ATmega88__ || __AVR_ATmega168__
}
```

7.1.3 Dokumentacja zmiennych

7.1.3.1 volatile uint8_t adc_channel

Definicja w linii 95 pliku adc.c.

Odwołania w adc_init().

7.1.3.2 volatile uint8_t adc_position_ready

Definicja w linii 98 pliku adc.c.

Odwołania w adc_init().

7.1.3.3 volatile uint16_t adc_position_value

Definicja w linii 99 pliku adc.c.

Odwołania w adc_init().

7.1.3.4 volatile uint8_t adc_power_ready

Definicja w linii 96 pliku adc.c.

Odwołania w adc_init().

7.1.3.5 volatile uint16_t adc_power_value

Definicja w linii 97 pliku adc.c.

Odwołania w adc_init().

7.1.3.6 volatile uint8_t adc_voltage_needed

Definicja w linii 100 pliku adc.c.

Odwołania w adc_init().

7.2 Dokumentacja pliku adc.h

Funkcje

- void adc_init (void)

Zmienne

- volatile uint8_t adc_power_ready
- volatile uint16_t adc_power_value
- volatile uint8_t adc_position_ready
- volatile uint16_t adc_position_value
- volatile uint8_t adc_voltage_needed

7.2.1 Dokumentacja funkcji

7.2.1.1 void adc_init (void)

Definicja w linii 103 pliku adc.c.

Odwołuje się do adc_channel, ADC_CHANNEL_POSITION, adc_position_ready, adc_position_value, adc_power_ready, adc_power_value, adc_voltage_needed, ADPS, CRVALUE i CSPS.

Odwołania w main().

```
{
    // Read from position first.
    adc_channel = ADC_CHANNEL_POSITION;

    // Initialize flags and values.
    adc_power_ready = 0;
    adc_power_value = 0;
    adc_position_ready = 0;
    adc_position_value = 0;
    adc_voltage_needed = 1;

    //
    // Initialize ADC registers to yield a 125KHz clock.
    //

#if defined(__AVR_ATtiny45__) || defined(__AVR_ATtiny85__)
    // Make sure port PB4 (ADC3) and PB5 (ADC0) are set as input.
    PORTB &= ~((1<<PB4) | (1<<PB5));

    // Disable digital input for ADC3 and ADC0 to reduce power consumption.
    DIDR0 |= (1<<ADC3D) | (1<<ADC0D);

    // Set the ADC multiplexer selection register.
    ADMUX = (0<<REFS2) | (0<<REFS1) | (0<<REFS0) |          // Select VCC as
       voltage reference.
            (0<<MUX3) | (0<<MUX2) | (1<<MUX1) | (1<<MUX0) | // Select ADC3
       (PB3), no gain.
            (0<<ADLAR);                                     // Keep high bits
       right adjusted.

    // Set the ADC control and status register B.
    ADCSRB = (0<<BIN) |                                     // Gain working in
       unipolar mode.
             (0<<IPR) |                                     // No input
       polarity reversal.
             (0<<ADTS2) | (1<<ADTS1) | (1<<ADTS0);          // Timer/Counter0
       Compare Match A.

    // Set the ADC control and status register A.
    ADCSRA = (1<<ADEN) |                                    // Enable ADC.
             (0<<ADSC) |                                    // Don’s start yet,
       will be auto triggered.
             (1<<ADATE) |                                   // Start auto
       triggering.
             (1<<ADIE) |                                    // Activate ADC
       conversion complete interrupt.
             ADPS;                                          // Prescale --
       see above.
#endif // __AVR_ATtiny45__ || __AVR_ATtiny85____

#if defined(__AVR_ATmega8__)
    // Make sure ports PC0 (ADC0), PC1 (ADC1) and PC2 (ADC2) are set low.
    PORTC &= ~((1<<PC2) | (1<<PC1) | (1<<PC0));

    // Set the ADC multiplexer selection register.
    ADMUX = (0<<REFS1) | (1<<REFS0) |                       // Select AVCC as
       voltage reference.
            (0<<MUX3) | (0<<MUX2) | (1<<MUX1) | (0<<MUX0) | // Select ADC2
       (PC2) as analog input.
```

```
            (0<<ADLAR);                                     // Keep high bits
       right adjusted.

    // Set the ADC control and status register A.
    ADCSRA = (1<<ADEN) |                                    // Enable ADC.
             (1<<ADSC) |                                    // Start the first
       conversion.
             (0<<ADFR) |                                    // Free running
       disabled.
             (1<<ADIF) |                                    // Clear any
       pending interrupt.
             (1<<ADIE) |                                    // Activate ADC
       conversion complete interrupt.
             ADPS;                                          // Prescale --
       see above.

    // Reset the counter value to initiate another ADC sample at the specified
       time.
    TCNT0 = 256 - CRVALUE;
#endif // __AVR_ATmega8____

#if defined(__AVR_ATmega88__) || defined(__AVR_ATmega168__)
    // Make sure ports PC0 (ADC0), PC1 (ADC1) and PC2 (ADC2) are set low.
    PORTC &= ~((1<<PC0) | (1<<PC1) | (1<<PC2));

    // Disable digital input for ADC0, ADC1 and ADC2 to reduce power
       consumption.
    DIDR0 |= (1<<ADC2D) | (1<<ADC1D) |(1<<ADC0D);

    // Set the ADC multiplexer selection register.
    ADMUX = (0<<REFS1) | (1<<REFS0) |                       // Select AVCC as
       voltage reference.
            (0<<MUX3) | (0<<MUX2) | (1<<MUX1) | (0<<MUX0) | // Select ADC2
       (PC2) as analog input.
            (0<<ADLAR);                                     // Keep high bits
       right adjusted.

    // Set the ADC control and status register B.
    ADCSRB = (0<<ADTS2) | (1<<ADTS1) | (1<<ADTS0);          // Timer/Counter0
       Compare Match A.

    // Set the ADC control and status register A.
    ADCSRA = (1<<ADEN) |                                    // Enable ADC.
             (0<<ADSC) |                                    // Don’s start yet,
       will be auto triggered.
             (1<<ADATE) |                                   // Start auto
       triggering.
             (1<<ADIE) |                                    // Activate ADC
       conversion complete interrupt.
             ADPS;                                          // Prescale --
       see above.
#endif // __AVR_ATmega88__ || __AVR_ATmega168__

#if defined(__AVR_ATtiny45__) || defined(__AVR_ATtiny85__)
    // Set timer/counter0 control register A.
    TCCR0A = (0<<COM0A1) | (0<<COM0A0) |                    // Disconnect OCOA.
             (0<<COM0B1) | (0<<COM0B0) |                    // Disconnect OCOB.
             (1<<WGM01) | (0<<WGM00);                       // Mode 2 - clear
       timer on compare match.

    // Set timer/counter0 control register B.
    TCCR0B = (0<<FOC0A) | (0<<FOC0B) |                      // No force output
       compare A or B.
             (0<<WGM02) |                                   // Mode 2 - clear
       timer on compare match.
             CSPS;                                          // Timer clock
       prescale -- see above.

    // Set the timer/counter0 interrupt masks.
    TIMSK = (1<<OCIE0A) |                                   // Interrupt on
       compare match A.
            (0<<OCIE0B) |                                   // No interrupt on
       compare match B.
            (0<<TOIE0);                                     // No interrupt on
       overflow.

    // Set the compare match A value which initiates an ADC sample.
    OCR0A = CRVALUE;
#endif // __AVR_ATtiny45__ || __AVR_ATtiny85__

#if defined(__AVR_ATmega8__)
    // Set timer/counter0 control register.
    TCCR0 = CSPS;                                           // Timer clock
       prescale -- see above.

    // Clear any pending interrupt.
    TIFR |= (1<<TOV0);                                     // Interrupt on
       overflow.

    // Set the timer/counter0 interrupt masks.
    TIMSK |= (1<<TOIE0);                                   // Interrupt on
       overflow.
#endif // __AVR_ATmega8____

#if defined(__AVR_ATmega88__) || defined(__AVR_ATmega168__)
    // Set timer/counter0 control register A.
    TCCR0A = (0<<COM0A1) | (0<<COM0A0) |                   // Disconnect OC0A.
             (0<<COM0B1) | (0<<COM0B0) |                   // Disconnect OC0B.
             (1<<WGM01) | (0<<WGM00);                      // Mode 2 - clear
       timer on compare match.

    // Set timer/counter0 control register B.
    TCCR0B = (0<<FOC0A) | (0<<FOC0B) |                     // No force output
       compare A or B.
             (0<<WGM02) |                                  // Mode 2 - clear
       timer on compare match.
             CSPS;                                         // Timer clock
       prescale -- see above.

    // Set the timer/counter0 interrupt masks.
    TIMSK0 = (1<<OCIE0A) |                                 // Interrupt on
       compare match A.
             (0<<OCIE0B) |                                 // No interrupt on
       compare match B.
             (0<<TOIE0);                                   // No interrupt on
       overflow.

    // Set the compare match A value which initiates an ADC sample.
    OCR0A = CRVALUE;
#endif // __AVR_ATmega88__ || __AVR_ATmega168__
}
```

7.2.2 Dokumentacja zmiennych

7.2.2.1 volatile uint8_t adc_position_ready

Definicja w linii 98 pliku adc.c.

Odwołania w adc_init().

7.2.2.2 volatile uint16_t adc_position_value

Definicja w linii 99 pliku adc.c.

Odwołania w adc_init().

7.2.2.3 volatile uint8_t adc_power_ready

Definicja w linii 96 pliku adc.c.

Odwołania w adc_init().

7.2.2.4 volatile uint16_t adc_power_value

Definicja w linii 97 pliku adc.c.

Odwołania w adc_init().

7.2.2.5 volatile uint8_t adc_voltage_needed

Definicja w linii 100 pliku adc.c.

Odwołania w adc_init().

7.3 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

7.3.1 Dokumentacja definicji

7.3.1.1 #define CURVE_MOTION_ENABLED 1

Definicja w linii 99 pliku config.h.

7.3.1.2 #define DEFAULT_PWM_FREQ_DIVIDER 0x0010

Definicja w linii 189 pliku config.h.

Odwołania w pwm_registers_defaults().

7.3.1.3 #define ESTIMATOR_ENABLED (REGULATOR_MOTION_ENABLED)

Definicja w linii 88 pliku config.h.

7.3.1.4 #define FIXED_MATH_ENABLED (ESTIMATOR_ENABLED || REGULATOR_ENABLED)

Definicja w linii 93 pliku config.h.

7.3.1.5 #define HARDWARE_TYPE HARDWARE_TYPE_TOWERPRO_SG5010

Definicja w linii 147 pliku config.h.

7.3.1.6 #define HARDWARE_TYPE_FUTABA_S3003 1

Definicja w linii 139 pliku config.h.

7.3.1.7 #define HARDWARE_TYPE_HITEC_HS_311 2

Definicja w linii 140 pliku config.h.

7.3.1.8 #define HARDWARE_TYPE_HITEC_HS_475HB 3

Definicja w linii 141 pliku config.h.

7.3.1.9 #define HARDWARE_TYPE_TOWERPRO_SG5010 4

Definicja w linii 142 pliku config.h.

7.3.1.10 #define HARDWARE_TYPE_UNKNOWN 0

Definicja w linii 138 pliku config.h.

7.3.1.11 #define IPD_MOTION_ENABLED 0

Definicja w linii 70 pliku config.h.

7.3.1.12 #define MAIN_MOTION_TEST_ENABLED 0

Definicja w linii 105 pliku config.h.

7.3.1.13 #define PID_MOTION_ENABLED 1

Definicja w linii 60 pliku config.h.

7.3.1.14 #define PULSE_CONTROL_ENABLED 0

Definicja w linii 111 pliku config.h.

7.3.1.15 #define REGULATOR_MOTION_ENABLED 0

Definicja w linii 80 pliku config.h.

7.3.1.16 #define SWAP_PWM_DIRECTION_ENABLED 0

Definicja w linii 118 pliku config.h.

7.3.1.17 #define TWI_CHECKED_ENABLED 0

Definicja w linii 52 pliku config.h.

7.4 Dokumentacja pliku curve.c

```
#include <stdint.h>
#include "openservo.h"
#include "config.h"
#include "curve.h"
```

Funkcje

- void curve_init (uint16_t t0, uint16_t t1, float p0, float p1, float v0, float v1)
- void curve_solve (uint16_t t, float ∗x, float ∗dx)

Zmienne

- uint16_t curve_t0
- uint16_t curve_t1
- uint16_t curve_duration
- float curve_p0
- float curve_p1
- float curve_v0
- float curve_v1

7.4.1 Dokumentacja funkcji

7.4.1.1 void curve_init (uint16_t *t0,* uint16_t *t1,* float *p0,* float *p1,* float *v0,* float *v1*)

Definicja w linii 53 pliku curve.c.

Odwołuje się do curve_duration, curve_p0, curve_p1, curve_t0, curve_t1, curve_v0 i curve_v1.

Odwołania w motion_append(), motion_init(), motion_next() i motion_reset().

```
{
    // Set the time parameters.
    curve_t0 = t0;
    curve_t1 = t1;
    curve_duration = t1 - t0;
    curve_duration_float = (float) curve_duration;

    // The tangents are expressed as slope of value/time.  The time span will
    // be normalized to 0.0 to 1.0 range so correct the tangents by scaling
    // them by the duration of the curve.
    v0 *= curve_duration_float;
    v1 *= curve_duration_float;

    // Set the curve parameters.
    curve_p0 = p0;
    curve_p1 = p1;
    curve_v0 = v0;
    curve_v1 = v1;

    // Set the cubic coefficients by multiplying the matrix form of
    // the Hermite curve by the curve parameters p0, p1, v0 and v1.
    //
    // | a |   |  2  -2   1   1 |   |       p0       |
    // | b |   | -3   3  -2  -1 |   |       p1       |
    // | c | = |  0   0   1   0 | . | (t1 - t0) * v0 |
    // | d |   |  1   0   0   0 |   | (t1 - t0) * v1 |
    //
    // a = 2p0 - 2p1 + v0 + v1
    // b = -3p0 + 3p1 -2v0 - v1
    // c = v0
    // d = p0
    //
    curve_a = (2.0 * p0) - (2.0 * p1) + v0 + v1;
    curve_b = -(3.0 * p0) + (3.0 * p1) - (2.0 * v0) - v1;
    curve_c = v0;
    curve_d = p0;
}
```

7.4.1.2 void curve_solve (uint16_t *t,* float ∗ *x,* float ∗ *dx*)

Definicja w linii 93 pliku curve.c.

Odwołuje się do curve_p0, curve_p1, curve_t0, curve_t1, curve_v0 i curve_v1.

Odwołania w motion_next().

```
{
    // Handle cases where t is outside and indise the curve.
    if (t <= curve_t0)
    {
        // Set x and in and out dx.
        *x = curve_p0;
        *dx = t < curve_t0 ? 0.0 : curve_v0;
    }
    else if (t >= curve_t1)
    {
        // Set x and in and out dx.
        *x = curve_p1;
        *dx = t > curve_t1 ? 0.0 : curve_v1;
    }
    else
    {
        // Subtract out the t0 value from t.
        float t1 = ((float) (t - curve_t0)) / curve_duration_float;
        float t2 = t1 * t1;
        float t3 = t2 * t1;

        // Determine the cubic polynomial.
        // x = at^3 + bt^2 + ct + d
        *x = (curve_a * t3) + (curve_b * t2) + (curve_c * t1) + curve_d;

        // Determine the cubic polynomial derivative.
        // dx = 3at^2 + 2bt + c
        *dx = (3.0 * curve_a * t2) + (2.0 * curve_b * t1) + curve_c;

        // The time span has been normalized to 0.0 to 1.0 range so correct
        // the derivative to the duration of the curve.
        *dx /= curve_duration_float;
    }
}
```

7.4.2 Dokumentacja zmiennych

7.4.2.1 uint16_t curve_duration

Definicja w linii 38 pliku curve.c.

Odwołania w curve_init().

7.4.2.2 float curve_p0

Definicja w linii 42 pliku curve.c.

Odwołania w curve_init() i curve_solve().

7.4.2.3 float curve_p1

Definicja w linii 43 pliku curve.c.

Odwołania w curve_init() i curve_solve().

7.4.2.4 uint16_t curve_t0

Definicja w linii 36 pliku curve.c.

Odwołania w curve_init() i curve_solve().

7.4.2.5 uint16_t curve_t1

Definicja w linii 37 pliku curve.c.

Odwołania w curve_init() i curve_solve().

7.4.2.6 float curve_v0

Definicja w linii 44 pliku curve.c.

Odwołania w curve_init() i curve_solve().

7.4.2.7 float curve_v1

Definicja w linii 45 pliku curve.c.

Odwołania w curve_init() i curve_solve().

7.5 Dokumentacja pliku curve.h

Funkcje

- void curve_init (uint16_t t0, uint16_t t1, float p0, float p1, float v0, float v1)
- void curve_solve (uint16_t t, float ∗x, float ∗dx)

Zmienne

- uint16_t curve_t0
- uint16_t curve_t1
- uint16_t curve_duration
- float curve_p0
- float curve_p1
- float curve_v0
- float curve_v1

7.5.1 Dokumentacja funkcji

7.5.1.1 void curve_init (uint16_t *t0,* uint16_t *t1,* float *p0,* float *p1,* float *v0,* float *v1*)

Definicja w linii 53 pliku curve.c.

Odwołuje się do curve_duration, curve_p0, curve_p1, curve_t0, curve_t1, curve_v0 i curve_v1.

Odwołania w motion_append(), motion_init(), motion_next() i motion_reset().

```
{
    // Set the time parameters.
    curve_t0 = t0;
    curve_t1 = t1;
    curve_duration = t1 - t0;
    curve_duration_float = (float) curve_duration;

    // The tangents are expressed as slope of value/time.  The time span will
    // be normalized to 0.0 to 1.0 range so correct the tangents by scaling
    // them by the duration of the curve.
    v0 *= curve_duration_float;
    v1 *= curve_duration_float;

    // Set the curve parameters.
    curve_p0 = p0;
    curve_p1 = p1;
    curve_v0 = v0;
    curve_v1 = v1;

    // Set the cubic coefficients by multiplying the matrix form of
    // the Hermite curve by the curve parameters p0, p1, v0 and v1.
    //
    // | a |   |  2  -2   1   1 |   |        p0       |
    // | b |   | -3   3  -2  -1 |   |        p1       |
    // | c | = |  0   0   1   0 | . | (t1 - t0) * v0 |
    // | d |   |  1   0   0   0 |   | (t1 - t0) * v1 |
    //
    // a = 2p0 - 2p1 + v0 + v1
    // b = -3p0 + 3p1 -2v0 - v1
    // c = v0
    // d = p0
    //
    curve_a = (2.0 * p0) - (2.0 * p1) + v0 + v1;
    curve_b = -(3.0 * p0) + (3.0 * p1) - (2.0 * v0) - v1;
    curve_c = v0;
    curve_d = p0;
}
```

7.5.1.2 void curve_solve (uint16_t *t,* float ∗ *x,* float ∗ *dx*)

Definicja w linii 93 pliku curve.c.

Odwołuje się do curve_p0, curve_p1, curve_t0, curve_t1, curve_v0 i curve_v1.

Odwołania w motion_next().

```
{
    // Handle cases where t is outside and indise the curve.
    if (t <= curve_t0)
    {
        // Set x and in and out dx.
        *x = curve_p0;
        *dx = t < curve_t0 ? 0.0 : curve_v0;
    }
    else if (t >= curve_t1)
    {
        // Set x and in and out dx.
        *x = curve_p1;
        *dx = t > curve_t1 ? 0.0 : curve_v1;
    }
    else
    {
        // Subtract out the t0 value from t.
        float t1 = ((float) (t - curve_t0)) / curve_duration_float;
        float t2 = t1 * t1;
        float t3 = t2 * t1;

        // Determine the cubic polynomial.
        // x = at^3 + bt^2 + ct + d
        *x = (curve_a * t3) + (curve_b * t2) + (curve_c * t1) + curve_d;

        // Determine the cubic polynomial derivative.
        // dx = 3at^2 + 2bt + c
        *dx = (3.0 * curve_a * t2) + (2.0 * curve_b * t1) + curve_c;

        // The time span has been normalized to 0.0 to 1.0 range so correct
        // the derivative to the duration of the curve.
        *dx /= curve_duration_float;
    }
}
```

7.5.2 Dokumentacja zmiennych

7.5.2.1 uint16_t curve_duration

Definicja w linii 38 pliku curve.c.

Odwołania w curve_init().

7.5.2.2 float curve_p0

Definicja w linii 42 pliku curve.c.

Odwołania w curve_init() i curve_solve().

7.5.2.3 float curve_p1

Definicja w linii 43 pliku curve.c.

Odwołania w curve_init() i curve_solve().

7.5.2.4 uint16_t curve_t0

Definicja w linii 36 pliku curve.c.

Odwołania w curve_init() i curve_solve().

7.5.2.5 uint16_t curve_t1

Definicja w linii 37 pliku curve.c.

Odwołania w curve_init() i curve_solve().

7.5.2.6 float curve_v0

Definicja w linii 44 pliku curve.c.

Odwołania w curve_init() i curve_solve().

7.5.2.7 float curve_v1

Definicja w linii 45 pliku curve.c.

Odwołania w curve_init() i curve_solve().

7.6 Dokumentacja pliku eeprom.c

```
#include <inttypes.h>
#include <string.h>
#include <avr/io.h>
#include <avr/eeprom.h>
#include "openservo.h"
#include "config.h"
#include "eeprom.h"
#include "registers.h"
```

Funkcje

- uint8_t eeprom_erase (void)
- uint8_t eeprom_restore_registers (void)
- uint8_t eeprom_save_registers (void)

7.6.1 Dokumentacja funkcji

7.6.1.1 uint8_t eeprom_erase (void)

Definicja w linii 56 pliku eeprom.c.

Odwołania w RS485CMD().

```
{
    uint16_t i;
    uint8_t buffer[16];

    // XXX Disable PWM to servo motor while reading registers.

    // Clear the buffer contents to 0xFF.
    memset(buffer, 0xFF, sizeof(buffer));

    // Loop over the EEPROM in buffer increments.
    for (i = 0; i < E2END; i += sizeof(buffer))
    {
        // Write the buffer to the block of EEPROM.
        eeprom_write_block(buffer, (void *) i, sizeof(buffer));
    }

    // XXX Restore PWM to servo motor.

