Embedded Linux Code

1.0

Generated by Doxygen 1.8.14

Contents

Chapter 1

Main Page

1.1 Introduction

The software program "focserver" implements a web server and a websocket server in the Field-Oriented Control (FOC) system, developed with Xilinx SDSoC tools.

1.2 Synopsis

Command line:

```
focserver [-c filename] [-D] [-d <log bitfield>] [-f filepath] [-h]
      [-p] [-s[speed]] [-t] [-v] [-w reg=val] [-W www-directory]
```

See the Table 1.1 for the detailed description of command line options.

Table 1.1 Command line options

Option	Description
-c filename	Capture ADC data and write it to a file, don't start the server
-D	Start the server as a daemon
-d bitfield	Set Libwebsocket debug log bitfield. Example values: 0 log nothing, 255: log everything
-f filepath	Use the given configuration file
-h	Show this text
-p	Print values of all registers, don't start the server
-s[speed]	Start the motor. The speed (in RPM) is optional
-v	Print version information and exit
-t	Test flag
-w reg=val	Write the value to the register, don't start the server
-W directory	Document root directory for the web server

Executing "focserver" will start it in server mode. Unless the option *-c*, *-v* or *-w* was supplied on the command line, the program "focserver" will perform as follows:

1. Read the configuration file; see section Configuration file for the format and location.
2. Open the hardware devices; see section Requirements for the Linux operating system for the devices required.
3. Start the internal web server. Default document root directory is "/usr/share/focserver".
4. Blink the heartbeat LED LD3 on the Arty Z7 board once per second; this features is not available in the SDSoC FOC project.

1.3 Configuration file

The configuration file is in JSON format and contains just one JSON object with FOC parameters and initialization values for the parameter registers. See the Table 1.2 for the list of supported fields. The complete list of parameter register names can be found in the the *Network API*, table "Parameter registers".

The default path for the configuration file is "/etc/focserver.conf".

Table 1.2 Fields in the configuration file

Field	Description	Default value
ppr	Pulses per revolution	1000
adc2A	Conversion factor from ADC samples to amperes	0.00039
pwm2V	Conversion factor from PWM duty cycle to volts	0.0003662
init	Parameter register values to be written during initialization	See the Table 1.3
speed	Parameter register values to be written before starting the motor in a speed control loop	
torque	Parameter register values to be written before starting the motor in a current control loop	

Table 1.3 The default initialization values

Step #	Name of SDSoC FOC register	Value
1	FluxSp	0
2	FluxKp	-4096
3	FluxKi	0
4	RPMSp	3000
5	RPMKp	0
6	RPMKi	-10
7	TorqueSp	1000
8	TorqueKp	5000
9	TorqueKi	0
10	Shift	719
11	Vd	-7424
12	Vq	-16128
13	Fa	18120
14	Fb	14647
15	Control2	10

The configuration file as used in the SDSoC FOC design:

```
{
    "init" : {
        "FluxSp" : 0,
        "FluxKp" : -4096,
        "FluxKi" : 0,
        "RPMSp" : 3000,
        "RPMKp" : -200,
        "RPMKi" : -5,
        "Shift" : 719,
        "Vd" : -7424,
        "Vq" : -16128,
        "Fa" : 18120,
        "Fb" : 14647,
        "Mode" : 0,
        "FixedDelay" : 20
    },
    "speed" : {
        "TorqueSp" : 0,
        "TorqueKp" : 5000,
        "TorqueKi" : 0
    },
    "torque" : {
        "TorqueKp" : -20000,
        "TorqueKi" : -5000
    },

    "ppr" : 1000,
    "adc2A" : 0.00039,
    "pwm2V" : 0.0003662
}
```

1.4 Requirements for the Linux operating system

The program "focserver" expects the following hardware to be available on the Linux system:

1. The capture device IP core as the UIO device "AXI-Data-Capture".
2. The FOC IP core, either through the UIO device named "foc" (HLS FOC project) or the name in the device tree must have the prefix "xlnx,foc-" (SDSoC FOC project).

For the reference, following are the device tree overrides as used in SDSoC FOC design:

```
&AXI_StreamCapture_0 {
    compatible = "trenz.biz,smartio-1.0";
    trenz.biz,name = "AXI-Data-Capture";
    trenz.biz,buffer-size = <0x400000>;
    trenz.biz,sample-rate = <78125>;
    xlnx,cdata-width = <16>;
    xlnx,channels = <4>;
};
```

In order to fulfill the above conditions automatically, it is recommended to use the Petalinux project provided in the IIoT-EDDP repository as a basis as follows:

1. In the Vivado SDx IDE: Create and build an SDSoC application based on the SDSoC platform and utilizing the FOC IP core.
2. Locate the Hardware Definition File (∗.hdf) and copy it to the root of your copy of the Petalinux project.

3. On the Linux command line and in the Petalinux project directory, run the following command:

   petalinux-config –get-hw-description

   This will update the Petalinux project and the device tree with the latest hardware information.
4. Build your petalinux project.
5. Copy the files "images/linux/image.ub" and "images/linux/u-boot.elf" to the SDSoC platform, overriding existing files.
6. Rebuild your SDSoC application for the changes to be effective.

Important: Using the Hardware Definition File (∗.hdf) from the platform Vivado project will not work, because it doesn't have the necessary information for the device tree.

1.5 Startup script "focinit"

A startup script named "focserver" is provided for use in the Petalinux project for TEC0053.

At the Linux startup, this script executes as follows:

1. Mount