    // Return success.
    return 1;
}
```

7.6.1.2 uint8_t eeprom_restore_registers (void)

Definicja w linii 81 pliku eeprom.c.

Odwołuje się do EEPROM_VERSION, MIN_WRITE_PROTECT_REGISTER, REDIRECT_REGISTER_COUNT, registers i WRITE_PROTECT_REGISTER_COUNT.

Odwołania w registers_init() i RS485CMD().

```
{
    uint8_t header[2];
    // XXX Disable PWM to servo motor while reading registers.

    // Read EEPROM header which is the first two bytes of EEPROM.
    eeprom_read_block(&header[0], (void *) 0, 2);

    // Does the version match?
    if (header[0] != EEPROM_VERSION) return 0;

    // Read the write protected and redirect registers from EEPROM.
    eeprom_read_block(&registers[MIN_WRITE_PROTECT_REGISTER
      ], (void *) 2, WRITE_PROTECT_REGISTER_COUNT +
      REDIRECT_REGISTER_COUNT);

    // Does the checksum match?
    if (header[1] != eeprom_checksum(&registers[
      MIN_WRITE_PROTECT_REGISTER], WRITE_PROTECT_REGISTER_COUNT +
      REDIRECT_REGISTER_COUNT, EEPROM_VERSION))
      return 0;

    // XXX Restore PWM to servo motor.

    // Return success.
    return 1;
}
```

7.6.1.3 uint8_t eeprom_save_registers (void)

Definicja w linii 108 pliku eeprom.c.

Odwołuje się do EEPROM_VERSION, MIN_WRITE_PROTECT_REGISTER, REDIRECT_REGISTER_COUNT, registers i WRITE_PROTECT_REGISTER_COUNT.

Odwołania w RS485CMD().

```
{
    uint8_t header[2];

    // XXX Disable PWM to servo motor while reading registers.

    // Fill in the EEPROM header.
    header[0] = EEPROM_VERSION;
    header[1] = eeprom_checksum(&registers[MIN_WRITE_PROTECT_REGISTER
      ], WRITE_PROTECT_REGISTER_COUNT +
      REDIRECT_REGISTER_COUNT, EEPROM_VERSION);

    // Write the EEPROM header which is the first two bytes of EEPROM.
    eeprom_write_block(&header[0], (void *) 0, 2);

    // Write the write protected and redirect registers from EEPROM.
    eeprom_write_block(&registers[MIN_WRITE_PROTECT_REGISTER
      ], (void *) 2, WRITE_PROTECT_REGISTER_COUNT +
      REDIRECT_REGISTER_COUNT);

    // XXX Restore PWM to servo motor.

    // Return success.
    return 1;
}
```

7.7 Dokumentacja pliku eeprom.h

Definicje

- #define EEPROM_VERSION 0x03

Funkcje

- uint8_t eeprom_erase (void)
- uint8_t eeprom_restore_registers (void)
- uint8_t eeprom_save_registers (void)

7.7.1 Dokumentacja definicji

7.7.1.1 #define EEPROM_VERSION 0x03

Definicja w linii 35 pliku eeprom.h.

Odwołania w eeprom_restore_registers() i eeprom_save_registers().

7.7.2 Dokumentacja funkcji

7.7.2.1 uint8_t eeprom_erase (void)

Definicja w linii 56 pliku eeprom.c.

Odwołania w RS485CMD().

```
{
    uint16_t i;
    uint8_t buffer[16];

    // XXX Disable PWM to servo motor while reading registers.

    // Clear the buffer contents to 0xFF.
    memset(buffer, 0xFF, sizeof(buffer));

    // Loop over the EEPROM in buffer increments.
    for (i = 0; i < E2END; i += sizeof(buffer))
    {
        // Write the buffer to the block of EEPROM.
        eeprom_write_block(buffer, (void *) i, sizeof(buffer));
    }

    // XXX Restore PWM to servo motor.

    // Return success.
    return 1;
}
```

7.7.2.2 uint8_t eeprom_restore_registers (void)

Definicja w linii 81 pliku eeprom.c.

Odwołuje się do EEPROM_VERSION, MIN_WRITE_PROTECT_REGISTER, REDIRECT_REGISTER_COUNT, registers i WRITE_PROTECT_REGISTER_COUNT.

Odwołania w registers_init() i RS485CMD().

```
{
    uint8_t header[2];

    // XXX Disable PWM to servo motor while reading registers.

    // Read EEPROM header which is the first two bytes of EEPROM.
    eeprom_read_block(&header[0], (void *) 0, 2);

    // Does the version match?
    if (header[0] != EEPROM_VERSION) return 0;

    // Read the write protected and redirect registers from EEPROM.
    eeprom_read_block(&registers[MIN_WRITE_PROTECT_REGISTER
      ], (void *) 2, WRITE_PROTECT_REGISTER_COUNT +
      REDIRECT_REGISTER_COUNT);

    // Does the checksum match?
    if (header[1] != eeprom_checksum(&registers[
      MIN_WRITE_PROTECT_REGISTER], WRITE_PROTECT_REGISTER_COUNT +
      REDIRECT_REGISTER_COUNT, EEPROM_VERSION))
      return 0;

    // XXX Restore PWM to servo motor.

    // Return success.
    return 1;
}
```

7.7.2.3 uint8_t eeprom_save_registers (void)

Definicja w linii 108 pliku eeprom.c.

Odwołuje się do EEPROM_VERSION, MIN_WRITE_PROTECT_REGISTER, REDIRECT_REGISTER_COUNT, registers i WRITE_PROTECT_REGISTER_COUNT.

Odwołania w RS485CMD().

```
{
    uint8_t header[2];

    // XXX Disable PWM to servo motor while reading registers.

    // Fill in the EEPROM header.
    header[0] = EEPROM_VERSION;
    header[1] = eeprom_checksum(&registers[MIN_WRITE_PROTECT_REGISTER
      ], WRITE_PROTECT_REGISTER_COUNT +
      REDIRECT_REGISTER_COUNT, EEPROM_VERSION);

    // Write the EEPROM header which is the first two bytes of EEPROM.
    eeprom_write_block(&header[0], (void *) 0, 2);

    // Write the write protected and redirect registers from EEPROM.
    eeprom_write_block(&registers[MIN_WRITE_PROTECT_REGISTER
      ], (void *) 2, WRITE_PROTECT_REGISTER_COUNT +
      REDIRECT_REGISTER_COUNT);

    // XXX Restore PWM to servo motor.

    // Return success.
    return 1;
}
```

7.8 Dokumentacja pliku estimator.c

```
#include <inttypes.h>
#include "openservo.h"
#include "config.h"
#include "registers.h"
#include "math.h"
```

7.9 Dokumentacja pliku estimator.h

Funkcje

- void estimator_init (void)
- void estimator_registers_defaults (void)
- void estimate_velocity (int16_t position)

7.9.1 Dokumentacja funkcji

7.9.1.1 void estimate_velocity (int16_t *position*)

Odwołania w main().

7.9.1.2 void estimator_init (void)

Odwołania w main().

7.9.1.3 void estimator_registers_defaults (void)

Odwołania w registers_defaults().

7.10 Dokumentacja pliku ipd.c

```
#include <inttypes.h>
#include "openservo.h"
#include "config.h"
#include "ipd.h"
#include "registers.h"
```

7.11 Dokumentacja pliku ipd.h

Funkcje

- void ipd_init (void)
- void ipd_registers_defaults (void)
- int16_t ipd_position_to_pwm (int16_t position)

7.11.1 Dokumentacja funkcji

7.11.1.1 void ipd_init (void)

Odwołania w main().

7.11.1.2 int16_t ipd_position_to_pwm (int16_t *position*)

Odwołania w main().

7.11.1.3 void ipd_registers_defaults (void)

Odwołania w registers_defaults().

7.12 Dokumentacja pliku macros.h

```
#include <avr/io.h>
```

Definicje

- #define __USER_LABEL_PREFIX__ _
- #define _L $
- #define CONCAT1(a, b) CONCAT2(a, b)
- #define CONCAT2(a, b) a ## b
- #define _U(x) CONCAT1(__USER_LABEL_PREFIX__, x)
- #define _R(x) CONCAT1(__REGISTER_PREFIX__, x)
- #define r0 _R(r0)
- #define r1 _R(r1)
- #define r2 _R(r2)
- #define r3 _R(r3)
- #define r4 _R(r4)
- #define r5 _R(r5)
- #define r6 _R(r6)
- #define r7 _R(r7)
- #define r8 _R(r8)
- #define r9 _R(r9)

- #define r10 _R(r10)
- #define r11 _R(r11)
- #define r12 _R(r12)
- #define r13 _R(r13)
- #define r14 _R(r14)
- #define r15 _R(r15)
- #define r16 _R(r16)
- #define r17 _R(r17)
- #define r18 _R(r18)
- #define r19 _R(r19)
- #define r20 _R(r20)
- #define r21 _R(r21)
- #define r22 _R(r22)
- #define r23 _R(r23)
- #define r24 _R(r24)
- #define r25 _R(r25)
- #define r26 _R(r26)
- #define r27 _R(r27)
- #define r28 _R(r28)
- #define r29 _R(r29)
- #define r30 _R(r30)
- #define r31 _R(r31)
- #define __tmp_reg__ r0
- #define __zero_reg__ r1
- #define XJMP rjmp
- #define XCALL rcall
- #define PROLOGUE_SAVES(offset) XJMP (__prologue_saves__ + 2 ∗ (offset))
- #define EPILOGUE_RESTORES(offset) XJMP (__epilogue_restores__ + 2 ∗ (offset))
- #define BIG_CODE 0

Funkcje

- Invalid X_movw arg endif if ((.L_movw_src)-(.L_movw_dst)).if(((.L_movw_src)|(.L_movw_dst))&0x01).if(((.L_movw_src)-(.L_movw_dst))&0x80) mov(.L_movw_dst)+1
- Invalid X_movw arg endif L_movw_src mov (.L_movw_dst)

Zmienne

- macro X_movw dst src L_movw_dst
- macro X_movw dst src r0
- macro X_movw dst src r1
- macro X_movw dst src r2
- macro X_movw dst src r3
- macro X_movw dst src r4
- macro X_movw dst src r5
- macro X_movw dst src r6
- macro X_movw dst src r7
- macro X_movw dst src r8
- macro X_movw dst src r9
- macro X_movw dst src r10
- macro X_movw dst src r11
- macro X_movw dst src r12
- macro X_movw dst src r13
- macro X_movw dst src r14

- macro X_movw dst src r15
- macro X_movw dst src r16
- macro X_movw dst src r17
- macro X_movw dst src r18
- macro X_movw dst src r19
- macro X_movw dst src r20
- macro X_movw dst src r21
- macro X_movw dst src r22
- macro X_movw dst src r23
- macro X_movw dst src r24
- macro X_movw dst src r25
- macro X_movw dst src r26
- macro X_movw dst src r27
- macro X_movw dst src r28
- macro X_movw dst src r29
- macro X_movw dst src r30
- macro X_movw dst src r31 ifc reg
- macro X_movw dst src r31 ifc dst src L_movw_src
- macro X_movw dst src r31 ifc dst src R0
- macro X_movw dst src r31 ifc dst src R1
- macro X_movw dst src r31 ifc dst src R2
- macro X_movw dst src r31 ifc dst src R3
- macro X_movw dst src r31 ifc dst src R4
- macro X_movw dst src r31 ifc dst src R5
- macro X_movw dst src r31 ifc dst src R6
- macro X_movw dst src r31 ifc dst src R7
- macro X_movw dst src r31 ifc dst src R8
- macro X_movw dst src r31 ifc dst src R9
- macro X_movw dst src r31 ifc dst src R10
- macro X_movw dst src r31 ifc dst src R11
- macro X_movw dst src r31 ifc dst src R12
- macro X_movw dst src r31 ifc dst src R13
- macro X_movw dst src r31 ifc dst src R14
- macro X_movw dst src r31 ifc dst src R15
- macro X_movw dst src r31 ifc dst src R16
- macro X_movw dst src r31 ifc dst src R17

- macro X_movw dst src r31 ifc dst src R18
- macro X_movw dst src r31 ifc dst src R19
- macro X_movw dst src r31 ifc dst src R20
- macro X_movw dst src r31 ifc dst src R21
- macro X_movw dst src r31 ifc dst src R22
- macro X_movw dst src r31 ifc dst src R23
- macro X_movw dst src r31 ifc dst src R24
- macro X_movw dst src r31 ifc dst src R25
- macro X_movw dst src r31 ifc dst src R26
- macro X_movw dst src r31 ifc dst src R27
- macro X_movw dst src r31 ifc dst src R28
- macro X_movw dst src r31 ifc dst src R29
- macro X_movw dst src r31 ifc dst src R30
- Invalid X_movw arg endif L_movw_src L_movw_src else L_movw_src L_movw_src endif else L_movw_src L_movw_src endif endif endm macro X_lpm dst = r0
- Invalid X_movw arg endif L_movw_src L_movw_src else L_movw_src L_movw_src endif else L_movw_src L_movw_src endif endif endm macro X_lpm src
- Invalid X_movw arg endif L_movw_src L_movw_src else L_movw_src L_movw_src endif else L_movw_src L_movw_src endif endif endm macro X_lpm r31 ifc dst L_lpm_dst
- Invalid dst arg of X_lpm macro endif L_lpm_src
- Invalid dst arg of X_lpm macro endif z
- Invalid dst arg of X_lpm macro endif Z
- Invalid src arg of X_lpm macro endif if L_lpm_src< 2.if.L_lpm_dst==0lpm.elselpmmov.L_lpm_dst, r0.endif.else.if(.L_lpm_dst >

7.12.1 Dokumentacja definicji

7.12.1.1 #define __tmp_reg__ r0

Definicja w linii 101 pliku macros.h.

7.12.1.2 #define __USER_LABEL_PREFIX__ _

Definicja w linii 47 pliku macros.h.

7.12.1.3 #define __zero_reg__ r1

Definicja w linii 105 pliku macros.h.

7.12.1.4 #define _L $

Definicja w linii 55 pliku macros.h.

7.12.1.5 #define _R(*x*) CONCAT1(__REGISTER_PREFIX__, x)

Definicja w linii 62 pliku macros.h.

7.12.1.6 #define _U(*x*) CONCAT1(__USER_LABEL_PREFIX__, x)

Definicja w linii 60 pliku macros.h.

7.12.1.7 #define BIG_CODE 0

Definicja w linii 123 pliku macros.h.

7.12.1.8 #define CONCAT1(*a,* *b*) CONCAT2(a, b)

Definicja w linii 57 pliku macros.h.

7.12.1.9 #define CONCAT2(*a,* *b*) a ## b

Definicja w linii 58 pliku macros.h.

7.12.1.10 #define EPILOGUE_RESTORES(*offset*) XJMP (__epilogue_restores__ + 2 ∗ (offset))

Definicja w linii 118 pliku macros.h.

7.12.1.11 #define PROLOGUE_SAVES(*offset*) XJMP (__prologue_saves__ + 2 ∗ (offset))

Definicja w linii 117 pliku macros.h.

7.12.1.12 Invalid X_movw arg endif L_movw_src L_movw_src else L_movw_src L_movw_src endif else L_movw_src L_movw_src endif endif endm macro X_lpm r0 _R(r0)

Definicja w linii 67 pliku macros.h.

7.12.1.13 Invalid X_movw arg endif L_movw_src L_movw_src else L_movw_src L_movw_src endif else L_movw_src L_movw_src endif endif endm macro X_lpm r1 _R(r1)

Definicja w linii 68 pliku macros.h.

7.12.1.14 Invalid X_movw arg endif L_movw_src L_movw_src else L_movw_src L_movw_src endif else L_movw_src L_movw_src endif endif endm macro X_lpm r10 _R(r10)

Definicja w linii 77 pliku macros.h.

7.12.1.15 Invalid X_movw arg endif L_movw_src L_movw_src else L_movw_src L_movw_src endif else L_movw_src L_movw_src endif endif endm macro X_lpm r11 _R(r11)

Definicja w linii 78 pliku macros.h.

7.12.1.16 Invalid X_movw arg endif L_movw_src L_movw_src else L_movw_src L_movw_src endif else L_movw_src L_movw_src endif endif endm macro X_lpm r12 _R(r12)

Definicja w linii 79 pliku macros.h.

7.12.1.17 Invalid X_movw arg endif L_movw_src L_movw_src else L_movw_src L_movw_src endif else L_movw_src L_movw_src endif endif endm macro X_lpm r13 _R(r13)

Definicja w linii 80 pliku macros.h.

7.12.1.18 Invalid X_movw arg endif L_movw_src L_movw_src else L_movw_src L_movw_src endif else L_movw_src L_movw_src endif endif endm macro X_lpm r14 _R(r14)

Definicja w linii 81 pliku macros.h.

7.12.1.19 Invalid X_movw arg endif L_movw_src L_movw_src else L_movw_src L_movw_src endif else L_movw_src L_movw_src endif endif endm macro X_lpm r15 _R(r15)

Definicja w linii 82 pliku macros.h.

7.12.1.20 Invalid X_movw arg endif L_movw_src L_movw_src else L_movw_src L_movw_src endif else L_movw_src L_movw_src endif endif endm macro X_lpm r16 _R(r16)

Definicja w linii 83 pliku macros.h.

7.12.1.21 Invalid X_movw arg endif L_movw_src L_movw_src else L_movw_src L_movw_src endif else L_movw_src L_movw_src endif endif endm macro X_lpm r17 _R(r17)

Definicja w linii 84 pliku macros.h.

7.12.1.22 Invalid X_movw arg endif L_movw_src L_movw_src else L_movw_src L_movw_src endif else L_movw_src L_movw_src endif endif endm macro X_lpm r18 _R(r18)

Definicja w linii 85 pliku macros.h.

7.12.1.23 Invalid X_movw arg endif L_movw_src L_movw_src else L_movw_src L_movw_src endif else L_movw_src L_movw_src endif endif endm macro X_lpm r19 _R(r19)

Definicja w linii 86 pliku macros.h.

7.12.1.24 Invalid X_movw arg endif L_movw_src L_movw_src else L_movw_src L_movw_src endif else L_movw_src L_movw_src endif endif endm macro X_lpm r2 _R(r2)

Definicja w linii 69 pliku macros.h.

7.12.1.25 Invalid X_movw arg endif L_movw_src L_movw_src else L_movw_src L_movw_src endif else L_movw_src L_movw_src endif endif endm macro X_lpm r20 _R(r20)

Definicja w linii 87 pliku macros.h.

7.12.1.26 Invalid X_movw arg endif L_movw_src L_movw_src else L_movw_src L_movw_src endif else L_movw_src L_movw_src endif endif endm macro X_lpm r21 _R(r21)

Definicja w linii 88 pliku macros.h.

7.12.1.27 Invalid X_movw arg endif L_movw_src L_movw_src else L_movw_src L_movw_src endif else L_movw_src L_movw_src endif endif endm macro X_lpm r22 _R(r22)

Definicja w linii 89 pliku macros.h.

7.12.1.28 Invalid X_movw arg endif L_movw_src L_movw_src else L_movw_src L_movw_src endif else L_movw_src L_movw_src endif endif endm macro X_lpm r23 _R(r23)

Definicja w linii 90 pliku macros.h.

7.12.1.29 Invalid X_movw arg endif L_movw_src L_movw_src else L_movw_src L_movw_src endif else L_movw_src L_movw_src endif endif endm macro X_lpm r24 _R(r24)

Definicja w linii 91 pliku macros.h.

7.12.1.30 Invalid X_movw arg endif L_movw_src L_movw_src else L_movw_src L_movw_src endif else L_movw_src L_movw_src endif endif endm macro X_lpm r25 _R(r25)

Definicja w linii 92 pliku macros.h.

7.12.1.31 Invalid X_movw arg endif L_movw_src L_movw_src else L_movw_src L_movw_src endif else L_movw_src L_movw_src endif endif endm macro X_lpm r26 _R(r26)

Definicja w linii 93 pliku macros.h.

7.12.1.32 Invalid X_movw arg endif L_movw_src L_movw_src else L_movw_src L_movw_src endif else L_movw_src L_movw_src endif endif endm macro X_lpm r27 _R(r27)

Definicja w linii 94 pliku macros.h.

7.12.1.33 Invalid X_movw arg endif L_movw_src L_movw_src else L_movw_src L_movw_src endif else L_movw_src L_movw_src endif endif endm macro X_lpm r28 _R(r28)

Definicja w linii 95 pliku macros.h.

7.12.1.34 Invalid X_movw arg endif L_movw_src L_movw_src else L_movw_src L_movw_src endif else L_movw_src L_movw_src endif endif endm macro X_lpm r29 _R(r29)

Definicja w linii 96 pliku macros.h.

7.12.1.35 Invalid X_movw arg endif L_movw_src L_movw_src else L_movw_src L_movw_src endif else L_movw_src L_movw_src endif endif endm macro X_lpm r3 _R(r3)

Definicja w linii 70 pliku macros.h.

7.12.1.36 Registers and are inhibited as X_lpm Z dst endif lpm if L_lpm_dst mov r0 endif adiw endif endm macro LPM_R0_ZPLUS_INIT hhi endm macro LPM_R0_ZPLUS_NEXT hhi lpm adiw r30 _R(r30)

Definicja w linii 97 pliku macros.h.

7.12.1.37 #define r31 _R(r31)

Definicja w linii 98 pliku macros.h.

7.12.1.38 Invalid X_movw arg endif L_movw_src L_movw_src else L_movw_src L_movw_src endif else L_movw_src L_movw_src endif endif endm macro X_lpm r4 _R(r4)

Definicja w linii 71 pliku macros.h.

7.12.1.39 Invalid X_movw arg endif L_movw_src L_movw_src else L_movw_src L_movw_src endif else L_movw_src L_movw_src endif endif endm macro X_lpm r5 _R(r5)

Definicja w linii 72 pliku macros.h.

7.12.1.40 Invalid X_movw arg endif L_movw_src L_movw_src else L_movw_src L_movw_src endif else L_movw_src L_movw_src endif endif endm macro X_lpm r6 _R(r6)

Definicja w linii 73 pliku macros.h.

7.12.1.41 Invalid X_movw arg endif L_movw_src L_movw_src else L_movw_src L_movw_src endif else L_movw_src L_movw_src endif endif endm macro X_lpm r7 _R(r7)

Definicja w linii 74 pliku macros.h.

7.12.1.42 Invalid X_movw arg endif L_movw_src L_movw_src else L_movw_src L_movw_src endif else L_movw_src L_movw_src endif endif endm macro X_lpm r8 _R(r8)

Definicja w linii 75 pliku macros.h.

7.12.1.43 Invalid X_movw arg endif L_movw_src L_movw_src else L_movw_src L_movw_src endif else L_movw_src L_movw_src endif endif endm macro X_lpm r9 _R(r9)

Definicja w linii 76 pliku macros.h.

7.12.1.44 #define XCALL rcall

Definicja w linii 113 pliku macros.h.

7.12.1.45 #define XJMP rjmp

Definicja w linii 112 pliku macros.h.

7.12.2 Dokumentacja funkcji

7.12.2.1 Invalid X_movw arg endif if ((.L_movw_src)-(.L_movw_dst))

7.12.2.2 Invalid X_movw arg endif L_movw_src L_movw_src else L_movw_src L_movw_src endif else L_movw_src mov (. *L_movw_dst*)

7.12.3 Dokumentacja zmiennych

7.12.3.1 Invalid X_movw arg endif L_movw_src L_movw_src else L_movw_src L_movw_src endif else L_movw_src L_movw_src endif endif endm macro X_lpm dst = r0

Definicja w linii 236 pliku macros.h.

7.12.3.2 Registers and are inhibited as X_lpm Z dst endif lpm if L_lpm_dst mov L_lpm_dst

Wartość początkowa:

```
.L_lpm_n
   .endif
   .L_lpm_n = .L_lpm_n + 1
 .endr

 .L_lpm_n = 0
 .irp  reg
```

Definicja w linii 247 pliku macros.h.

7.12.3.3 Invalid dst arg of X_lpm macro endif Z ifc src L_lpm_src

Wartość początkowa:

```
-1
 .L_lpm_n = 0
 .irp  reg
```

Definicja w linii 278 pliku macros.h.

7.12.3.4 Invalid src arg of X_lpm macro endif if L_lpm_src< 2.if.L_lpm_dst==0lpm.elselpmmov.L_lpm_dst, r0.endif.else.if(.L_lpm_dst >

Wartość początkowa:

```
30)
     .err
```

Definicja w linii 304 pliku macros.h.

7.12.3.5 macro X_movw dst src r31 ifc dst src R31 ifc dst L_movw_dst

Wartość początkowa:

```
-1
   .L_movw_src = -1
   .L_movw_n = 0
   .irp  reg
```

Definicja w linii 147 pliku macros.h.

7.12.3.6 macro X_movw dst src r31 ifc dst src R31 ifc dst src L_movw_src

Wartość początkowa:

```
.L_movw_n
       .endif
       .L_movw_n = .L_movw_n + 1
   .endr
   .L_movw_n = 0
   .irp  reg
```

Definicja w linii 158 pliku macros.h.

7.12.3.7 Invalid X_movw arg endif L_movw_src L_movw_src else L_movw_src L_movw_src endif else L_movw_src L_movw_src endif endif endm macro X_lpm r0

Definicja w linii 147 pliku macros.h.

7.12.3.8 Invalid X_movw arg endif L_movw_src L_movw_src else L_movw_src L_movw_src endif else L_movw_src L_movw_src endif endif endm macro X_lpm r31 ifc dst R0

Definicja w linii 158 pliku macros.h.

7.12.3.9 Invalid X_movw arg endif L_movw_src L_movw_src else L_movw_src L_movw_src endif else L_movw_src L_movw_src endif endif endm macro X_lpm r1

Definicja w linii 147 pliku macros.h.

7.12.3.10 Invalid X_movw arg endif L_movw_src L_movw_src else L_movw_src L_movw_src endif else L_movw_src L_movw_src endif endif endm macro X_lpm r31 ifc dst R1

Definicja w linii 158 pliku macros.h.

7.12.3.11 Invalid X_movw arg endif L_movw_src L_movw_src else L_movw_src L_movw_src endif else L_movw_src L_movw_src endif endif endm macro X_lpm r10

Definicja w linii 147 pliku macros.h.

7.12.3.12 Invalid X_movw arg endif L_movw_src L_movw_src else L_movw_src L_movw_src endif else L_movw_src L_movw_src endif endif endm macro X_lpm r31 ifc dst R10

Definicja w linii 158 pliku macros.h.

7.12.3.13 Invalid X_movw arg endif L_movw_src L_movw_src else L_movw_src L_movw_src endif else L_movw_src L_movw_src endif endif endm macro X_lpm r11

Definicja w linii 147 pliku macros.h.

7.12.3.14 Invalid X_movw arg endif L_movw_src L_movw_src else L_movw_src L_movw_src endif else L_movw_src L_movw_src endif endif endm macro X_lpm r31 ifc dst R11

Definicja w linii 158 pliku macros.h.

7.12.3.15 Invalid X_movw arg endif L_movw_src L_movw_src else L_movw_src L_movw_src endif else L_movw_src L_movw_src endif endif endm macro X_lpm r12

Definicja w linii 147 pliku macros.h.

7.12.3.16 Invalid X_movw arg endif L_movw_src L_movw_src else L_movw_src L_movw_src endif else L_movw_src L_movw_src endif endif endm macro X_lpm r31 ifc dst R12

Definicja w linii 158 pliku macros.h.

7.12.3.17 Invalid X_movw arg endif L_movw_src L_movw_src else L_movw_src L_movw_src endif else L_movw_src L_movw_src endif endif endm macro X_lpm r13

Definicja w linii 147 pliku macros.h.

7.12.3.18 Invalid X_movw arg endif L_movw_src L_movw_src else L_movw_src L_movw_src endif else L_movw_src L_movw_src endif endif endm macro X_lpm r31 ifc dst R13

Definicja w linii 158 pliku macros.h.

7.12.3.19 Invalid X_movw arg endif L_movw_src L_movw_src else L_movw_src L_movw_src endif else L_movw_src L_movw_src endif endif endm macro X_lpm r14

Definicja w linii 147 pliku macros.h.

7.12.3.20 Invalid X_movw arg endif L_movw_src L_movw_src else L_movw_src L_movw_src endif else L_movw_src L_movw_src endif endif endm macro X_lpm r31 ifc dst R14

Definicja w linii 158 pliku macros.h.

7.12.3.21 Invalid X_movw arg endif L_movw_src L_movw_src else L_movw_src L_movw_src endif else L_movw_src L_movw_src endif endif endm macro X_lpm r15

Definicja w linii 147 pliku macros.h.

7.12.3.22 Invalid X_movw arg endif L_movw_src L_movw_src else L_movw_src L_movw_src endif else L_movw_src L_movw_src endif endif endm macro X_lpm r31 ifc dst R15

Definicja w linii 158 pliku macros.h.

7.12.3.23 Invalid X_movw arg endif L_movw_src L_movw_src else L_movw_src L_movw_src endif else L_movw_src L_movw_src endif endif endm macro X_lpm r16

Definicja w linii 147 pliku macros.h.

7.12.3.24 Invalid X_movw arg endif L_movw_src L_movw_src else L_movw_src L_movw_src endif else L_movw_src L_movw_src endif endif endm macro X_lpm r31 ifc dst R16

Definicja w linii 158 pliku macros.h.

7.12.3.25 Invalid X_movw arg endif L_movw_src L_movw_src else L_movw_src L_movw_src endif else L_movw_src L_movw_src endif endif endm macro X_lpm r17

Definicja w linii 147 pliku macros.h.

7.12.3.26 Invalid X_movw arg endif L_movw_src L_movw_src else L_movw_src L_movw_src endif else L_movw_src L_movw_src endif endif endm macro X_lpm r31 ifc dst R17

Definicja w linii 158 pliku macros.h.

7.12.3.27 Invalid X_movw arg endif L_movw_src L_movw_src else L_movw_src L_movw_src endif else L_movw_src L_movw_src endif endif endm macro X_lpm r18

Definicja w linii 147 pliku macros.h.

7.12.3.28 Invalid X_movw arg endif L_movw_src L_movw_src else L_movw_src L_movw_src endif else L_movw_src L_movw_src endif endif endm macro X_lpm r31 ifc dst R18

Definicja w linii 158 pliku macros.h.

7.12.3.29 Invalid X_movw arg endif L_movw_src L_movw_src else L_movw_src L_movw_src endif else L_movw_src L_movw_src endif endif endm macro X_lpm r19

Definicja w linii 147 pliku macros.h.

7.12.3.30 Invalid X_movw arg endif L_movw_src L_movw_src else L_movw_src L_movw_src endif else L_movw_src L_movw_src endif endif endm macro X_lpm r31 ifc dst R19

Definicja w linii 158 pliku macros.h.

7.12.3.31 Invalid X_movw arg endif L_movw_src L_movw_src else L_movw_src L_movw_src endif else L_movw_src L_movw_src endif endif endm macro X_lpm r2

Definicja w linii 147 pliku macros.h.

7.12.3.32 Invalid X_movw arg endif L_movw_src L_movw_src else L_movw_src L_movw_src endif else L_movw_src L_movw_src endif endif endm macro X_lpm r31 ifc dst R2

Definicja w linii 158 pliku macros.h.

7.12.3.33 Invalid X_movw arg endif L_movw_src L_movw_src else L_movw_src L_movw_src endif else L_movw_src L_movw_src endif endif endm macro X_lpm r20

Definicja w linii 147 pliku macros.h.

7.12.3.34 Invalid X_movw arg endif L_movw_src L_movw_src else L_movw_src L_movw_src endif else L_movw_src L_movw_src endif endif endm macro X_lpm r31 ifc dst R20

Definicja w linii 158 pliku macros.h.

7.12.3.35 Invalid X_movw arg endif L_movw_src L_movw_src else L_movw_src L_movw_src endif else L_movw_src L_movw_src endif endif endm macro X_lpm r21

Definicja w linii 147 pliku macros.h.

7.12.3.36 Invalid X_movw arg endif L_movw_src L_movw_src else L_movw_src L_movw_src endif else L_movw_src L_movw_src endif endif endm macro X_lpm r31 ifc dst R21

Definicja w linii 158 pliku macros.h.

7.12.3.37 Invalid X_movw arg endif L_movw_src L_movw_src else L_movw_src L_movw_src endif else L_movw_src L_movw_src endif endif endm macro X_lpm r22

Definicja w linii 147 pliku macros.h.

7.12.3.38 Invalid X_movw arg endif L_movw_src L_movw_src else L_movw_src L_movw_src endif else L_movw_src L_movw_src endif endif endm macro X_lpm r31 ifc dst R22

Definicja w linii 158 pliku macros.h.

7.12.3.39 Invalid X_movw arg endif L_movw_src L_movw_src else L_movw_src L_movw_src endif else L_movw_src L_movw_src endif endif endm macro X_lpm r23

Definicja w linii 147 pliku macros.h.

7.12.3.40 Invalid X_movw arg endif L_movw_src L_movw_src else L_movw_src L_movw_src endif else L_movw_src L_movw_src endif endif endm macro X_lpm r31 ifc dst R23

Definicja w linii 158 pliku macros.h.

7.12.3.41 Invalid X_movw arg endif L_movw_src L_movw_src else L_movw_src L_movw_src endif else L_movw_src L_movw_src endif endif endm macro X_lpm r24

Definicja w linii 147 pliku macros.h.

7.12.3.42 Invalid X_movw arg endif L_movw_src L_movw_src else L_movw_src L_movw_src endif else L_movw_src L_movw_src endif endif endm macro X_lpm r31 ifc dst R24

Definicja w linii 158 pliku macros.h.

7.12.3.43 Invalid X_movw arg endif L_movw_src L_movw_src else L_movw_src L_movw_src endif else L_movw_src L_movw_src endif endif endm macro X_lpm r25

Definicja w linii 147 pliku macros.h.

7.12.3.44 Invalid X_movw arg endif L_movw_src L_movw_src else L_movw_src L_movw_src endif else L_movw_src L_movw_src endif endif endm macro X_lpm r31 ifc dst R25

Definicja w linii 158 pliku macros.h.

7.12.3.45 Invalid X_movw arg endif L_movw_src L_movw_src else L_movw_src L_movw_src endif else L_movw_src L_movw_src endif endif endm macro X_lpm r26

Definicja w linii 147 pliku macros.h.

7.12.3.46 Invalid X_movw arg endif L_movw_src L_movw_src else L_movw_src L_movw_src endif else L_movw_src L_movw_src endif endif endm macro X_lpm r31 ifc dst R26

Definicja w linii 158 pliku macros.h.

7.12.3.47 Invalid X_movw arg endif L_movw_src L_movw_src else L_movw_src L_movw_src endif else L_movw_src L_movw_src endif endif endm macro X_lpm r27

Definicja w linii 147 pliku macros.h.

7.12.3.48 Invalid X_movw arg endif L_movw_src L_movw_src else L_movw_src L_movw_src endif else L_movw_src L_movw_src endif endif endm macro X_lpm r31 ifc dst R27

Definicja w linii 158 pliku macros.h.

7.12.3.49 Invalid X_movw arg endif L_movw_src L_movw_src else L_movw_src L_movw_src endif else L_movw_src L_movw_src endif endif endm macro X_lpm r28

Definicja w linii 147 pliku macros.h.

7.12.3.50 Invalid X_movw arg endif L_movw_src L_movw_src else L_movw_src L_movw_src endif else L_movw_src L_movw_src endif endif endm macro X_lpm r31 ifc dst R28

Definicja w linii 158 pliku macros.h.

7.12.3.51 Invalid X_movw arg endif L_movw_src L_movw_src else L_movw_src L_movw_src endif else L_movw_src L_movw_src endif endif endm macro X_lpm r29

Definicja w linii 147 pliku macros.h.

7.12.3.52 Invalid X_movw arg endif L_movw_src L_movw_src else L_movw_src L_movw_src endif else L_movw_src L_movw_src endif endif endm macro X_lpm r31 ifc dst R29

Definicja w linii 158 pliku macros.h.

7.12.3.53 Invalid X_movw arg endif L_movw_src L_movw_src else L_movw_src L_movw_src endif else L_movw_src L_movw_src endif endif endm macro X_lpm r3

Definicja w linii 147 pliku macros.h.

7.12.3.54 Invalid X_movw arg endif L_movw_src L_movw_src else L_movw_src L_movw_src endif else L_movw_src L_movw_src endif endif endm macro X_lpm r31 ifc dst R3

Definicja w linii 158 pliku macros.h.

7.12.3.55 Registers and are inhibited as X_lpm Z dst endif lpm if L_lpm_dst mov r0 endif adiw endif endm macro LPM_R0_ZPLUS_INIT hhi endm macro LPM_R0_ZPLUS_NEXT hhi lpm adiw r30

Definicja w linii 147 pliku macros.h.

7.12.3.56 Invalid X_movw arg endif L_movw_src L_movw_src else L_movw_src L_movw_src endif else L_movw_src L_movw_src endif endif endm macro X_lpm r31 ifc dst R30

Definicja w linii 158 pliku macros.h.

7.12.3.57 Invalid X_movw arg endif L_movw_src L_movw_src else L_movw_src L_movw_src endif else L_movw_src L_movw_src endif endif endm macro X_lpm r4

Definicja w linii 147 pliku macros.h.

7.12.3.58 Invalid X_movw arg endif L_movw_src L_movw_src else L_movw_src L_movw_src endif else L_movw_src L_movw_src endif endif endm macro X_lpm r31 ifc dst R4

Definicja w linii 158 pliku macros.h.

7.12.3.59 Invalid X_movw arg endif L_movw_src L_movw_src else L_movw_src L_movw_src endif else L_movw_src L_movw_src endif endif endm macro X_lpm r5

Definicja w linii 147 pliku macros.h.

7.12.3.60 Invalid X_movw arg endif L_movw_src L_movw_src else L_movw_src L_movw_src endif else L_movw_src L_movw_src endif endif endm macro X_lpm r31 ifc dst R5

Definicja w linii 158 pliku macros.h.

7.12.3.61 Invalid X_movw arg endif L_movw_src L_movw_src else L_movw_src L_movw_src endif else L_movw_src L_movw_src endif endif endm macro X_lpm r6

Definicja w linii 147 pliku macros.h.

7.12.3.62 Invalid X_movw arg endif L_movw_src L_movw_src else L_movw_src L_movw_src endif else L_movw_src L_movw_src endif endif endm macro X_lpm r31 ifc dst R6

Definicja w linii 158 pliku macros.h.

7.12.3.63 Invalid X_movw arg endif L_movw_src L_movw_src else L_movw_src L_movw_src endif else L_movw_src L_movw_src endif endif endm macro X_lpm r7

Definicja w linii 147 pliku macros.h.

7.12.3.64 Invalid X_movw arg endif L_movw_src L_movw_src else L_movw_src L_movw_src endif else L_movw_src L_movw_src endif endif endm macro X_lpm r31 ifc dst R7

Definicja w linii 158 pliku macros.h.

7.12.3.65 Invalid X_movw arg endif L_movw_src L_movw_src else L_movw_src L_movw_src endif else L_movw_src L_movw_src endif endif endm macro X_lpm r8

Definicja w linii 147 pliku macros.h.

7.12.3.66 Invalid X_movw arg endif L_movw_src L_movw_src else L_movw_src L_movw_src endif else L_movw_src L_movw_src endif endif endm macro X_lpm r31 ifc dst R8

Definicja w linii 158 pliku macros.h.

7.12.3.67 Invalid X_movw arg endif L_movw_src L_movw_src else L_movw_src L_movw_src endif else L_movw_src L_movw_src endif endif endm macro X_lpm r9

Definicja w linii 147 pliku macros.h.

7.12.3.68 Invalid X_movw arg endif L_movw_src L_movw_src else L_movw_src L_movw_src endif else L_movw_src L_movw_src endif endif endm macro X_lpm r31 ifc dst R9

Definicja w linii 158 pliku macros.h.

7.12.3.69 Invalid dst arg of X_lpm macro endif Z ifc reg

Definicja w linii 147 pliku macros.h.

7.12.3.70 Invalid X_movw arg endif L_movw_src L_movw_src else L_movw_src L_movw_src endif else L_movw_src L_movw_src endif endif endm macro X_lpm src

Wartość początkowa:

```
Z


  .L_lpm_dst = -1

  .L_lpm_n = 0
  .irp  reg
```

Definicja w linii 236 pliku macros.h.

7.12.3.71 Invalid dst arg of X_lpm macro endif z

Definicja w linii 278 pliku macros.h.

7.12.3.72 Invalid dst arg of X_lpm macro endif Z

Definicja w linii 278 pliku macros.h.

7.13 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)

7.14 Dokumentacja pliku math.c

```
#include <inttypes.h>
#include "openservo.h"
#include "config.h"
#include "math.h"
```

7.15 Dokumentacja pliku math.h

7.16 Dokumentacja pliku motion.c

```
#include <stdint.h>
#include "openservo.h"
#include "config.h"
#include "curve.h"
#include "motion.h"
#include "registers.h"
```

Struktury danych

- struct motion_key

Definicje typów

- typedef struct motion_key motion_key

Funkcje

- void motion_init (void)
- void motion_reset (int16_t position)
- void motion_registers_reset (void)
- uint8_t motion_append (void)
- void motion_next (uint16_t delta)
- uint8_t motion_buffer_left (void)

Zmienne

- uint8_t motion_head
- uint8_t motion_tail
- uint32_t motion_counter
- uint32_t motion_duration

7.16.1 Dokumentacja definicji typów

7.16.1.1 typedef struct motion_key motion_key

7.16.2 Dokumentacja funkcji

7.16.2.1 uint8_t motion_append (void)

Definicja w linii 156 pliku motion.c.

Odwołuje się do curve_init(), motion_key::delta, motion_key::in_velocity, MOTION_BUFFER_MASK, motion_duration, motion_head, motion_registers_reset(), motion_tail, motion_key::out_velocity, motion_key::position, REG_CURVE_DELTA_HI, REG_CURVE_DELTA_LO, REG_CURVE_IN_VELOCITY_HI, REG_CURVE_IN_VELOCITY_LO, REG_CURVE_OUT_VELOCITY_HI, REG_CURVE_OUT_VELOCITY_LO, REG_CURVE_POSITION_HI, REG_CURVE_POSITION_LO i registers_read_word().

Odwołania w main() i RS485CMD().

```
{
    int16_t position;
    int16_t in_velocity;
    int16_t out_velocity;
    uint8_t next;
    uint16_t delta;

    // Get the next index in the buffer.
    next = (motion_head + 1) & MOTION_BUFFER_MASK;

    // Return error if we have looped the head to the tail and the buffer is
       filled.
    if (next == motion_tail) return 0;

    // Get the position, velocity and time delta values from the registers.
    position = (int16_t) registers_read_word(
      REG_CURVE_POSITION_HI, REG_CURVE_POSITION_LO
      );
    in_velocity = (int16_t) registers_read_word(
      REG_CURVE_IN_VELOCITY_HI, REG_CURVE_IN_VELOCITY_LO
      );
    out_velocity = (int16_t) registers_read_word(
      REG_CURVE_OUT_VELOCITY_HI, REG_CURVE_OUT_VELOCITY_LO
      );
    delta = (uint16_t) registers_read_word(
      REG_CURVE_DELTA_HI, REG_CURVE_DELTA_LO);

    // Keypoint delta must be greater than zero.
    if (delta < 1) return 0;

    // Fill in the next keypoint.
    keys[next].delta = delta;
    keys[next].position = int_to_float(position);
    keys[next].in_velocity = fixed_to_float(in_velocity);
    keys[next].out_velocity = fixed_to_float(out_velocity);

    // Is this keypoint being added to an empty buffer?
    if (motion_tail == motion_head)
    {
        // Initialize a new hermite curve that gets us from the current
       position to the new position.
        // We use a velocity of zero at each end to smoothly transition from
       one to the other.
        curve_init(0, delta, curve_get_p1(), keys[next].position, 0.0
      , 0.0);
    }

    // Increase the duration of the buffer.
    motion_duration += delta;

    // Set the new head index.
    motion_head = next;

    // Reset the motion registers and update the buffer status.
    motion_registers_reset();

    return 1;
}
```

7.16.2.2 uint8_t motion_buffer_left (void)

Definicja w linii 292 pliku motion.c.

Odwołuje się do MOTION_BUFFER_SIZE, motion_head i motion_tail.

Odwołania w motion_next() i motion_registers_reset().

```
{
    uint8_t space_left;

    // Determine the points left to store curve data.
    if (motion_head < motion_tail)
    {
        space_left = (MOTION_BUFFER_SIZE - 1) - (
      MOTION_BUFFER_SIZE + motion_head - motion_tail
      );
    }
    else
    {
        space_left = (MOTION_BUFFER_SIZE - 1) - (motion_head
       - motion_tail);
    }

    return space_left;
}
```

7.16.2.3 void motion_init (void)

Definicja w linii 86 pliku motion.c.

Odwołuje się do curve_init(), motion_key::delta, motion_key::in_velocity, motion_counter, motion_duration, motion_head, motion_registers_reset(), motion_tail, motion_key::out_velocity i motion_key::position.

Odwołania w main().

```
{
    // Initialize the counter.
    motion_counter = 0;

    // Initialize the duration.
    motion_duration = 0;

    // Initialize the queue.
    motion_head = 0;
    motion_tail = 0;

    // Initialize the keypoint.
    keys[0].delta = 0;
    keys[0].position = 512.0;
    keys[0].in_velocity = 0.0;
    keys[0].out_velocity = 0.0;

    // Initialize an empty hermite curve at the center servo position.
    curve_init(0, 0, 512.0, 512.0, 0.0, 0.0);

    // Reset the registers.
    motion_registers_reset();
}
```

7.16.2.4 void motion_next (uint16_t *delta*)

Definicja w linii 210 pliku motion.c.

Odwołuje się do curve_init(), curve_solve(), FLAGS_LO_MOTION_ENABLED, motion_buffer_left(), MOTION_BUFFER_MASK, motion_counter, motion_duration, motion_head, motion_tail, REG_CURVE_BUFFER, REG_FLAGS_LO, REG_SEEK_POSITION_HI, REG_SEEK_POSITION_LO, REG_SEEK_VELOCITY_HI, REG_SEEK_VELOCITY_LO i registers_write_word().

Odwołania w main().

```
{
    float fposition;
    float fvelocity;

    // Determine if curve motion is disabled in the registers.
    if (!(registers_read_byte(REG_FLAGS_LO) & (1<<
      FLAGS_LO_MOTION_ENABLED))) return;

    // Are we processing an empty curve?
    if (motion_tail == motion_head)
    {
        // Yes. Keep the counter and duration at zero.
        motion_counter = 0;
        motion_duration = 0;
    }
    else
    {
        // Increment the counter.
        motion_counter += delta;

        // Have we exceeded the duration of the currently buffered curve?
        while (motion_counter > curve_get_duration())
        {
            // Reduce the buffer counter by the currently buffered curve
       duration.
            motion_counter -= curve_get_duration();

            // Reduce the buffer duration by the currently buffered curve
       duration.
            motion_duration -= curve_get_duration();

            // Increment the tail to process the next buffered curve.
            motion_tail = (motion_tail + 1) &
      MOTION_BUFFER_MASK;

            // Has the tail caught up with the head?
            if (motion_tail == motion_head)
            {
                // Initialize an empty hermite curve with a zero duration.
       This is a degenerate case for
                // the hermite cuve that will always return the position of the
       curve without velocity.
                curve_init(0, 0, keys[motion_head].
      position, keys[motion_head].position, 0.0, 0.0);

                // Reset the buffer counter and duration to zero.
                motion_counter = 0;
                motion_duration = 0;
            }
            else
            {
                uint8_t curr_point;
                uint8_t next_point;

                // Get the current point and next point for the curve.
                curr_point = motion_tail;
                next_point = (curr_point + 1) & MOTION_BUFFER_MASK
      ;
```

```
                // Initialize the hermite curve from the current and next
       point.
                curve_init(0, keys[next_point].delta,
                           keys[curr_point].position, keys[next_point].position
      ,
                           keys[curr_point].out_velocity, keys[next_point].
      in_velocity);
            }

            // Update the space available in the buffer.
            registers_write_byte(REG_CURVE_BUFFER,
      motion_buffer_left());
        }
    }

    // Get the position and velocity from the hermite curve.
    curve_solve(motion_counter, &fposition, &fvelocity
      );

    // The velocity is in position units a millisecond, but we really need the
    // velocity to be measured in position units every 10 milliseconds to match
    // the sample period of the ADC.
    fvelocity *= 10.0;

    // Update the seek position register.
    registers_write_word(REG_SEEK_POSITION_HI
      , REG_SEEK_POSITION_LO, float_to_int(fposition));

    // Update the seek velocity register.
    registers_write_word(REG_SEEK_VELOCITY_HI
      , REG_SEEK_VELOCITY_LO, float_to_int(fvelocity));
}
```

7.16.2.5 void motion_registers_reset (void)

Definicja w linii 141 pliku motion.c.

Odwołuje się do motion_buffer_left(), REG_CURVE_BUFFER, REG_CURVE_DELTA_HI, REG_CURVE_DELTA_LO, REG_CURVE_IN_VELOCITY_HI, REG_CURVE_IN_VELOCITY_LO, REG_CURVE_OUT_VELOCITY_HI, REG_CURVE_OUT_VELOCITY_LO, REG_CURVE_POSITION_HI, REG_CURVE_POSITION_LO, REG_CURVE_RESERVED i registers_write_word().

Odwołania w motion_append(), motion_init() i motion_reset().

```
{
    // Set the default position, velocity and delta data.
    registers_write_word(REG_CURVE_POSITION_HI
      , REG_CURVE_POSITION_LO, 0);
    registers_write_word(REG_CURVE_IN_VELOCITY_HI
      , REG_CURVE_IN_VELOCITY_LO, 0);
    registers_write_word(REG_CURVE_OUT_VELOCITY_HI
      , REG_CURVE_OUT_VELOCITY_LO, 0);
    registers_write_word(REG_CURVE_DELTA_HI
      , REG_CURVE_DELTA_LO, 0);

    // Update the buffer status.
    registers_write_byte(REG_CURVE_RESERVED, 0);
    registers_write_byte(REG_CURVE_BUFFER, motion_buffer_left
      ());
}
```

7.16.2.6 void motion_reset (int16_t *position*)

Definicja w linii 113 pliku motion.c.

Odwołuje się do curve_init(), motion_key::delta, motion_key::in_velocity, motion_counter, motion_duration, motion_head, motion_registers_reset(), motion_tail, motion_key::out_velocity i motion_key::position.

Odwołania w main() i RS485CMD().

```
{
    // Reset the counter.
    motion_counter = 0;

    // Reset the duration.
    motion_duration = 0;
    // Reset the queue.
    motion_head = 0;
    motion_tail = 0;

    // Reset the keypoint.
    keys[0].delta = 0;
    keys[0].position = int_to_float(position);
    keys[0].in_velocity = 0.0;
    keys[0].out_velocity = 0.0;

    // Initialize an empty hermite curve.  This is a degenerate case for the
       hermite
    // curve that will always return the position of the curve without
       velocity.
    curve_init(0, 0, keys[0].position, keys[0].position, 0.0, 0.0);

    // Reset the registers.
    motion_registers_reset();
}
```

7.16.3 Dokumentacja zmiennych

7.16.3.1 uint32_t motion_counter

Definicja w linii 50 pliku motion.c.

Odwołania w motion_init(), motion_next() i motion_reset().

7.16.3.2 uint32_t motion_duration

Definicja w linii 51 pliku motion.c.

Odwołania w motion_append(), motion_init(), motion_next() i motion_reset().

7.16.3.3 uint8_t motion_head

Definicja w linii 48 pliku motion.c.

Odwołania w motion_append(), motion_buffer_left(), motion_init(), motion_next() i motion_reset().

7.16.3.4 uint8_t motion_tail

Definicja w linii 49 pliku motion.c.

Odwołania w motion_append(), motion_buffer_left(), motion_init(), motion_next() i motion_reset().

7.17 Dokumentacja pliku motion.h

```
#include "registers.h"
```

Definicje

- #define MOTION_BUFFER_SIZE 8
- #define MOTION_BUFFER_MASK (MOTION_BUFFER_SIZE - 1)

Funkcje

- void motion_init (void)
- void motion_reset (int16_t position)
- void motion_registers_reset (void)
- uint8_t motion_append (void)
- void motion_next (uint16_t delta)
- uint8_t motion_buffer_left (void)

Zmienne

- uint8_t motion_head
- uint8_t motion_tail
- uint32_t motion_counter
- uint32_t motion_duration

7.17.1 Dokumentacja definicji

7.17.1.1 #define MOTION_BUFFER_MASK (MOTION_BUFFER_SIZE - 1)

Definicja w linii 36 pliku motion.h.

Odwołania w motion_append() i motion_next().

7.17.1.2 #define MOTION_BUFFER_SIZE 8

Definicja w linii 35 pliku motion.h.

Odwołania w motion_buffer_left().

7.17.2 Dokumentacja funkcji

7.17.2.1 uint8_t motion_append (void)

Definicja w linii 156 pliku motion.c.

Odwołuje się do curve_init(), motion_key::delta, motion_key::in_velocity, MOTION_BUFFER_MASK, motion_duration, motion_head, motion_registers_reset(), motion_tail, motion_key::out_velocity, motion_key::position, REG_CURVE_DELTA_HI, REG_CURVE_DELTA_LO, REG_CURVE_IN_VELOCITY_HI, REG_CURVE_IN_VELOCITY_LO, REG_CURVE_OUT_VELOCITY_HI, REG_CURVE_OUT_VELOCITY_LO, REG_CURVE_POSITION_HI, REG_CURVE_POSITION_LO i registers_read_word().

Odwołania w main() i RS485CMD().

```
{
    int16_t position;
    int16_t in_velocity;
    int16_t out_velocity;
    uint8_t next;
    uint16_t delta;

    // Get the next index in the buffer.
    next = (motion_head + 1) & MOTION_BUFFER_MASK;

    // Return error if we have looped the head to the tail and the buffer is
       filled.
    if (next == motion_tail) return 0;

    // Get the position, velocity and time delta values from the registers.
    position = (int16_t) registers_read_word(
      REG_CURVE_POSITION_HI, REG_CURVE_POSITION_LO
      );
    in_velocity = (int16_t) registers_read_word(
      REG_CURVE_IN_VELOCITY_HI, REG_CURVE_IN_VELOCITY_LO
      );
    out_velocity = (int16_t) registers_read_word(
      REG_CURVE_OUT_VELOCITY_HI, REG_CURVE_OUT_VELOCITY_LO
      );
    delta = (uint16_t) registers_read_word(
      REG_CURVE_DELTA_HI, REG_CURVE_DELTA_LO);

    // Keypoint delta must be greater than zero.
    if (delta < 1) return 0;

    // Fill in the next keypoint.
    keys[next].delta = delta;
    keys[next].position = int_to_float(position);
    keys[next].in_velocity = fixed_to_float(in_velocity);
    keys[next].out_velocity = fixed_to_float(out_velocity);
    // Is this keypoint being added to an empty buffer?
    if (motion_tail == motion_head)
    {
        // Initialize a new hermite curve that gets us from the current
       position to the new position.
        // We use a velocity of zero at each end to smoothly transition from
       one to the other.
        curve_init(0, delta, curve_get_p1(), keys[next].position, 0.0
      , 0.0);
    }

    // Increase the duration of the buffer.
    motion_duration += delta;

    // Set the new head index.
    motion_head = next;

    // Reset the motion registers and update the buffer status.
    motion_registers_reset();

    return 1;
}
```

7.17.2.2 uint8_t motion_buffer_left (void)

Definicja w linii 292 pliku motion.c.

Odwołuje się do MOTION_BUFFER_SIZE, motion_head i motion_tail.

Odwołania w motion_next() i motion_registers_reset().

```
{
    uint8_t space_left;

    // Determine the points left to store curve data.
    if (motion_head < motion_tail)
    {
        space_left = (MOTION_BUFFER_SIZE - 1) - (
      MOTION_BUFFER_SIZE + motion_head - motion_tail
      );
    }
    else
    {
        space_left = (MOTION_BUFFER_SIZE - 1) - (motion_head
       - motion_tail);
    }

    return space_left;
}
```

7.17.2.3 void motion_init (void)

Definicja w linii 86 pliku motion.c.

Odwołuje się do curve_init(), motion_key::delta, motion_key::in_velocity, motion_counter, motion_duration, motion_head, motion_registers_reset(), motion_tail, motion_key::out_velocity i motion_key::position.

Odwołania w main().

```
{
    // Initialize the counter.
    motion_counter = 0;

    // Initialize the duration.
    motion_duration = 0;

    // Initialize the queue.
    motion_head = 0;
    motion_tail = 0;

    // Initialize the keypoint.
    keys[0].delta = 0;
    keys[0].position = 512.0;
    keys[0].in_velocity = 0.0;
    keys[0].out_velocity = 0.0;

    // Initialize an empty hermite curve at the center servo position.
    curve_init(0, 0, 512.0, 512.0, 0.0, 0.0);
    // Reset the registers.
    motion_registers_reset();
}
```

7.17.2.4 void motion_next (uint16_t *delta*)

Definicja w linii 210 pliku motion.c.

Odwołuje się do curve_init(), curve_solve(), FLAGS_LO_MOTION_ENABLED, motion_buffer_left(), MOTION_BUFFER_MASK, motion_counter, motion_duration, motion_head, motion_tail, REG_CURVE_BUFFER, REG_FLAGS_LO, REG_SEEK_POSITION_HI, REG_SEEK_POSITION_LO, REG_SEEK_VELOCITY_HI, REG_SEEK_VELOCITY_LO i registers_write_word().

Odwołania w main().

```
{
    float fposition;
    float fvelocity;

    // Determine if curve motion is disabled in the registers.
    if (!(registers_read_byte(REG_FLAGS_LO) & (1<<
      FLAGS_LO_MOTION_ENABLED))) return;

    // Are we processing an empty curve?
    if (motion_tail == motion_head)
    {
        // Yes. Keep the counter and duration at zero.
        motion_counter = 0;
        motion_duration = 0;
    }
    else
    {
        // Increment the counter.
        motion_counter += delta;

        // Have we exceeded the duration of the currently buffered curve?
        while (motion_counter > curve_get_duration())
        {
            // Reduce the buffer counter by the currently buffered curve
       duration.
            motion_counter -= curve_get_duration();

            // Reduce the buffer duration by the currently buffered curve
       duration.
            motion_duration -= curve_get_duration();

            // Increment the tail to process the next buffered curve.
            motion_tail = (motion_tail + 1) &
      MOTION_BUFFER_MASK;

            // Has the tail caught up with the head?
            if (motion_tail == motion_head)
            {
                // Initialize an empty hermite curve with a zero duration.
       This is a degenerate case for
                // the hermite cuve that will always return the position of the
       curve without velocity.
                curve_init(0, 0, keys[motion_head].
      position, keys[motion_head].position, 0.0, 0.0);

                // Reset the buffer counter and duration to zero.
                motion_counter = 0;
                motion_duration = 0;
            }
            else
            {
                uint8_t curr_point;
                uint8_t next_point;

                // Get the current point and next point for the curve.
                curr_point = motion_tail;
                next_point = (curr_point + 1) & MOTION_BUFFER_MASK
      ;

                // Initialize the hermite curve from the current and next
       point.
                curve_init(0, keys[next_point].delta,
                           keys[curr_point].position, keys[next_point].position
      ,
                           keys[curr_point].out_velocity, keys[next_point].
      in_velocity);
            }
            // Update the space available in the buffer.
            registers_write_byte(REG_CURVE_BUFFER,
      motion_buffer_left());
        }
    }

    // Get the position and velocity from the hermite curve.
    curve_solve(motion_counter, &fposition, &fvelocity
      );

    // The velocity is in position units a millisecond, but we really need the
    // velocity to be measured in position units every 10 milliseconds to match
    // the sample period of the ADC.
    fvelocity *= 10.0;

    // Update the seek position register.
    registers_write_word(REG_SEEK_POSITION_HI
      , REG_SEEK_POSITION_LO, float_to_int(fposition));

    // Update the seek velocity register.
    registers_write_word(REG_SEEK_VELOCITY_HI
      , REG_SEEK_VELOCITY_LO, float_to_int(fvelocity));
}
```

7.17.2.5 void motion_registers_reset (void)

Definicja w linii 141 pliku motion.c.

Odwołuje się do motion_buffer_left(), REG_CURVE_BUFFER, REG_CURVE_DELTA_HI, REG_CURVE_DELTA_LO, REG_CURVE_IN_VELOCITY_HI, REG_CURVE_IN_VELOCITY_LO, REG_CURVE_OUT_VELOCITY_HI, REG_CURVE_OUT_VELOCITY_LO, REG_CURVE_POSITION_HI, REG_CURVE_POSITION_LO, REG_CURVE_RESERVED i registers_write_word().

Odwołania w motion_append(), motion_init() i motion_reset().

```
{
    // Set the default position, velocity and delta data.
    registers_write_word(REG_CURVE_POSITION_HI
      , REG_CURVE_POSITION_LO, 0);
    registers_write_word(REG_CURVE_IN_VELOCITY_HI
      , REG_CURVE_IN_VELOCITY_LO, 0);
    registers_write_word(REG_CURVE_OUT_VELOCITY_HI
      , REG_CURVE_OUT_VELOCITY_LO, 0);
    registers_write_word(REG_CURVE_DELTA_HI
      , REG_CURVE_DELTA_LO, 0);

    // Update the buffer status.
    registers_write_byte(REG_CURVE_RESERVED, 0);
    registers_write_byte(REG_CURVE_BUFFER, motion_buffer_left
      ());
}
```

7.17.2.6 void motion_reset (int16_t *position*)

Definicja w linii 113 pliku motion.c.

Odwołuje się do curve_init(), motion_key::delta, motion_key::in_velocity, motion_counter, motion_duration, motion_head, motion_registers_reset(), motion_tail, motion_key::out_velocity i motion_key::position.

Odwołania w main() i RS485CMD().

```
{
    // Reset the counter.
    motion_counter = 0;

    // Reset the duration.
    motion_duration = 0;

    // Reset the queue.
    motion_head = 0;
    motion_tail = 0;

    // Reset the keypoint.
    keys[0].delta = 0;
    keys[0].position = int_to_float(position);
    keys[0].in_velocity = 0.0;
    keys[0].out_velocity = 0.0;

    // Initialize an empty hermite curve.  This is a degenerate case for the
       hermite
    // curve that will always return the position of the curve without
       velocity.
    curve_init(0, 0, keys[0].position, keys[0].position, 0.0, 0.0);

    // Reset the registers.
    motion_registers_reset();
}
```

7.17.3 Dokumentacja zmiennych

7.17.3.1 uint32_t motion_counter

Definicja w linii 50 pliku motion.c.

Odwołania w motion_init(), motion_next() i motion_reset().

7.17.3.2 uint32_t motion_duration

Definicja w linii 51 pliku motion.c.

Odwołania w motion_append(), motion_init(), motion_next() i motion_reset().

7.17.3.3 uint8_t motion_head

Definicja w linii 48 pliku motion.c.

Odwołania w motion_append(), motion_buffer_left(), motion_init(), motion_next() i motion_reset().

7.17.3.4 uint8_t motion_tail

Definicja w linii 49 pliku motion.c.

Odwołania w motion_append(), motion_buffer_left(), motion_init(), motion_next() i motion_reset().

7.18 Dokumentacja pliku openservo.h

Definicje

- #define OPENSERVO_DEVICE_TYPE 1
- #define OPENSERVO_DEVICE_SUBTYPE 1
- #define SOFTWARE_VERSION_MAJOR 0
- #define SOFTWARE_VERSION_MINOR 2
- #define REG_DEFAULT_TWI_ADDR 0x10
- #define FALSE 0
- #define TRUE -1
- #define NULL 0
- #define enterCritical()
- #define exitCritical()

Definicje typów

- typedef int8_t bool

7.18.1 Dokumentacja definicji

7.18.1.1 #define enterCritical( )

Wartość:

```
__asm__ __volatile__ ("in __tmp_reg__, __SREG__\n\t " \
                                             "push __tmp_reg__\n\t" \
                                             "cli" ::)
```

Definicja w linii 54 pliku openservo.h.

Odwołania w UartStartTx().

7.18.1.2 #define exitCritical( )

Wartość:

```
__asm__ __volatile__ ("pop __tmp_reg__ \n\t" \
                                             "out __SREG__, __tmp_reg__" ::)
```

Definicja w linii 58 pliku openservo.h.

Odwołania w UartStartTx().

7.18.1.3 #define FALSE 0

Definicja w linii 48 pliku openservo.h.

Odwołania w FrameCheckCRC(), GetFrame(), SendFrame(), UartGetFrameISR(), UartPutFrameISR() i UartSetBaud().

7.18.1.4 #define NULL 0

Definicja w linii 52 pliku openservo.h.

7.18.1.5 #define OPENSERVO_DEVICE_SUBTYPE 1

Definicja w linii 33 pliku openservo.h.

Odwołania w registers_init().

7.18.1.6 #define OPENSERVO_DEVICE_TYPE 1

Definicja w linii 32 pliku openservo.h.

Odwołania w registers_init().

7.18.1.7 #define REG_DEFAULT_TWI_ADDR 0x10

Definicja w linii 42 pliku openservo.h.

Odwołania w registers_defaults().

7.18.1.8 #define SOFTWARE_VERSION_MAJOR 0

Definicja w linii 38 pliku openservo.h.

Odwołania w registers_init().

7.18.1.9 #define SOFTWARE_VERSION_MINOR 2

Definicja w linii 39 pliku openservo.h.

Odwołania w registers_init().

7.18.1.10 #define TRUE -1

Definicja w linii 49 pliku openservo.h.

Odwołania w FrameCheckCRC(), GetFrame(), SendFrame(), UartGetFrameISR(), UartInit(), UartInitRs485() i UartSetBaud().

7.18.2 Dokumentacja definicji typów

7.18.2.1 typedef int8_t bool

Definicja w linii 47 pliku openservo.h.

7.19 Dokumentacja pliku pid.c

```
#include <inttypes.h>
#include "openservo.h"
#include "config.h"
#include "pid.h"
#include "registers.h"
```

Definicje

- #define MIN_POSITION (0)
- #define MAX_POSITION (1023)
- #define MAX_OUTPUT (255)
- #define MIN_OUTPUT (-MAX_OUTPUT)
- #define FILTER_SHIFT 1

Funkcje

- void pid_init (void)
- void pid_registers_defaults (void)
- int16_t pid_position_to_pwm (int16_t current_position)

7.19.1 Dokumentacja definicji

7.19.1.1 #define FILTER_SHIFT 1

Definicja w linii 63 pliku pid.c.

7.19.1.2 #define MAX_OUTPUT (255)

Definicja w linii 39 pliku pid.c.

Odwołania w pid_position_to_pwm().

7.19.1.3 #define MAX_POSITION (1023)

Definicja w linii 36 pliku pid.c.

Odwołania w pid_position_to_pwm().

7.19.1.4 #define MIN_OUTPUT (-MAX_OUTPUT)

Definicja w linii 40 pliku pid.c.

Odwołania w pid_position_to_pwm().

7.19.1.5 #define MIN_POSITION (0)

Definicja w linii 35 pliku pid.c.

7.19.2 Dokumentacja funkcji

7.19.2.1 void pid_init (void)

Definicja w linii 76 pliku pid.c.

Odwołania w main().

```
{
    // Initialize preserved values.
    previous_seek = 0;
    previous_position = 0;
}
```

7.19.2.2 int16_t pid_position_to_pwm (int16_t *current_position*)

Definicja w linii 106 pliku pid.c.

Odwołuje się do MAX_OUTPUT, MAX_POSITION, MIN_OUTPUT, REG_MAX_SEEK_HI, REG_MAX_SEEK_LO, REG_MIN_SEEK_HI, REG_MIN_SEEK_LO, REG_PID_DEADBAND, REG_PID_DGAIN_HI, REG_PID_DGAIN_LO, REG_PID_PGAIN_HI, REG_PID_PGAIN_LO, REG_POSITION_HI, REG_POSITION_LO, REG_REVERSE_SEEK, REG_SEEK_POSITION_HI, REG_SEEK_POSITION_LO, REG_SEEK_VELOCITY_HI, REG_SEEK_VELOCITY_LO, REG_VELOCITY_HI, REG_VELOCITY_LO, registers_read_word() i registers_write_word().

Odwołania w main().

```
{
    // We declare these static to keep them off the stack.
    static int16_t deadband;
    static int16_t p_component;
    static int16_t d_component;
    static int16_t seek_position;
    static int16_t seek_velocity;
    static int16_t minimum_position;
    static int16_t maximum_position;
    static int16_t current_velocity;
    static int16_t filtered_position;
    static int32_t pwm_output;
    static uint16_t d_gain;
    static uint16_t p_gain;

    // Filter the current position thru a digital low-pass filter.
    filtered_position = filter_update(current_position);

    // Use the filtered position to determine velocity.
    current_velocity = filtered_position - previous_position;
    previous_position = filtered_position;

    // Get the seek position and velocity.
    seek_position = (int16_t) registers_read_word(
      REG_SEEK_POSITION_HI, REG_SEEK_POSITION_LO
      );
    seek_velocity = (int16_t) registers_read_word(
      REG_SEEK_VELOCITY_HI, REG_SEEK_VELOCITY_LO
      );

    // Get the minimum and maximum position.
    minimum_position = (int16_t) registers_read_word(
      REG_MIN_SEEK_HI, REG_MIN_SEEK_LO);
    maximum_position = (int16_t) registers_read_word(
      REG_MAX_SEEK_HI, REG_MAX_SEEK_LO);

    // Are we reversing the seek sense?
    if (registers_read_byte(REG_REVERSE_SEEK) != 0)
    {
        // Yes. Update the position and velocity using reverse sense.
        registers_write_word(REG_POSITION_HI
      , REG_POSITION_LO, (uint16_t) (MAX_POSITION -
      current_position));
        registers_write_word(REG_VELOCITY_HI
      , REG_VELOCITY_LO, (uint16_t) -current_velocity);

        // Reverse sense the seek and other position values.
        seek_position = MAX_POSITION - seek_position;
        minimum_position = MAX_POSITION - minimum_position;
        maximum_position = MAX_POSITION - maximum_position;
    }
    else
    {
        // No. Update the position and velocity registers without change.
        registers_write_word(REG_POSITION_HI
      , REG_POSITION_LO, (uint16_t) current_position);
        registers_write_word(REG_VELOCITY_HI
      , REG_VELOCITY_LO, (uint16_t) current_velocity);
    }

    // Get the deadband.
    deadband = (int16_t) registers_read_byte(REG_PID_DEADBAND);

    // Use the filtered position when the seek position is not changing.
    if (seek_position == previous_seek) current_position = filtered_position;
    previous_seek = seek_position;

    // Keep the seek position bound within the minimum and maximum position.
    if (seek_position < minimum_position) seek_position = minimum_position;
    if (seek_position > maximum_position) seek_position = maximum_position;

    // The proportional component to the PID is the position error.
    p_component = seek_position - current_position;

    // The derivative component to the PID is the velocity.
    d_component = seek_velocity - current_velocity;

    // Get the proportional, derivative and integral gains.
    p_gain = registers_read_word(REG_PID_PGAIN_HI
      , REG_PID_PGAIN_LO);
    d_gain = registers_read_word(REG_PID_DGAIN_HI
      , REG_PID_DGAIN_LO);

    // Start with zero PWM output.
    pwm_output = 0;

    // Apply proportional component to the PWM output if outside the deadband.
    if ((p_component > deadband) || (p_component < -deadband))
    {
        // Apply the proportional component of the PWM output.
        pwm_output += (int32_t) p_component * (int32_t) p_gain;
    }

    // Apply the derivative component of the PWM output.
    pwm_output += (int32_t) d_component * (int32_t) d_gain;

    // Shift by 8 to account for the multiply by the 8:8 fixed point gain
       values.
    pwm_output >>= 8;

    // Check for output saturation.
    if (pwm_output > MAX_OUTPUT)
    {
        // Can't go higher than the maximum output value.
        pwm_output = MAX_OUTPUT;
    }
    else if (pwm_output < MIN_OUTPUT)
    {
        // Can't go lower than the minimum output value.
        pwm_output = MIN_OUTPUT;
    }

    // Return the PID output.
    return (int16_t) pwm_output;
}
```

7.19.2.3 void pid_registers_defaults (void)

Definicja w linii 85 pliku pid.c.

Odwołuje się do DEFAULT_MAX_SEEK, DEFAULT_MIN_SEEK, DEFAULT_PID_DEADBAND, DEFAULT_PID_DGAIN, DEFAULT_PID_IGAIN, DEFAULT_PID_PGAIN, REG_MAX_SEEK_HI, REG_MAX_SEEK_LO, REG_MIN_SEEK_HI, REG_MIN_SEEK_LO, REG_PID_DEADBAND, REG_PID_DGAIN_HI, REG_PID_DGAIN_LO, REG_PID_IGAIN_HI, REG_PID_IGAIN_LO, REG_PID_PGAIN_HI, REG_PID_PGAIN_LO, REG_REVERSE_SEEK i registers_write_word().

Odwołania w registers_defaults().

```
{
    // Default deadband.
    registers_write_byte(REG_PID_DEADBAND, DEFAULT_PID_DEADBAND
      );
    // Default gain values.
    registers_write_word(REG_PID_PGAIN_HI,
      REG_PID_PGAIN_LO, DEFAULT_PID_PGAIN);
    registers_write_word(REG_PID_DGAIN_HI,
      REG_PID_DGAIN_LO, DEFAULT_PID_DGAIN);
    registers_write_word(REG_PID_IGAIN_HI,
      REG_PID_IGAIN_LO, DEFAULT_PID_IGAIN);

    // Default position limits.
    registers_write_word(REG_MIN_SEEK_HI,
      REG_MIN_SEEK_LO, DEFAULT_MIN_SEEK);
    registers_write_word(REG_MAX_SEEK_HI,
      REG_MAX_SEEK_LO, DEFAULT_MAX_SEEK);

    // Default reverse seek setting.
    registers_write_byte(REG_REVERSE_SEEK, 0x00);
}
```

7.20 Dokumentacja pliku pid.h

Funkcje

- void pid_init (void)
- void pid_registers_defaults (void)
- int16_t pid_position_to_pwm (int16_t position)

7.20.1 Dokumentacja funkcji

7.20.1.1 void pid_init (void)

Definicja w linii 76 pliku pid.c.

Odwołania w main().

```
{
    // Initialize preserved values.
    previous_seek = 0;
    previous_position = 0;
}
```

7.20.1.2 int16_t pid_position_to_pwm (int16_t *position*)

Definicja w linii 106 pliku pid.c.

Odwołuje się do MAX_OUTPUT, MAX_POSITION, MIN_OUTPUT, REG_MAX_SEEK_HI, REG_MAX_SEEK_LO, REG_MIN_SEEK_HI, REG_MIN_SEEK_LO, REG_PID_DEADBAND, REG_PID_DGAIN_HI, REG_PID_DGAIN_LO, REG_PID_PGAIN_HI, REG_PID_PGAIN_LO, REG_POSITION_HI, REG_POSITION_LO, REG_REVERSE_SEEK, REG_SEEK_POSITION_HI, REG_SEEK_POSITION_LO, REG_SEEK_VELOCITY_HI, REG_SEEK_VELOCITY_LO, REG_VELOCITY_HI, REG_VELOCITY_LO, registers_read_word() i registers_write_word().

Odwołania w main().

```
{
    // We declare these static to keep them off the stack.
    static int16_t deadband;
    static int16_t p_component;
    static int16_t d_component;
    static int16_t seek_position;
    static int16_t seek_velocity;
    static int16_t minimum_position;
    static int16_t maximum_position;
    static int16_t current_velocity;
    static int16_t filtered_position;
    static int32_t pwm_output;
    static uint16_t d_gain;
    static uint16_t p_gain;

    // Filter the current position thru a digital low-pass filter.
    filtered_position = filter_update(current_position);

    // Use the filtered position to determine velocity.
```

```
current_velocity = filtered_position - previous_position;
previous_position = filtered_position;

// Get the seek position and velocity.
seek_position = (int16_t) registers_read_word(
  REG_SEEK_POSITION_HI, REG_SEEK_POSITION_LO
  );
seek_velocity = (int16_t) registers_read_word(
  REG_SEEK_VELOCITY_HI, REG_SEEK_VELOCITY_LO
  );

// Get the minimum and maximum position.
minimum_position = (int16_t) registers_read_word(
  REG_MIN_SEEK_HI, REG_MIN_SEEK_LO);
maximum_position = (int16_t) registers_read_word(
  REG_MAX_SEEK_HI, REG_MAX_SEEK_LO);

// Are we reversing the seek sense?
if (registers_read_byte(REG_REVERSE_SEEK) != 0)
{
    // Yes. Update the position and velocity using reverse sense.
    registers_write_word(REG_POSITION_HI
  , REG_POSITION_LO, (uint16_t) (MAX_POSITION -
  current_position));
    registers_write_word(REG_VELOCITY_HI
  , REG_VELOCITY_LO, (uint16_t) -current_velocity);

    // Reverse sense the seek and other position values.
    seek_position = MAX_POSITION - seek_position;
    minimum_position = MAX_POSITION - minimum_position;
    maximum_position = MAX_POSITION - maximum_position;
}
else
{
    // No. Update the position and velocity registers without change.
    registers_write_word(REG_POSITION_HI
  , REG_POSITION_LO, (uint16_t) current_position);
    registers_write_word(REG_VELOCITY_HI
  , REG_VELOCITY_LO, (uint16_t) current_velocity);
}

// Get the deadband.
deadband = (int16_t) registers_read_byte(REG_PID_DEADBAND);

// Use the filtered position when the seek position is not changing.
if (seek_position == previous_seek) current_position = filtered_position;
previous_seek = seek_position;

// Keep the seek position bound within the minimum and maximum position.
if (seek_position < minimum_position) seek_position = minimum_position;
if (seek_position > maximum_position) seek_position = maximum_position;

// The proportional component to the PID is the position error.
p_component = seek_position - current_position;

// The derivative component to the PID is the velocity.
d_component = seek_velocity - current_velocity;

// Get the proportional, derivative and integral gains.
p_gain = registers_read_word(REG_PID_PGAIN_HI
  , REG_PID_PGAIN_LO);
d_gain = registers_read_word(REG_PID_DGAIN_HI
  , REG_PID_DGAIN_LO);

// Start with zero PWM output.
pwm_output = 0;

// Apply proportional component to the PWM output if outside the deadband.
if ((p_component > deadband) || (p_component < -deadband))
{
    // Apply the proportional component of the PWM output.
    pwm_output += (int32_t) p_component * (int32_t) p_gain;
}

// Apply the derivative component of the PWM output.
pwm_output += (int32_t) d_component * (int32_t) d_gain;

// Shift by 8 to account for the multiply by the 8:8 fixed point gain
   values.
pwm_output >>= 8;

// Check for output saturation.
if (pwm_output > MAX_OUTPUT)
{
    // Can't go higher than the maximum output value.
    pwm_output = MAX_OUTPUT;
}
```

```
    else if (pwm_output < MIN_OUTPUT)
    {
        // Can't go lower than the minimum output value.
        pwm_output = MIN_OUTPUT;
    }

    // Return the PID output.
    return (int16_t) pwm_output;
}
```

7.20.1.3 void pid_registers_defaults (void)

Definicja w linii 85 pliku pid.c.

Odwołuje się do DEFAULT_MAX_SEEK, DEFAULT_MIN_SEEK, DEFAULT_PID_DEADBAND, DEFAULT_PID_DGAIN, DEFAULT_PID_IGAIN, DEFAULT_PID_PGAIN, REG_MAX_SEEK_HI, REG_MAX_SEEK_LO, REG_MIN_SEEK_HI, REG_MIN_SEEK_LO, REG_PID_DEADBAND, REG_PID_DGAIN_HI, REG_PID_DGAIN_LO, REG_PID_IGAIN_HI, REG_PID_IGAIN_LO, REG_PID_PGAIN_HI, REG_PID_PGAIN_LO, REG_REVERSE_SEEK i registers_write_word().

Odwołania w registers_defaults().

```
{
    // Default deadband.
    registers_write_byte(REG_PID_DEADBAND, DEFAULT_PID_DEADBAND
      );

    // Default gain values.
    registers_write_word(REG_PID_PGAIN_HI,
      REG_PID_PGAIN_LO, DEFAULT_PID_PGAIN);
    registers_write_word(REG_PID_DGAIN_HI,
      REG_PID_DGAIN_LO, DEFAULT_PID_DGAIN);
    registers_write_word(REG_PID_IGAIN_HI,
      REG_PID_IGAIN_LO, DEFAULT_PID_IGAIN);

    // Default position limits.
    registers_write_word(REG_MIN_SEEK_HI,
      REG_MIN_SEEK_LO, DEFAULT_MIN_SEEK);
    registers_write_word(REG_MAX_SEEK_HI,
      REG_MAX_SEEK_LO, DEFAULT_MAX_SEEK);

    // Default reverse seek setting.
    registers_write_byte(REG_REVERSE_SEEK, 0x00);
}
```

7.21 Dokumentacja pliku power.c

```
#include <inttypes.h>
#include "openservo.h"
#include "config.h"
#include "power.h"
#include "registers.h"
```

Funkcje

- void power_init (void)
- void power_update (uint16_t power)

7.21.1 Dokumentacja funkcji

7.21.1.1 void power_init (void)

Definicja w linii 42 pliku power.c.

Odwołuje się do REG_POWER_HI, REG_POWER_LO i registers_write_word().

Odwołania w main().

```
{
    uint8_t i;

    // Initialize the power index.
    power_index = 0;

    // Initialize the power array.
    for (i = 0; i < 8; ++i) power_array[i] = 0;

    // Initialize the power values within the system registers.
    registers_write_word(REG_POWER_HI,
      REG_POWER_LO, 0);
}
```

7.21.1.2 void power_update (uint16_t *power*)

Definicja w linii 58 pliku power.c.

Odwołuje się do REG_POWER_HI, REG_POWER_LO i registers_write_word().

Odwołania w main().

```
{
    uint8_t i;

    // Insert the power value into the power array.
    power_array[power_index] = power;

    // Keep the index within the array bounds.
    power_index = (power_index + 1) & 7;

    // Reset the power value.
    power = 0;

    // Determine the power values across the power array.
    for (i = 0; i < 8; ++i) power += power_array[i];

    // Shift the sum of power values to find the average.
    power >>= 3;

    // Update the power values within the system registers.
    registers_write_word(REG_POWER_HI,
      REG_POWER_LO, power);
}
```

7.22 Dokumentacja pliku power.h

Funkcje

- void power_init (void)
- void power_update (uint16_t power)

7.22.1 Dokumentacja funkcji

7.22.1.1 void power_init (void)

Definicja w linii 42 pliku power.c.

Odwołuje się do REG_POWER_HI, REG_POWER_LO i registers_write_word().

Odwołania w main().

```
{
    uint8_t i;

    // Initialize the power index.
    power_index = 0;

    // Initialize the power array.
    for (i = 0; i < 8; ++i) power_array[i] = 0;

    // Initialize the power values within the system registers.
    registers_write_word(REG_POWER_HI,
      REG_POWER_LO, 0);
}
```

7.22.1.2 void power_update (uint16_t *power*)

Definicja w linii 58 pliku power.c.

Odwołuje się do REG_POWER_HI, REG_POWER_LO i registers_write_word().

Odwołania w main().

```
{
    uint8_t i;

    // Insert the power value into the power array.
    power_array[power_index] = power;

    // Keep the index within the array bounds.
    power_index = (power_index + 1) & 7;

    // Reset the power value.
    power = 0;

    // Determine the power values across the power array.
    for (i = 0; i < 8; ++i) power += power_array[i];

    // Shift the sum of power values to find the average.
    power >>= 3;

    // Update the power values within the system registers.
    registers_write_word(REG_POWER_HI,
      REG_POWER_LO, power);
}
```

7.23 Dokumentacja pliku pulsectl.c

```
#include <inttypes.h>
#include <avr/io.h>
#include <avr/interrupt.h>
#include "openservo.h"
#include "config.h"
#include "registers.h"
#include "pulsectl.h"
#include "pwm.h"
#include "timer.h"
```

7.24 Dokumentacja pliku pulsectl.h

Funkcje

- void pulse_control_init (void)
- void pulse_control_update (void)

7.24.1 Dokumentacja funkcji

7.24.1.1 void pulse_control_init (void)

Odwołania w main().

7.24.1.2 void pulse_control_update (void)

Odwołania w main().

7.25 Dokumentacja pliku pwm.c

```
#include <inttypes.h>
#include <avr/interrupt.h>
#include <avr/io.h>
#include "openservo.h"
#include "config.h"
#include "pwm.h"
#include "registers.h"
```

Definicje

- #define PWM_TOP_VALUE(div) ((uint16_t) div << 4) - 1;
- #define PWM_OCRN_VALUE(div, pwm) (uint16_t) (((uint32_t) pwm * (((uint32_t) div << 4) - 1)) / 255)
- #define DELAYLOOP 8

Funkcje

- void pwm_registers_defaults (void)
- void pwm_init (void)
- void pwm_update (uint16_t position, int16_t pwm)
- void pwm_stop (void)

7.25.1 Dokumentacja definicji

7.25.1.1 #define DELAYLOOP 8

Definicja w linii 78 pliku pwm.c.

Odwołania w pwm_stop() i pwm_update().

7.25.1.2 #define PWM_OCRN_VALUE(*div,* *pwm*) (uint16_t) (((uint32_t) pwm * (((uint32_t) div << 4) - 1)) / 255)

Definicja w linii 62 pliku pwm.c.

7.25.1.3 #define PWM_TOP_VALUE(*div*) ((uint16_t) div << 4) - 1;

Definicja w linii 59 pliku pwm.c.

Odwołania w pwm_init() i pwm_update().

7.25.2 Dokumentacja funkcji

7.25.2.1 void pwm_init (void)

Definicja w linii 207 pliku pwm.c.

Odwołuje się do PWM_TOP_VALUE, REG_PWM_DIRA, REG_PWM_DIRB, REG_PWM_FREQ_DIVIDER_HI, REG_PWM_FREQ_DIVIDER_LO i registers_read_word().

Odwołania w main().

```
{
    // Initialize the pwm frequency divider value.
    pwm_div = registers_read_word(REG_PWM_FREQ_DIVIDER_HI
      , REG_PWM_FREQ_DIVIDER_LO);

    TCCR1A = 0;
        __asm__("nop");
        __asm__("nop");
        __asm__("nop");

    // Set PB1/OC1A and PB2/OC1B to low.
    PORTB &= ~((1<<PB1) | (1<<PB2));

    // Enable PB1/OC1A and PB2/OC1B as outputs.
    DDRB |= ((1<<DDB1) | (1<<DDB2));

    // Reset the timer1 configuration.
    TCNT1 = 0;
    TCCR1A = 0;
    TCCR1B = 0;
    TCCR1C = 0;
    TIMSK1 = 0;

    // Set timer top value.
    ICR1 = PWM_TOP_VALUE(pwm_div);

    // Set the PWM duty cycle to zero.
    OCR1A = 0;
    OCR1B = 0;

    // Configure timer 1 for PWM, Phase and Frequency Correct operation, but
       leave outputs disabled.
    TCCR1A = (0<<COM1A1) | (0<<COM1A0) |                    // Disable OC1A
       output.
             (0<<COM1B1) | (0<<COM1B0) |                    // Disable OC1B
       output.
             (0<<WGM11) | (0<<WGM10);                       // PWM, Phase and
       Frequency Correct, TOP = ICR1
    TCCR1B = (0<<ICNC1) | (0<<ICES1) |                      // Input on ICP1
       disabled.
             (1<<WGM13) | (0<<WGM12) |                      // PWM, Phase and
       Frequency Correct, TOP = ICR1
             (0<<CS12) | (0<<CS11) | (1<<CS10);             // No prescaling.

    // Update the pwm values.
    registers_write_byte(REG_PWM_DIRA, 0);
    registers_write_byte(REG_PWM_DIRB, 0);
}
```

7.25.2.2 void pwm_registers_defaults (void)

Definicja w linii 193 pliku pwm.c.

Odwołuje się do DEFAULT_PWM_FREQ_DIVIDER, REG_PWM_FREQ_DIVIDER_HI, REG_PWM_FREQ_DIVIDER_LO i registers_write_word().

Odwołania w registers_defaults().

```
{
    // PWM divider is a value between 1 and 1024.  This divides the fundamental
    // PWM frequency (500 kHz for 8MHz clock, 1250 kHz for 20MHz clock) by a
    // constant value to produce a PWM frequency suitable to drive a motor.  A
    // small motor with low inductance and impedance such as those found in an
    // RC servo will my typically use a divider value between 16 and 64.  A
       larger
    // motor with higher inductance and impedance may require a greater
       divider.
    registers_write_word(REG_PWM_FREQ_DIVIDER_HI
      , REG_PWM_FREQ_DIVIDER_LO, DEFAULT_PWM_FREQ_DIVIDER
      );
}
```

7.25.2.3 void pwm_stop (void)

Definicja w linii 373 pliku pwm.c.

Odwołuje się do DELAYLOOP, REG_PWM_DIRA i REG_PWM_DIRB.

Odwołania w pwm_update().

```
{
    // Disable interrupts.
    cli();

    // Are we moving in the A or B direction?
    if (pwm_a || pwm_b)
    {
        // Disable OC1A and OC1B outputs.
        TCCR1A &= ~((1<<COM1A1) | (1<<COM1A0));
        TCCR1A &= ~((1<<COM1B1) | (1<<COM1B0));

        // Clear PB1 and PB2.
        PORTB &= ~((1<<PB1) | (1<<PB2));

        delay_loop(DELAYLOOP);

        // Reset the A and B direction flags.
        pwm_a = 0;
        pwm_b = 0;
    }

    // Set the PWM duty cycle to zero.
    OCR1A = 0;
    OCR1B = 0;

    // Restore interrupts.
    sei();

    // Save the pwm A and B duty values.
    registers_write_byte(REG_PWM_DIRA, pwm_a);
    registers_write_byte(REG_PWM_DIRB, pwm_b);
}
```

7.25.2.4 void pwm_update (uint16_t *position,* int16_t *pwm*)

Definicja w linii 252 pliku pwm.c.

Odwołuje się do DELAYLOOP, FLAGS_LO_PWM_ENABLED, pwm_stop(), PWM_TOP_VALUE, REG_FLAGS_LO, REG_MAX_SEEK_HI, REG_MAX_SEEK_LO, REG_MIN_SEEK_HI, REG_MIN_SEEK_LO, REG_PWM_FREQ_DIVIDER_HI, REG_PWM_FREQ_DIVIDER_LO, REG_REVERSE_SEEK i registers_read_word().

Odwołania w main().

```
{
    uint8_t pwm_width;
    uint16_t min_position;
    uint16_t max_position;

    // Quick check to see if the frequency divider changed.  If so we need to
    // configure a new top value for timer/counter1.  This value should only
    // change infrequently so we aren't too elegant in how we handle updating
    // the value.  However, we need to be careful that we don't configure the
    // top to a value lower than the counter and compare values.
    if (registers_read_word(REG_PWM_FREQ_DIVIDER_HI
      , REG_PWM_FREQ_DIVIDER_LO) != pwm_div)
    {
        // Disable OC1A and OC1B outputs.
        TCCR1A &= ~((1<<COM1A1) | (1<<COM1A0));
        TCCR1A &= ~((1<<COM1B1) | (1<<COM1B0));

        // Clear PB1 and PB2.
        PORTB &= ~((1<<PB1) | (1<<PB2));

        delay_loop(DELAYLOOP);

        // Reset the A and B direction flags.
        pwm_a = 0;
        pwm_b = 0;

        // Update the pwm frequency divider value.
        pwm_div = registers_read_word(
      REG_PWM_FREQ_DIVIDER_HI, REG_PWM_FREQ_DIVIDER_LO);

        // Update the timer top value.
        ICR1 = PWM_TOP_VALUE(pwm_div);

        // Reset the counter and compare values to prevent problems with the
       new top value.
        TCNT1 = 0;
        OCR1A = 0;
        OCR1B = 0;
    }

    // Are we reversing the seek sense?
    if (registers_read_byte(REG_REVERSE_SEEK) != 0)
    {
        // Yes. Swap the minimum and maximum position.
        // Get the minimum and maximum seek position.
        min_position = registers_read_word(REG_MAX_SEEK_HI
      , REG_MAX_SEEK_LO);
        max_position = registers_read_word(REG_MIN_SEEK_HI
      , REG_MIN_SEEK_LO);

        // Make sure these values are sane 10-bit values.
        if (min_position > 0x3ff) min_position = 0x3ff;
        if (max_position > 0x3ff) max_position = 0x3ff;

        // Adjust the values because of the reverse sense.
        min_position = 0x3ff - min_position;
        max_position = 0x3ff - max_position;
    }
    else
    {
        // No. Use the minimum and maximum position as is.

        // Get the minimum and maximum seek position.
        min_position = registers_read_word(REG_MIN_SEEK_HI
      , REG_MIN_SEEK_LO);
        max_position = registers_read_word(REG_MAX_SEEK_HI
      , REG_MAX_SEEK_LO);

        // Make sure these values are sane 10-bit values.
        if (min_position > 0x3ff) min_position = 0x3ff;
        if (max_position > 0x3ff) max_position = 0x3ff;
    }

    // Disable clockwise movements when position is below the minimum position.
    if ((position < min_position) && (pwm < 0)) pwm = 0;

    // Disable counter-clockwise movements when position is above the maximum
       position.
    if ((position > max_position) && (pwm > 0)) pwm = 0;

    // Determine if PWM is disabled in the registers.
    if (!(registers_read_byte(REG_FLAGS_LO) & (1<<
      FLAGS_LO_PWM_ENABLED))) pwm = 0;

    // Determine direction of servo movement or stop.
    if (pwm < 0)
    {
        // Less than zero. Turn clockwise.

        // Get the PWM width from the PWM value.
        pwm_width = (uint8_t) -pwm;

        // Turn clockwise.
#if SWAP_PWM_DIRECTION_ENABLED
        pwm_dir_a(pwm_width);
#else
        pwm_dir_b(pwm_width);
#endif
    }
    else if (pwm > 0)
    {
        // More than zero. Turn counter-clockwise.

        // Get the PWM width from the PWM value.
        pwm_width = (uint8_t) pwm;

        // Turn counter-clockwise.
#if SWAP_PWM_DIRECTION_ENABLED
        pwm_dir_b(pwm_width);
#else
        pwm_dir_a(pwm_width);
#endif

    }
    else
    {
        // Stop all PWM activity to the motor.
        pwm_stop();
    }
}
```

7.26 Dokumentacja pliku pwm.h

```
#include "registers.h"
```

Funkcje

- void pwm_registers_defaults (void)
- void pwm_init (void)
- void pwm_update (uint16_t position, int16_t pwm)
- void pwm_stop (void)

7.26.1 Dokumentacja funkcji

7.26.1.1 void pwm_init (void)

Definicja w linii 207 pliku pwm.c.

Odwołuje się do PWM_TOP_VALUE, REG_PWM_DIRA, REG_PWM_DIRB, REG_PWM_FREQ_DIVIDER_HI, REG_PWM_FREQ_DIVIDER_LO i registers_read_word().

Odwołania w main().

```
{
    // Initialize the pwm frequency divider value.
    pwm_div = registers_read_word(REG_PWM_FREQ_DIVIDER_HI
      , REG_PWM_FREQ_DIVIDER_LO);

    TCCR1A = 0;
        __asm__("nop");
        __asm__("nop");
        __asm__("nop");

    // Set PB1/OC1A and PB2/OC1B to low.
    PORTB &= ~((1<<PB1) | (1<<PB2));

    // Enable PB1/OC1A and PB2/OC1B as outputs.
    DDRB |= ((1<<DDB1) | (1<<DDB2));

    // Reset the timer1 configuration.
    TCNT1 = 0;
    TCCR1A = 0;
    TCCR1B = 0;
    TCCR1C = 0;
    TIMSK1 = 0;

    // Set timer top value.
    ICR1 = PWM_TOP_VALUE(pwm_div);

    // Set the PWM duty cycle to zero.
    OCR1A = 0;
    OCR1B = 0;

    // Configure timer 1 for PWM, Phase and Frequency Correct operation, but
       leave outputs disabled.
    TCCR1A = (0<<COM1A1) | (0<<COM1A0) |                    // Disable OC1A
       output.
             (0<<COM1B1) | (0<<COM1B0) |                    // Disable OC1B
       output.
             (0<<WGM11) | (0<<WGM10);                       // PWM, Phase and
       Frequency Correct, TOP = ICR1
    TCCR1B = (0<<ICNC1) | (0<<ICES1) |                      // Input on ICP1
       disabled.
             (1<<WGM13) | (0<<WGM12) |                      // PWM, Phase and
       Frequency Correct, TOP = ICR1
             (0<<CS12) | (0<<CS11) | (1<<CS10);             // No prescaling.

    // Update the pwm values.
    registers_write_byte(REG_PWM_DIRA, 0);
    registers_write_byte(REG_PWM_DIRB, 0);
}
```

7.26.1.2 void pwm_registers_defaults (void)

Definicja w linii 193 pliku pwm.c.

Odwołuje się do DEFAULT_PWM_FREQ_DIVIDER, REG_PWM_FREQ_DIVIDER_HI, REG_PWM_FREQ_DIVIDER_LO i registers_write_word().

Odwołania w registers_defaults().

```
{
    // PWM divider is a value between 1 and 1024.  This divides the fundamental
    // PWM frequency (500 kHz for 8MHz clock, 1250 kHz for 20MHz clock) by a
    // constant value to produce a PWM frequency suitable to drive a motor.  A
    // small motor with low inductance and impedance such as those found in an
    // RC servo will my typically use a divider value between 16 and 64.  A
       larger
    // motor with higher inductance and impedance may require a greater
       divider.
    registers_write_word(REG_PWM_FREQ_DIVIDER_HI
      , REG_PWM_FREQ_DIVIDER_LO, DEFAULT_PWM_FREQ_DIVIDER
      );
}
```

7.26.1.3 void pwm_stop (void)

Definicja w linii 373 pliku pwm.c.

Odwołuje się do DELAYLOOP, REG_PWM_DIRA i REG_PWM_DIRB.

Odwołania w pwm_update().

```
{
    // Disable interrupts.
    cli();

    // Are we moving in the A or B direction?
    if (pwm_a || pwm_b)
    {
        // Disable OC1A and OC1B outputs.
        TCCR1A &= ~((1<<COM1A1) | (1<<COM1A0));
        TCCR1A &= ~((1<<COM1B1) | (1<<COM1B0));

        // Clear PB1 and PB2.
        PORTB &= ~((1<<PB1) | (1<<PB2));

        delay_loop(DELAYLOOP);

        // Reset the A and B direction flags.
        pwm_a = 0;
        pwm_b = 0;
    }

    // Set the PWM duty cycle to zero.
    OCR1A = 0;
    OCR1B = 0;

    // Restore interrupts.
    sei();

    // Save the pwm A and B duty values.
    registers_write_byte(REG_PWM_DIRA, pwm_a);
    registers_write_byte(REG_PWM_DIRB, pwm_b);
}
```

7.26.1.4 void pwm_update (uint16_t *position,* int16_t *pwm*)

Definicja w linii 252 pliku pwm.c.

Odwołuje się do DELAYLOOP, FLAGS_LO_PWM_ENABLED, pwm_stop(), PWM_TOP_VALUE, REG_FLAGS_LO, REG_MAX_SEEK_HI, REG_MAX_SEEK_LO, REG_MIN_SEEK_HI, REG_MIN_SEEK_LO, REG_PWM_FREQ_DIVIDER_HI, REG_PWM_FREQ_DIVIDER_LO, REG_REVERSE_SEEK i registers_read_word().

Odwołania w main().

```
{
    uint8_t pwm_width;
    uint16_t min_position;
    uint16_t max_position;

    // Quick check to see if the frequency divider changed.  If so we need to
    // configure a new top value for timer/counter1.  This value should only
    // change infrequently so we aren't too elegant in how we handle updating
    // the value.  However, we need to be careful that we don't configure the
    // top to a value lower than the counter and compare values.
    if (registers_read_word(REG_PWM_FREQ_DIVIDER_HI
      , REG_PWM_FREQ_DIVIDER_LO) != pwm_div)
    {
        // Disable OC1A and OC1B outputs.
        TCCR1A &= ~((1<<COM1A1) | (1<<COM1A0));
        TCCR1A &= ~((1<<COM1B1) | (1<<COM1B0));

        // Clear PB1 and PB2.
        PORTB &= ~((1<<PB1) | (1<<PB2));

        delay_loop(DELAYLOOP);

        // Reset the A and B direction flags.
        pwm_a = 0;
        pwm_b = 0;

        // Update the pwm frequency divider value.
        pwm_div = registers_read_word(
      REG_PWM_FREQ_DIVIDER_HI, REG_PWM_FREQ_DIVIDER_LO);

        // Update the timer top value.
        ICR1 = PWM_TOP_VALUE(pwm_div);

        // Reset the counter and compare values to prevent problems with the
       new top value.
        TCNT1 = 0;
        OCR1A = 0;
        OCR1B = 0;
    }

    // Are we reversing the seek sense?
    if (registers_read_byte(REG_REVERSE_SEEK) != 0)
    {
        // Yes. Swap the minimum and maximum position.

        // Get the minimum and maximum seek position.
        min_position = registers_read_word(REG_MAX_SEEK_HI
      , REG_MAX_SEEK_LO);
        max_position = registers_read_word(REG_MIN_SEEK_HI
      , REG_MIN_SEEK_LO);

        // Make sure these values are sane 10-bit values.
        if (min_position > 0x3ff) min_position = 0x3ff;
        if (max_position > 0x3ff) max_position = 0x3ff;

        // Adjust the values because of the reverse sense.
        min_position = 0x3ff - min_position;
        max_position = 0x3ff - max_position;
    }
    else
    {
        // No. Use the minimum and maximum position as is.

        // Get the minimum and maximum seek position.
        min_position = registers_read_word(REG_MIN_SEEK_HI
      , REG_MIN_SEEK_LO);
        max_position = registers_read_word(REG_MAX_SEEK_HI
      , REG_MAX_SEEK_LO);

        // Make sure these values are sane 10-bit values.
        if (min_position > 0x3ff) min_position = 0x3ff;
        if (max_position > 0x3ff) max_position = 0x3ff;
    }

    // Disable clockwise movements when position is below the minimum position.
    if ((position < min_position) && (pwm < 0)) pwm = 0;

    // Disable counter-clockwise movements when position is above the maximum
       position.
    if ((position > max_position) && (pwm > 0)) pwm = 0;

    // Determine if PWM is disabled in the registers.
    if (!(registers_read_byte(REG_FLAGS_LO) & (1<<
      FLAGS_LO_PWM_ENABLED))) pwm = 0;

    // Determine direction of servo movement or stop.
    if (pwm < 0)
    {
        // Less than zero. Turn clockwise.

        // Get the PWM width from the PWM value.
        pwm_width = (uint8_t) -pwm;

        // Turn clockwise.
#if SWAP_PWM_DIRECTION_ENABLED
        pwm_dir_a(pwm_width);
#else
        pwm_dir_b(pwm_width);
#endif
    }
    else if (pwm > 0)
    {
        // More than zero. Turn counter-clockwise.

        // Get the PWM width from the PWM value.
        pwm_width = (uint8_t) pwm;

        // Turn counter-clockwise.
#if SWAP_PWM_DIRECTION_ENABLED
        pwm_dir_b(pwm_width);
#else
        pwm_dir_a(pwm_width);
#endif

    }
    else
    {
        // Stop all PWM activity to the motor.
        pwm_stop();
    }
}
```

7.27 Dokumentacja pliku registers.c

```
#include <inttypes.h>
#include <string.h>
#include "openservo.h"
#include "config.h"
#include "eeprom.h"
#include "estimator.h"
#include "ipd.h"
#include "pid.h"
#include "pwm.h"
#include "regulator.h"
#include "registers.h"
```

Funkcje

- void registers_init (void)
- void registers_defaults (void)
- uint16_t registers_read_word (uint8_t address_hi, uint8_t address_lo)
- void registers_write_word (uint8_t address_hi, uint8_t address_lo, uint16_t value)

Zmienne

- uint8_t registers [REGISTER_COUNT]

7.27.1 Dokumentacja funkcji

7.27.1.1 void registers_defaults (void)

Definicja w linii 69 pliku registers.c.

Odwołuje się do estimator_registers_defaults(), ipd_registers_defaults(), pid_registers_defaults(), pwm_registers_defaults(), REG_DEFAULT_TWI_ADDR, REG_TWI_ADDRESS i regulator_registers_defaults().

Odwołania w registers_init() i RS485CMD().

```
{
    // Initialize read/write protected registers to defaults.

    // Default TWI address.
    registers_write_byte(REG_TWI_ADDRESS, REG_DEFAULT_TWI_ADDR
      );
    // Call the PWM module to initialize the PWM related default values.
    pwm_registers_defaults();

#if ESTIMATOR_ENABLED
    // Call the motion module to initialize the velocity estimator related
    // default values. This is done so the estimator related parameters can
    // be kept in a single file.
    estimator_registers_defaults();
#endif

#if REGULATOR_MOTION_ENABLED
    // Call the regulator module to initialize the regulator related default
       values.
    regulator_registers_defaults();
#endif

#if PID_MOTION_ENABLED
    // Call the PID module to initialize the PID related default values.
    pid_registers_defaults();
#endif

#if IPD_MOTION_ENABLED
    // Call the IPD module to initialize the IPD related default values.
    ipd_registers_defaults();
#endif
}
```

7.27.1.2 void registers_init (void)

Definicja w linii 43 pliku registers.c.

Odwołuje się do eeprom_restore_registers(), MIN_WRITE_PROTECT_REGISTER, OPENSERVO_DEVICE_SUBTYPE, OPENSERVO_DEVICE_TYPE, REDIRECT_REGISTER_COUNT, REG_DEVICE_SUBTYPE, REG_DEVICE_TYPE, REG_VERSION_MAJOR, REG_VERSION_MINOR, REGISTER_COUNT, registers, registers_defaults(), SOFTWARE_VERSION_MAJOR, SOFTWARE_VERSION_MINOR i WRITE_PROTECT_REGISTER_COUNT.

Odwołania w main().

```
{
    // Initialize all registers to zero.
    memset(&registers[0], 0, REGISTER_COUNT);

    // Set device and software identification information.
    registers_write_byte(REG_DEVICE_TYPE, OPENSERVO_DEVICE_TYPE
      );
    registers_write_byte(REG_DEVICE_SUBTYPE,
      OPENSERVO_DEVICE_SUBTYPE);
    registers_write_byte(REG_VERSION_MAJOR,
      SOFTWARE_VERSION_MAJOR);
    registers_write_byte(REG_VERSION_MINOR,
      SOFTWARE_VERSION_MINOR);

    // Restore the read/write protected registers from EEPROM.  If the
    // EEPROM fails checksum this function will return zero and the
    // read/write protected registers should be initialized to defaults.
    if (!eeprom_restore_registers())
    {
        // Reset read/write protected registers to zero.
        memset(&registers[MIN_WRITE_PROTECT_REGISTER
      ], WRITE_PROTECT_REGISTER_COUNT +
      REDIRECT_REGISTER_COUNT, REGISTER_COUNT);

        // Initialize read/write protected registers to defaults.
        registers_defaults();
    }
}
```

7.27.1.3 uint16_t registers_read_word (uint8_t *address_hi*, uint8_t *address_lo*)

Definicja w linii 104 pliku registers.c.

Odwołuje się do registers.

Odwołania w motion_append(), pid_position_to_pwm(), pwm_init(), pwm_update() i RS485CMD().

```
{
    uint8_t sreg;
    uint16_t value;

    // Clear interrupts.
    __asm__ volatile ("in %0,__SREG__\n\tcli\n\t" : "=&r" (sreg));

    // Read the registers.
    value = (registers[address_hi] << 8) | registers[
      address_lo];

    // Restore status.
    __asm__ volatile ("out __SREG__,%0\n\t" : : "r" (sreg));

    return value;
}
```

7.27.1.4 void registers_write_word (uint8_t *address_hi,* uint8_t *address_lo,* uint16_t *value*)

Definicja w linii 125 pliku registers.c.

Odwołuje się do registers.

Odwołania w main(), motion_next(), motion_registers_reset(), pid_position_to_pwm(), pid_registers_defaults(), power_init(), power_update(), pwm_registers_defaults() i RS485CMD().

```
{
    uint8_t sreg;

    // Clear interrupts.
    __asm__ volatile ("in %0,__SREG__\n\tcli\n\t" : "=&r" (sreg));

    // Write the registers.
    registers[address_hi] = value >> 8;
    registers[address_lo] = value;

    // Restore status.
    __asm__ volatile ("out __SREG__,%0\n\t" : : "r" (sreg));
}
```

7.27.2 Dokumentacja zmiennych

7.27.2.1 uint8_t registers[REGISTER_COUNT]

Definicja w linii 41 pliku registers.c.

Odwołania w eeprom_restore_registers(), eeprom_save_registers(), registers_init(), registers_read_word() i registers_write_word().

7.28 Dokumentacja pliku registers.h

Definicje

- #define REG_DEVICE_TYPE 0x00
- #define REG_DEVICE_SUBTYPE 0x01
- #define REG_VERSION_MAJOR 0x02
- #define REG_VERSION_MINOR 0x03
- #define REG_FLAGS_HI 0x04
- #define REG_FLAGS_LO 0x05
- #define REG_TIMER_HI 0x06
- #define REG_TIMER_LO 0x07
- #define REG_POSITION_HI 0x08
- #define REG_POSITION_LO 0x09
- #define REG_VELOCITY_HI 0x0A
- #define REG_VELOCITY_LO 0x0B
- #define REG_POWER_HI 0x0C

- #define REG_POWER_LO 0x0D
- #define REG_PWM_DIRA 0x0E
- #define REG_PWM_DIRB 0x0F
- #define REG_SEEK_POSITION_HI 0x10
- #define REG_SEEK_POSITION_LO 0x11
- #define REG_SEEK_VELOCITY_HI 0x12
- #define REG_SEEK_VELOCITY_LO 0x13
- #define REG_VOLTAGE_HI 0x14
- #define REG_VOLTAGE_LO 0x15
- #define REG_CURVE_RESERVED 0x16
- #define REG_CURVE_BUFFER 0x17
- #define REG_CURVE_DELTA_HI 0x18
- #define REG_CURVE_DELTA_LO 0x19
- #define REG_CURVE_POSITION_HI 0x1A
- #define REG_CURVE_POSITION_LO 0x1B
- #define REG_CURVE_IN_VELOCITY_HI 0x1C
- #define REG_CURVE_IN_VELOCITY_LO 0x1D
- #define REG_CURVE_OUT_VELOCITY_HI 0x1E
- #define REG_CURVE_OUT_VELOCITY_LO 0x1F
- #define REG_TWI_ADDRESS 0x20
- #define REG_PID_DEADBAND 0x21
- #define REG_PID_PGAIN_HI 0x22
- #define REG_PID_PGAIN_LO 0x23
- #define REG_PID_DGAIN_HI 0x24
- #define REG_PID_DGAIN_LO 0x25
- #define REG_PID_IGAIN_HI 0x26
- #define REG_PID_IGAIN_LO 0x27
- #define REG_PWM_FREQ_DIVIDER_HI 0x28
- #define REG_PWM_FREQ_DIVIDER_LO 0x29
- #define REG_MIN_SEEK_HI 0x2A
- #define REG_MIN_SEEK_LO 0x2B
- #define REG_MAX_SEEK_HI 0x2C
- #define REG_MAX_SEEK_LO 0x2D
- #define REG_REVERSE_SEEK 0x2E
- #define REG_RESERVED_2F 0x2F
- #define MIN_READ_ONLY_REGISTER 0x00
- #define MAX_READ_ONLY_REGISTER 0x0F
- #define MIN_READ_WRITE_REGISTER 0x10
- #define MAX_READ_WRITE_REGISTER 0x1F
- #define MIN_WRITE_PROTECT_REGISTER 0x20
- #define MAX_WRITE_PROTECT_REGISTER 0x2F
- #define MIN_UNUSED_REGISTER 0x30
- #define MAX_UNUSED_REGISTER 0x5F
- #define MIN_REDIRECT_REGISTER 0x60
- #define MAX_REDIRECT_REGISTER 0x6F
- #define MIN_REDIRECTED_REGISTER 0x70
- #define MAX_REDIRECTED_REGISTER 0x7F
- #define REGISTER_COUNT (MIN_UNUSED_REGISTER + 16)
- #define WRITE_PROTECT_REGISTER_COUNT (MAX_WRITE_PROTECT_REGISTER - MIN_WRITE_PROTECT_REGISTER + 1)
- #define REDIRECT_REGISTER_COUNT (MAX_REDIRECT_REGISTER - MIN_REDIRECT_REGISTER + 1)
- #define FLAGS_HI_RESERVED_07 0x07
- #define FLAGS_HI_RESERVED_06 0x06
- #define FLAGS_HI_RESERVED_05 0x05

- #define FLAGS_HI_RESERVED_04 0x04
- #define FLAGS_HI_RESERVED_03 0x03
- #define FLAGS_HI_RESERVED_02 0x02
- #define FLAGS_HI_RESERVED_01 0x01
- #define FLAGS_HI_RESERVED_00 0x00
- #define FLAGS_LO_RESERVED_07 0x07
- #define FLAGS_LO_RESERVED_06 0x06
- #define FLAGS_LO_RESERVED_05 0x05
- #define FLAGS_LO_RESERVED_04 0x04
- #define FLAGS_LO_RESERVED_03 0x03
- #define FLAGS_LO_MOTION_ENABLED 0x02
- #define FLAGS_LO_WRITE_ENABLED 0x01
- #define FLAGS_LO_PWM_ENABLED 0x00

Funkcje

- void registers_init (void)
- void registers_defaults (void)
- uint16_t registers_read_word (uint8_t address_hi, uint8_t address_lo)
- void registers_write_word (uint8_t address_hi, uint8_t address_lo, uint16_t value)

Zmienne

- uint8_t registers [REGISTER_COUNT]

7.28.1 Dokumentacja definicji

7.28.1.1 #define FLAGS_HI_RESERVED_00 0x00

Definicja w linii 172 pliku registers.h.

7.28.1.2 #define FLAGS_HI_RESERVED_01 0x01

Definicja w linii 171 pliku registers.h.

7.28.1.3 #define FLAGS_HI_RESERVED_02 0x02

Definicja w linii 170 pliku registers.h.

7.28.1.4 #define FLAGS_HI_RESERVED_03 0x03

Definicja w linii 169 pliku registers.h.

7.28.1.5 #define FLAGS_HI_RESERVED_04 0x04

Definicja w linii 168 pliku registers.h.

7.28.1.6 #define FLAGS_HI_RESERVED_05 0x05

Definicja w linii 167 pliku registers.h.

7.28.1.7 #define FLAGS_HI_RESERVED_06 0x06

Definicja w linii 166 pliku registers.h.

7.28.1.8 #define FLAGS_HI_RESERVED_07 0x07

Definicja w linii 165 pliku registers.h.

7.28.1.9 #define FLAGS_LO_MOTION_ENABLED 0x02

Definicja w linii 179 pliku registers.h.

Odwołania w motion_next().

7.28.1.10 #define FLAGS_LO_PWM_ENABLED 0x00

Definicja w linii 181 pliku registers.h.

Odwołania w pwm_update().

7.28.1.11 #define FLAGS_LO_RESERVED_03 0x03

Definicja w linii 178 pliku registers.h.

7.28.1.12 #define FLAGS_LO_RESERVED_04 0x04

Definicja w linii 177 pliku registers.h.

7.28.1.13 #define FLAGS_LO_RESERVED_05 0x05

Definicja w linii 176 pliku registers.h.

7.28.1.14 #define FLAGS_LO_RESERVED_06 0x06

Definicja w linii 175 pliku registers.h.

7.28.1.15 #define FLAGS_LO_RESERVED_07 0x07

Definicja w linii 174 pliku registers.h.

7.28.1.16 #define FLAGS_LO_WRITE_ENABLED 0x01

Definicja w linii 180 pliku registers.h.

7.28.1.17 #define MAX_READ_ONLY_REGISTER 0x0F

Definicja w linii 137 pliku registers.h.

7.28.1.18 #define MAX_READ_WRITE_REGISTER 0x1F

Definicja w linii 139 pliku registers.h.

7.28.1.19 #define MAX_REDIRECT_REGISTER 0x6F

Definicja w linii 145 pliku registers.h.

7.28.1.20 #define MAX_REDIRECTED_REGISTER 0x7F

Definicja w linii 147 pliku registers.h.

7.28.1.21 #define MAX_UNUSED_REGISTER 0x5F

Definicja w linii 143 pliku registers.h.

7.28.1.22 #define MAX_WRITE_PROTECT_REGISTER 0x2F

Definicja w linii 141 pliku registers.h.

Odwołania w RS485CMD().

7.28.1.23 #define MIN_READ_ONLY_REGISTER 0x00

Definicja w linii 136 pliku registers.h.

7.28.1.24 #define MIN_READ_WRITE_REGISTER 0x10

Definicja w linii 138 pliku registers.h.

7.28.1.25 #define MIN_REDIRECT_REGISTER 0x60

Definicja w linii 144 pliku registers.h.

7.28.1.26 #define MIN_REDIRECTED_REGISTER 0x70

Definicja w linii 146 pliku registers.h.

7.28.1.27 #define MIN_UNUSED_REGISTER 0x30

Definicja w linii 142 pliku registers.h.

7.28.1.28 #define MIN_WRITE_PROTECT_REGISTER 0x20

Definicja w linii 140 pliku registers.h.

Odwołania w eeprom_restore_registers(), eeprom_save_registers() i registers_init().

7.28.1.29 #define REDIRECT_REGISTER_COUNT (MAX_REDIRECT_REGISTER - MIN_REDIRECT_REGISTER + 1)

Definicja w linii 159 pliku registers.h.

Odwołania w eeprom_restore_registers(), eeprom_save_registers() i registers_init().

7.28.1.30 #define REG_CURVE_BUFFER 0x17

Definicja w linii 65 pliku registers.h.

Odwołania w motion_next() i motion_registers_reset().

7.28.1.31 #define REG_CURVE_DELTA_HI 0x18

Definicja w linii 67 pliku registers.h.

Odwołania w main(), motion_append() i motion_registers_reset().

7.28.1.32 #define REG_CURVE_DELTA_LO 0x19

Definicja w linii 68 pliku registers.h.

Odwołania w main(), motion_append() i motion_registers_reset().

7.28.1.33 #define REG_CURVE_IN_VELOCITY_HI 0x1C

Definicja w linii 71 pliku registers.h.

Odwołania w motion_append() i motion_registers_reset().

7.28.1.34 #define REG_CURVE_IN_VELOCITY_LO 0x1D

Definicja w linii 72 pliku registers.h.

Odwołania w motion_append() i motion_registers_reset().

7.28.1.35 #define REG_CURVE_OUT_VELOCITY_HI 0x1E

Definicja w linii 73 pliku registers.h.

Odwołania w motion_append() i motion_registers_reset().

7.28.1.36 #define REG_CURVE_OUT_VELOCITY_LO 0x1F

Definicja w linii 74 pliku registers.h.

Odwołania w motion_append() i motion_registers_reset().

7.28.1.37 #define REG_CURVE_POSITION_HI 0x1A

Definicja w linii 69 pliku registers.h.

Odwołania w main(), motion_append() i motion_registers_reset().

7.28.1.38 #define REG_CURVE_POSITION_LO 0x1B

Definicja w linii 70 pliku registers.h.

Odwołania w main(), motion_append() i motion_registers_reset().

7.28.1.39 #define REG_CURVE_RESERVED 0x16

Definicja w linii 64 pliku registers.h.

Odwołania w motion_registers_reset().

7.28.1.40 #define REG_DEVICE_SUBTYPE 0x01

Definicja w linii 38 pliku registers.h.

Odwołania w registers_init().

7.28.1.41 #define REG_DEVICE_TYPE 0x00

Definicja w linii 37 pliku registers.h.

Odwołania w registers_init().

7.28.1.42 #define REG_FLAGS_HI 0x04

Definicja w linii 41 pliku registers.h.

7.28.1.43 #define REG_FLAGS_LO 0x05

Definicja w linii 42 pliku registers.h.

Odwołania w motion_next() i pwm_update().

7.28.1.44 #define REG_MAX_SEEK_HI 0x2C

Definicja w linii 92 pliku registers.h.

Odwołania w pid_position_to_pwm(), pid_registers_defaults() i pwm_update().

7.28.1.45 #define REG_MAX_SEEK_LO 0x2D

Definicja w linii 93 pliku registers.h.

Odwołania w pid_position_to_pwm(), pid_registers_defaults() i pwm_update().

7.28.1.46 #define REG_MIN_SEEK_HI 0x2A

Definicja w linii 90 pliku registers.h.

Odwołania w pid_position_to_pwm(), pid_registers_defaults() i pwm_update().

7.28.1.47 #define REG_MIN_SEEK_LO 0x2B

Definicja w linii 91 pliku registers.h.

Odwołania w pid_position_to_pwm(), pid_registers_defaults() i pwm_update().

7.28.1.48 #define REG_PID_DEADBAND 0x21

Definicja w linii 80 pliku registers.h.

Odwołania w pid_position_to_pwm() i pid_registers_defaults().

7.28.1.49 #define REG_PID_DGAIN_HI 0x24

Definicja w linii 83 pliku registers.h.

Odwołania w pid_position_to_pwm() i pid_registers_defaults().

7.28.1.50 #define REG_PID_DGAIN_LO 0x25

Definicja w linii 84 pliku registers.h.

Odwołania w pid_position_to_pwm() i pid_registers_defaults().

7.28.1.51 #define REG_PID_IGAIN_HI 0x26

Definicja w linii 85 pliku registers.h.

Odwołania w pid_registers_defaults().

7.28.1.52 #define REG_PID_IGAIN_LO 0x27

Definicja w linii 86 pliku registers.h.

Odwołania w pid_registers_defaults().

7.28.1.53 #define REG_PID_PGAIN_HI 0x22

Definicja w linii 81 pliku registers.h.

Odwołania w pid_position_to_pwm() i pid_registers_defaults().

7.28.1.54 #define REG_PID_PGAIN_LO 0x23

Definicja w linii 82 pliku registers.h.

Odwołania w pid_position_to_pwm() i pid_registers_defaults().

7.28.1.55 #define REG_POSITION_HI 0x08

Definicja w linii 46 pliku registers.h.

Odwołania w pid_position_to_pwm().

7.28.1.56 #define REG_POSITION_LO 0x09

Definicja w linii 47 pliku registers.h.

Odwołania w pid_position_to_pwm().

7.28.1.57 #define REG_POWER_HI 0x0C

Definicja w linii 50 pliku registers.h.

Odwołania w power_init() i power_update().

7.28.1.58 #define REG_POWER_LO 0x0D

Definicja w linii 51 pliku registers.h.

Odwołania w power_init() i power_update().

7.28.1.59 #define REG_PWM_DIRA 0x0E

Definicja w linii 52 pliku registers.h.

Odwołania w pwm_init() i pwm_stop().

7.28.1.60 #define REG_PWM_DIRB 0x0F

Definicja w linii 53 pliku registers.h.

Odwołania w pwm_init() i pwm_stop().

7.28.1.61 #define REG_PWM_FREQ_DIVIDER_HI 0x28

Definicja w linii 88 pliku registers.h.

Odwołania w pwm_init(), pwm_registers_defaults() i pwm_update().

7.28.1.62 #define REG_PWM_FREQ_DIVIDER_LO 0x29

Definicja w linii 89 pliku registers.h.

Odwołania w pwm_init(), pwm_registers_defaults() i pwm_update().

7.28.1.63 #define REG_RESERVED_2F 0x2F

Definicja w linii 95 pliku registers.h.

7.28.1.64 #define REG_REVERSE_SEEK 0x2E

Definicja w linii 94 pliku registers.h.

Odwołania w pid_position_to_pwm(), pid_registers_defaults() i pwm_update().

7.28.1.65 #define REG_SEEK_POSITION_HI 0x10

Definicja w linii 58 pliku registers.h.

Odwołania w main(), motion_next() i pid_position_to_pwm().

7.28.1.66 #define REG_SEEK_POSITION_LO 0x11

Definicja w linii 59 pliku registers.h.

Odwołania w main(), motion_next() i pid_position_to_pwm().

7.28.1.67 #define REG_SEEK_VELOCITY_HI 0x12

Definicja w linii 60 pliku registers.h.

Odwołania w main(), motion_next() i pid_position_to_pwm().

7.28.1.68 #define REG_SEEK_VELOCITY_LO 0x13

Definicja w linii 61 pliku registers.h.

Odwołania w main(), motion_next() i pid_position_to_pwm().

7.28.1.69 #define REG_TIMER_HI 0x06

Definicja w linii 43 pliku registers.h.

7.28.1.70 #define REG_TIMER_LO 0x07

Definicja w linii 44 pliku registers.h.

7.28.1.71 #define REG_TWI_ADDRESS 0x20

Definicja w linii 79 pliku registers.h.

Odwołania w registers_defaults() i RS485CMD().

7.28.1.72 #define REG_VELOCITY_HI 0x0A

Definicja w linii 48 pliku registers.h.

Odwołania w pid_position_to_pwm().

7.28.1.73 #define REG_VELOCITY_LO 0x0B

Definicja w linii 49 pliku registers.h.

Odwołania w pid_position_to_pwm().

7.28.1.74 #define REG_VERSION_MAJOR 0x02

Definicja w linii 39 pliku registers.h.

Odwołania w registers_init().

7.28.1.75 #define REG_VERSION_MINOR 0x03

Definicja w linii 40 pliku registers.h.

Odwołania w registers_init().

7.28.1.76 #define REG_VOLTAGE_HI 0x14

Definicja w linii 62 pliku registers.h.

7.28.1.77 #define REG_VOLTAGE_LO 0x15

Definicja w linii 63 pliku registers.h.

7.28.1.78 #define REGISTER_COUNT (MIN_UNUSED_REGISTER + 16)

Definicja w linii 151 pliku registers.h.

Odwołania w registers_init().

7.28.1.79 #define WRITE_PROTECT_REGISTER_COUNT (MAX_WRITE_PROTECT_REGISTER - MIN_WRITE_PROTECT_REGISTER + 1)

Definicja w linii 156 pliku registers.h.

Odwołania w eeprom_restore_registers(), eeprom_save_registers() i registers_init().

7.28.2 Dokumentacja funkcji

7.28.2.1 void registers_defaults (void)

Definicja w linii 69 pliku registers.c.

Odwołuje się do estimator_registers_defaults(), ipd_registers_defaults(), pid_registers_defaults(), pwm_registers_-defaults(), REG_DEFAULT_TWI_ADDR, REG_TWI_ADDRESS i regulator_registers_defaults().

Odwołania w registers_init() i RS485CMD().

```
{
    // Initialize read/write protected registers to defaults.

    // Default TWI address.
    registers_write_byte(REG_TWI_ADDRESS, REG_DEFAULT_TWI_ADDR
      );

    // Call the PWM module to initialize the PWM related default values.
    pwm_registers_defaults();

#if ESTIMATOR_ENABLED
    // Call the motion module to initialize the velocity estimator related
    // default values. This is done so the estimator related parameters can
    // be kept in a single file.
    estimator_registers_defaults();
#endif

#if REGULATOR_MOTION_ENABLED
    // Call the regulator module to initialize the regulator related default
       values.
    regulator_registers_defaults();
#endif

#if PID_MOTION_ENABLED
    // Call the PID module to initialize the PID related default values.
    pid_registers_defaults();
#endif

#if IPD_MOTION_ENABLED
    // Call the IPD module to initialize the IPD related default values.
    ipd_registers_defaults();
#endif
}
```

7.28.2.2 void registers_init (void)

Definicja w linii 43 pliku registers.c.

Odwołuje się do eeprom_restore_registers(), MIN_WRITE_PROTECT_REGISTER, OPENSERVO_DEVICE_SUBTYPE, OPENSERVO_DEVICE_TYPE, REDIRECT_REGISTER_COUNT, REG_DEVICE_SUBTYPE, REG_DEVICE_TYPE, REG_VERSION_MAJOR, REG_VERSION_MINOR, REGISTER_COUNT, registers, registers_defaults(), SOFTWARE_VERSION_MAJOR, SOFTWARE_VERSION_MINOR i WRITE_PROTECT_REGISTER_COUNT.

Odwołania w main().

```
{
    // Initialize all registers to zero.
    memset(&registers[0], 0, REGISTER_COUNT);

    // Set device and software identification information.
    registers_write_byte(REG_DEVICE_TYPE, OPENSERVO_DEVICE_TYPE
      );
    registers_write_byte(REG_DEVICE_SUBTYPE,
      OPENSERVO_DEVICE_SUBTYPE);
    registers_write_byte(REG_VERSION_MAJOR,
      SOFTWARE_VERSION_MAJOR);
    registers_write_byte(REG_VERSION_MINOR,
      SOFTWARE_VERSION_MINOR);

    // Restore the read/write protected registers from EEPROM.  If the
    // EEPROM fails checksum this function will return zero and the
    // read/write protected registers should be initialized to defaults.
    if (!eeprom_restore_registers())
    {
        // Reset read/write protected registers to zero.
        memset(&registers[MIN_WRITE_PROTECT_REGISTER
      ], WRITE_PROTECT_REGISTER_COUNT +
      REDIRECT_REGISTER_COUNT, REGISTER_COUNT);

        // Initialize read/write protected registers to defaults.
        registers_defaults();
    }
}
```

7.28.2.3 uint16_t registers_read_word (uint8_t *address_hi,* uint8_t *address_lo*)

Definicja w linii 104 pliku registers.c.

Odwołuje się do registers.

Odwołania w motion_append(), pid_position_to_pwm(), pwm_init(), pwm_update() i RS485CMD().

```
{
    uint8_t sreg;
    uint16_t value;


    // Clear interrupts.
    __asm__ volatile ("in %0,__SREG__\n\tcli\n\t" : "=&r" (sreg));

    // Read the registers.
    value = (registers[address_hi] << 8) | registers[
      address_lo];

    // Restore status.
    __asm__ volatile ("out __SREG__,%0\n\t" : : "r" (sreg));

    return value;
}
```

7.28.2.4 void registers_write_word (uint8_t *address_hi,* uint8_t *address_lo,* uint16_t *value*)

Definicja w linii 125 pliku registers.c.

Odwołuje się do registers.

Odwołania w main(), motion_next(), motion_registers_reset(), pid_position_to_pwm(), pid_registers_defaults(), power_init(), power_update(), pwm_registers_defaults() i RS485CMD().

```
{
    uint8_t sreg;

    // Clear interrupts.
    __asm__ volatile ("in %0,__SREG__\n\tcli\n\t" : "=&r" (sreg));

    // Write the registers.
    registers[address_hi] = value >> 8;
    registers[address_lo] = value;

    // Restore status.
    __asm__ volatile ("out __SREG__,%0\n\t" : : "r" (sreg));
}
```

7.28.3 Dokumentacja zmiennych

7.28.3.1 uint8_t registers[REGISTER_COUNT]

Definicja w linii 41 pliku registers.c.

Odwołania w eeprom_restore_registers(), eeprom_save_registers(), registers_init(), registers_read_word() i registers_write_word().

7.29 Dokumentacja pliku regulator.c

```
#include <inttypes.h>
#include "openservo.h"
#include "config.h"
#include "math.h"
#include "regulator.h"
#include "registers.h"
```

7.30 Dokumentacja pliku regulator.h

Funkcje

- void regulator_init (void)
- void regulator_registers_defaults (void)
- int16_t regulator_position_to_pwm (int16_t position)

7.30.1 Dokumentacja funkcji

7.30.1.1 void regulator_init (void)

Odwołania w main().

7.30.1.2 int16_t regulator_position_to_pwm (int16_t *position*)

Odwołania w main().

7.30.1.3 void regulator_registers_defaults (void)

Odwołania w registers_defaults().

7.31 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

7.31.1 Dokumentacja definicji

7.31.1.1 #define DATA_BITS_MASK_UCSR0B 0x04

Definicja w linii 26 pliku rs485.c.

Odwołania w UartSetDataBits().

7.31.1.2 #define DATA_BITS_MASK_UCSR0C 0x06

Definicja w linii 25 pliku rs485.c.

Odwołania w UartSetDataBits().

7.31.1.3 #define FRAMELENGTH 9

Definicja w linii 36 pliku rs485.c.

7.31.1.4 #define MAX_int16_t 32767

Definicja w linii 21 pliku rs485.c.

7.31.1.5 #define MAX_int8_t 127

Definicja w linii 19 pliku rs485.c.

7.31.1.6 #define MAX_S32 2147483647

Definicja w linii 23 pliku rs485.c.

7.31.1.7 #define MAX_U32 4294967295

Definicja w linii 14 pliku rs485.c.

7.31.1.8 #define MAX_uint16_t 65535

Definicja w linii 13 pliku rs485.c.

7.31.1.9 #define MAX_uint8_t 255

Definicja w linii 12 pliku rs485.c.

7.31.1.10 #define MIN_int16_t -32768

Definicja w linii 20 pliku rs485.c.

7.31.1.11 #define MIN_int8_t -128

Definicja w linii 18 pliku rs485.c.

7.31.1.12 #define MIN_S32 -2147483648

Definicja w linii 22 pliku rs485.c.

7.31.1.13 #define PARITY_BITS_MASK 0x30

Definicja w linii 27 pliku rs485.c.

Odwołania w UartSetParity().

7.31.1.14 #define STOP_BITS_MASK 0x80

Definicja w linii 28 pliku rs485.c.

Odwołania w UartSetStopBits().

7.31.1.15 #define WAITONADDRESS 1

Definicja w linii 38 pliku rs485.c.

Odwołania w UartGetFrameISR() i UartPutFrameISR().

7.31.1.16 #define WAITONCMD 2

Definicja w linii 39 pliku rs485.c.

Odwołania w UartGetFrameISR() i UartPutFrameISR().

7.31.1.17 #define WAITONCRC1 7

Definicja w linii 44 pliku rs485.c.

Odwołania w UartGetFrameISR() i UartPutFrameISR().

7.31.1.18 #define WAITONCRC2 8

Definicja w linii 45 pliku rs485.c.

Odwołania w UartGetFrameISR() i UartPutFrameISR().

7.31.1.19 #define WAITONDATA1 3

Definicja w linii 40 pliku rs485.c.

Odwołania w UartGetFrameISR() i UartPutFrameISR().

7.31.1.20 #define WAITONDATA2 4

Definicja w linii 41 pliku rs485.c.

Odwołania w UartGetFrameISR() i UartPutFrameISR().

7.31.1.21 #define WAITONDATA3 5

Definicja w linii 42 pliku rs485.c.

Odwołania w UartGetFrameISR() i UartPutFrameISR().

7.31.1.22 #define WAITONDATA4 6

Definicja w linii 43 pliku rs485.c.

Odwołania w UartGetFrameISR() i UartPutFrameISR().

7.31.1.23 #define WAITONSTART 0

Definicja w linii 37 pliku rs485.c.

Odwołano w UartGetFrameISR() i UartPutFrameISR().

7.31.2 Dokumentacja funkcji

7.31.2.1 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
    }

}
```

7.31.2.2 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
}
```

7.31.2.3 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);
}
```

7.31.2.4 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, FALSE, FrameBytesCount, FrameReceived, TRUE, WAITONADDRESS, WAITONCMD, WAITONCRC1, WAITONCRC2, WAITONDATA1, W-AITONDATA2, 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;
}
```

7.31.2.5 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, FALSE, FrameSend, FrameSendBytesCount, WAITONADDRESS, WAITONCMD, WAITONCRC1, WAITONCRC2, WAITONDATA1, W-AITONDATA2, 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;
}
```

7.31.2.6 void UartRxFlush (void)

Definicja w linii 148 pliku rs485.c.

```
{
    uint8_t  dummy;

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

}
```

7.31.2.7 void UartStartTx (void)

Definicja w linii 134 pliku rs485.c.

Odwołuje się do enterCritical, exitCritical i 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();
}
```

7.31.3 Dokumentacja zmiennych

7.31.3.1 Frame cmd

Definicja w linii 64 pliku rs485.c.

7.31.3.2 Frame erro

Definicja w linii 65 pliku rs485.c.

7.31.3.3 volatile uint8_t FrameBytesCount = 0

Definicja w linii 51 pliku rs485.c.

Odwołania w ISR() i UartGetFrameISR().

7.31.3.4 volatile uint16_t FrameErrors = 0

Definicja w linii 56 pliku rs485.c.

Odwołania w ISR().

7.31.3.5 volatile uint16_t FrameOverflows = 0

Definicja w linii 57 pliku rs485.c.

Odwołania w ISR() i SendFrame().

7.31.3.6 volatile bool FrameReceived = FALSE

Definicja w linii 53 pliku rs485.c.

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

7.31.3.7 volatile bool FrameSend = FALSE

Definicja w linii 54 pliku rs485.c.

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

7.31.3.8 volatile uint8_t FrameSendBytesCount = 0

Definicja w linii 52 pliku rs485.c.

Odwołania w UartPutFrameISR().

7.31.3.9 volatile Frame InFrame

Definicja w linii 60 pliku rs485.c.

7.31.3.10 volatile Frame OutFrame

Definicja w linii 61 pliku rs485.c.

7.31.3.11 volatile uint16_t ReceiveErrors = 0

Definicja w linii 58 pliku rs485.c.

Odwołania w ISR().

7.31.3.12 Frame response

Definicja w linii 63 pliku rs485.c.

7.32 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.

7.33 Dokumentacja pliku seek.c

7.34 Dokumentacja pliku seek.h

7.35 Dokumentacja pliku timer.c

```
#include <inttypes.h>
#include "config.h"
#include "timer.h"
```

7.36 Dokumentacja pliku timer.h

```
#include "registers.h"
```

7.37 Dokumentacja pliku twi.c

```
#include <inttypes.h>
#include <avr/io.h>
#include <avr/interrupt.h>
#include "openservo.h"
#include "config.h"
#include "registers.h"
#include "twi.h"
```

Definicje

- #define TWI_RX_BUFFER_SIZE (4)
- #define TWI_RX_BUFFER_MASK (TWI_RX_BUFFER_SIZE - 1)
- #define TWI_CHK_WRITE_BUFFER_SIZE (16)
- #define TWI_CHK_WRITE_BUFFER_MASK (TWI_CHK_WRITE_BUFFER_SIZE - 1)

- #define TWI_ACK (0x00)
- #define TWI_NAK (0x01)
- #define TWI_DATA_STATE_COMMAND (0x00)
- #define TWI_DATA_STATE_DATA (0x01)

Funkcje

- void twi_slave_init (uint8_t slave_address)
- uint8_t twi_receive_byte (void)
- uint8_t twi_data_in_receive_buffer (void)

7.37.1 Dokumentacja definicji

7.37.1.1 #define TWI_ACK (0x00)

Definicja w linii 58 pliku twi.c.

7.37.1.2 #define TWI_CHK_WRITE_BUFFER_MASK (TWI_CHK_WRITE_BUFFER_SIZE - 1)

Definicja w linii 49 pliku twi.c.

7.37.1.3 #define TWI_CHK_WRITE_BUFFER_SIZE (16)

Definicja w linii 48 pliku twi.c.

7.37.1.4 #define TWI_DATA_STATE_COMMAND (0x00)

Definicja w linii 62 pliku twi.c.

7.37.1.5 #define TWI_DATA_STATE_DATA (0x01)

Definicja w linii 63 pliku twi.c.

7.37.1.6 #define TWI_NAK (0x01)

Definicja w linii 59 pliku twi.c.

7.37.1.7 #define TWI_RX_BUFFER_MASK (TWI_RX_BUFFER_SIZE - 1)

Definicja w linii 42 pliku twi.c.

Odwołania w twi_receive_byte().

7.37.1.8 #define TWI_RX_BUFFER_SIZE (4)

Definicja w linii 41 pliku twi.c.

7.37.2 Dokumentacja funkcji

7.37.2.1 uint8_t twi_data_in_receive_buffer (void)

Definicja w linii 481 pliku twi.c.

```
{
    // Return 0 (FALSE) if the receive buffer is empty.
    return (twi_rxhead != twi_rxtail);
}
```

7.37.2.2 uint8_t twi_receive_byte (void)

Definicja w linii 467 pliku twi.c.

Odwołuje się do TWI_RX_BUFFER_MASK.

```
{
    // Wait for data in the buffer.
    while (twi_rxhead == twi_rxtail);

    // Calculate buffer index.
    twi_rxtail = (twi_rxtail + 1 ) & TWI_RX_BUFFER_MASK;

    // Return data from the buffer.
    return twi_rxbuf[twi_rxtail];
}
```

7.37.2.3 void twi_slave_init (uint8_t *slave_address*)

Definicja w linii 412 pliku twi.c.

```
{
    // Flush the buffers.
    twi_rxtail = 0;
    twi_rxhead = 0;

#if defined(__AVR_ATtiny45__) || defined(__AVR_ATtiny85__)
    // Set the slave address.
    twi_slave_address = slave_address & 0x7f;

    // Set the interrupt enable, wire mode and clock settings.  Note: At this
    // time the wire mode must not be set to hold the SCL line low when the
    // counter overflows. Otherwise, this TWI slave will interfere with other
    // TWI slaves.
    USICR = (0<<USISIE) | (0<<USIOIE) |                 // Disable start
       condition and overflow interrupt.
            (1<<USIWM1) | (0<<USIWM0) |                 // Set USI to two-wire
       mode without clock stretching.
            (1<<USICS1) | (0<<USICS0) | (0<<USICLK) |   // Shift Register Clock
       Source = External, positive edge
            (0<<USITC);                                 // No toggle of clock
       pin.

    // Clear the interrupt flags and reset the counter.
    USISR = (1<<USISIF) | (1<<USIOIF) | (1<<USIPF) |        // Clear interrupt
       flags.
            (0x0<<USICNT0);                                 // USI to sample 8
       bits or 16 edge toggles.

    // Configure SDA.
    DDR_USI &= ~(1<<DD_SDA);
    PORT_USI &= ~(1<<P_SDA);

    // Configure SCL.
    DDR_USI |= (1<<DD_SCL);
    PORT_USI |= (1<<P_SCL);

    // Start condition interrupt enable.
    USICR |= (1<<USISIE);
#endif // __AVR_ATtiny45__ || __AVR_ATtiny85____

#if defined(__AVR_ATmega8__) || defined(__AVR_ATmega88__) ||
       defined(__AVR_ATmega168__)
    // Set own TWI slave address.
    TWAR = slave_address << 1;

    // Default content = SDA released.
    TWDR = 0xFF;

    // Initialize the TWI interrupt to wait for a new event.
    TWCR = (1<<TWEN) |                                  // Keep the TWI
       interface enabled.
           (1<<TWIE) |                                  // Keep the TWI
       interrupt enabled.
           (0<<TWSTA) |                                 // Don't generate start
       condition.
           (0<<TWSTO) |                                 // Don't generate stop
       condition.
           (1<<TWINT) |                                 // Clear the TWI
       interrupt.
           (1<<TWEA) |                                  // Acknowledge the
       data.
```

```
            (0<<TWWC);                                      //
#endif // __AVR_ATmega8__ || __AVR_ATmega88__ || __AVR_ATmega168__
}
```

7.38 Dokumentacja pliku twi.h

Definicje

- #define TWI_CMD_RESET 0x80
- #define TWI_CMD_CHECKED_TXN 0x81
- #define TWI_CMD_PWM_ENABLE 0x82
- #define TWI_CMD_PWM_DISABLE 0x83
- #define TWI_CMD_WRITE_ENABLE 0x84
- #define TWI_CMD_WRITE_DISABLE 0x85
- #define TWI_CMD_REGISTERS_SAVE 0x86
- #define TWI_CMD_REGISTERS_RESTORE 0x87
- #define TWI_CMD_REGISTERS_DEFAULT 0x88
- #define TWI_CMD_EEPROM_ERASE 0x89
- #define TWI_CMD_VOLTAGE_READ 0x90
- #define TWI_CMD_CURVE_MOTION_ENABLE 0x91
- #define TWI_CMD_CURVE_MOTION_DISABLE 0x92
- #define TWI_CMD_CURVE_MOTION_RESET 0x93
- #define TWI_CMD_CURVE_MOTION_APPEND 0x94

Funkcje

- void twi_slave_init (uint8_t)
- uint8_t twi_receive_byte (void)
- uint8_t twi_data_in_receive_buffer (void)

7.38.1 Dokumentacja definicji

7.38.1.1 #define TWI_CMD_CHECKED_TXN 0x81

Definicja w linii 31 pliku twi.h.

7.38.1.2 #define TWI_CMD_CURVE_MOTION_APPEND 0x94

Definicja w linii 44 pliku twi.h.

Odwołania w RS485CMD().

7.38.1.3 #define TWI_CMD_CURVE_MOTION_DISABLE 0x92

Definicja w linii 42 pliku twi.h.

Odwołania w RS485CMD().

7.38.1.4 #define TWI_CMD_CURVE_MOTION_ENABLE 0x91

Definicja w linii 41 pliku twi.h.

Odwołania w RS485CMD().

7.38.1.5 #define TWI_CMD_CURVE_MOTION_RESET 0x93

Definicja w linii 43 pliku twi.h.

Odwołania w RS485CMD().

7.38.1.6 #define TWI_CMD_EEPROM_ERASE 0x89

Definicja w linii 39 pliku twi.h.

Odwołania w RS485CMD().

7.38.1.7 #define TWI_CMD_PWM_DISABLE 0x83

Definicja w linii 33 pliku twi.h.

Odwołania w RS485CMD().

7.38.1.8 #define TWI_CMD_PWM_ENABLE 0x82

Definicja w linii 32 pliku twi.h.

Odwołania w RS485CMD().

7.38.1.9 #define TWI_CMD_REGISTERS_DEFAULT 0x88

Definicja w linii 38 pliku twi.h.

Odwołania w RS485CMD().

7.38.1.10 #define TWI_CMD_REGISTERS_RESTORE 0x87

Definicja w linii 37 pliku twi.h.

Odwołania w RS485CMD().

7.38.1.11 #define TWI_CMD_REGISTERS_SAVE 0x86

Definicja w linii 36 pliku twi.h.

Odwołania w RS485CMD().

7.38.1.12 #define TWI_CMD_RESET 0x80

Definicja w linii 30 pliku twi.h.

Odwołania w RS485CMD().

7.38.1.13 #define TWI_CMD_VOLTAGE_READ 0x90

Definicja w linii 40 pliku twi.h.

Odwołania w RS485CMD().

7.38.1.14 #define TWI_CMD_WRITE_DISABLE 0x85

Definicja w linii 35 pliku twi.h.

Odwołania w RS485CMD().

7.38.1.15 #define TWI_CMD_WRITE_ENABLE 0x84

Definicja w linii 34 pliku twi.h.

Odwołania w RS485CMD().

7.38.2 Dokumentacja funkcji

7.38.2.1 uint8_t twi_data_in_receive_buffer (void)

Definicja w linii 481 pliku twi.c.

```
{
    // Return 0 (FALSE) if the receive buffer is empty.
    return (twi_rxhead != twi_rxtail);
}
```

7.38.2.2 uint8_t twi_receive_byte (void)

Definicja w linii 467 pliku twi.c.

Odwołuje się do TWI_RX_BUFFER_MASK.

```
{
    // Wait for data in the buffer.
    while (twi_rxhead == twi_rxtail);

    // Calculate buffer index.
    twi_rxtail = (twi_rxtail + 1 ) & TWI_RX_BUFFER_MASK;

    // Return data from the buffer.
    return twi_rxbuf[twi_rxtail];
}
```

7.38.2.3 void twi_slave_init (uint8_t)

Definicja w linii 412 pliku twi.c.

```
{
    // Flush the buffers.
    twi_rxtail = 0;
    twi_rxhead = 0;

#if defined(__AVR_ATtiny45__) || defined(__AVR_ATtiny85__)
    // Set the slave address.
    twi_slave_address = slave_address & 0x7f;

    // Set the interrupt enable, wire mode and clock settings.  Note: At this
    // time the wire mode must not be set to hold the SCL line low when the
    // counter overflows. Otherwise, this TWI slave will interfere with other
    // TWI slaves.
    USICR = (0<<USISIE) | (0<<USIOIE) |                 // Disable start
       condition and overflow interrupt.
            (1<<USIWM1) | (0<<USIWM0) |                 // Set USI to two-wire
       mode without clock stretching.
            (1<<USICS1) | (0<<USICS0) | (0<<USICLK) |   // Shift Register Clock
       Source = External, positive edge
            (0<<USITC);                                 // No toggle of clock
       pin.

    // Clear the interrupt flags and reset the counter.
    USISR = (1<<USISIF) | (1<<USIOIF) | (1<<USIPF) |        // Clear interrupt
       flags.
            (0x0<<USICNT0);                                 // USI to sample 8
       bits or 16 edge toggles.

    // Configure SDA.
    DDR_USI &= ~(1<<DD_SDA);
    PORT_USI &= ~(1<<P_SDA);

    // Configure SCL.
    DDR_USI |= (1<<DD_SCL);
    PORT_USI |= (1<<P_SCL);

    // Start condition interrupt enable.
    USICR |= (1<<USISIE);
#endif // __AVR_ATtiny45__ || __AVR_ATtiny85____

#if defined(__AVR_ATmega8__) || defined(__AVR_ATmega88__) ||
       defined(__AVR_ATmega168__)
    // Set own TWI slave address.
    TWAR = slave_address << 1;

    // Default content = SDA released.
    TWDR = 0xFF;

    // Initialize the TWI interrupt to wait for a new event.
    TWCR = (1<<TWEN) |                                  // Keep the TWI
       interface enabled.
           (1<<TWIE) |                                  // Keep the TWI
       interrupt enabled.
           (0<<TWSTA) |                                 // Don't generate start
       condition.
           (0<<TWSTO) |                               // Don't generate stop
       condition.
           (1<<TWINT) |                               // Clear the TWI
       interrupt.
           (1<<TWEA) |                                // Acknowledge the
       data.
           (0<<TWWC);                                 //
#endif // __AVR_ATmega8__ || __AVR_ATmega88__ || __AVR_ATmega168__
}
```

7.39 Dokumentacja pliku watchdog.c

```
#include <inttypes.h>
#include <avr/io.h>
#include "openservo.h"
#include "config.h"
#include "pwm.h"
```

Funkcje

- void watchdog_init (void)
- void watchdog_hard_reset (void)

7.39.1 Dokumentacja funkcji

7.39.1.1 void watchdog_hard_reset (void)

Definicja w linii 73 pliku watchdog.c.

Odwołania w RS485CMD().

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

#if defined(__AVR_ATtiny45__) || defined(__AVR_ATtiny85__)
    // Enable the watchdog.
    WDTCR = (1<<WDIF) |                                    // Reset any
       interrupt.
            (0<<WDIE) |                                    // Disable
       interrupt.
            (1<<WDE) |                                     // Watchdog enable.
            (0<<WDP3) | (0<<WDP2) | (0<<WDP1) | (0<<WDP0);  // Minimum
       prescaling - 16mS.
#endif

#if defined(__AVR_ATmega8__)
    // Enable the watchdog.
    WDTCR = (0<<WDCE) |                                     // Don't set
       change enable.
            (1<<WDE) |                                      // Watchdog
       enable.
            (0<<WDP2) | (0<<WDP1) | (0<<WDP0);              // Minimum
       prescaling - 16mS.
#endif

#if defined(__AVR_ATmega88__) || defined(__AVR_ATmega168__)
    // Enable the watchdog.
    WDTCSR = (1<<WDIF) |                                    // Reset any
       interrupt.
             (0<<WDIE) |                                    // Disable
       interrupt.
             (1<<WDE) |                                     // Watchdog
       enable.
             (0<<WDP3) | (0<<WDP2) | (0<<WDP1) | (0<<WDP0);  // Minimum
       prescaling - 16mS.
#endif

    // Wait for reset to occur.
    for (;;);
}
```

7.39.1.2 void watchdog_init (void)

Definicja w linii 38 pliku watchdog.c.

Odwołania w main().

```
{
#if defined(__AVR_ATtiny45__) || defined(__AVR_ATtiny85__)
    // Clear WDRF in MCUSR.
    MCUSR = 0x00;

    // Write logical one to WDCE and WDE.
    WDTCR |= (1<<WDCE) | (1<<WDE);

    // Turn off WDT.
    WDTCR = 0x00;
#endif

#if defined(__AVR_ATmega8__)
    // Write logical one to WDCE and WDE.
    WDTCR |= (1<<WDCE) | (1<<WDE);

    // Turn off WDT.
    WDTCR = 0x00;
#endif

#if defined(__AVR_ATmega88__) || defined(__AVR_ATmega168__)
    // Clear WDRF in MCUSR.
    MCUSR &= ~(1<<WDRF);

    // Write logical one to WDCE and WDE.
    WDTCSR |= (1<<WDCE) | (1<<WDE);

    // Turn off WDT.
    WDTCSR = 0x00;
#endif
}
```

7.40 Dokumentacja pliku watchdog.h

Funkcje

- void watchdog_init (void)
- void watchdog_hard_reset (void)

7.40.1 Dokumentacja funkcji

7.40.1.1 void watchdog_hard_reset (void)

Definicja w linii 73 pliku watchdog.c.

Odwołania w RS485CMD().

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

#if defined(__AVR_ATtiny45__) || defined(__AVR_ATtiny85__)
    // Enable the watchdog.
    WDTCR = (1<<WDIF) |                                         // Reset any
       interrupt.
            (0<<WDIE) |                                         // Disable
       interrupt.
            (1<<WDE) |                                          // Watchdog enable.
            (0<<WDP3) | (0<<WDP2) | (0<<WDP1) | (0<<WDP0);      // Minimum
       prescaling - 16mS.
#endif

#if defined(__AVR_ATmega8__)
    // Enable the watchdog.
    WDTCR = (0<<WDCE) |                                          // Don't set
       change enable.
            (1<<WDE) |                                           // Watchdog
       enable.
            (0<<WDP2) | (0<<WDP1) | (0<<WDP0);                   // Minimum
       prescaling - 16mS.
#endif

#if defined(__AVR_ATmega88__) || defined(__AVR_ATmega168__)
    // Enable the watchdog.
    WDTCSR = (1<<WDIF) |                                      // Reset any
       interrupt.
             (0<<WDIE) |                                      // Disable
       interrupt.
             (1<<WDE) |                                       // Watchdog
       enable.
             (0<<WDP3) | (0<<WDP2) | (0<<WDP1) | (0<<WDP0);  // Minimum
       prescaling - 16mS.
#endif

    // Wait for reset to occur.
    for (;;);
}
```

7.40.1.2 void watchdog_init (void)

Definicja w linii 38 pliku watchdog.c.

Odwołania w main().

```
{
#if defined(__AVR_ATtiny45__) || defined(__AVR_ATtiny85__)
    // Clear WDRF in MCUSR.
    MCUSR = 0x00;

    // Write logical one to WDCE and WDE.
    WDTCR |= (1<<WDCE) | (1<<WDE);

    // Turn off WDT.
    WDTCR = 0x00;
#endif

#if defined(__AVR_ATmega8__)
    // Write logical one to WDCE and WDE.
    WDTCR |= (1<<WDCE) | (1<<WDE);

    // Turn off WDT.
    WDTCR = 0x00;
#endif

#if defined(__AVR_ATmega88__) || defined(__AVR_ATmega168__)
    // Clear WDRF in MCUSR.
    MCUSR &= ~(1<<WDRF);

    // Write logical one to WDCE and WDE.
    WDTCSR |= (1<<WDCE) | (1<<WDE);

    // Turn off WDT.
    WDTCSR = 0x00;
#endif
}
```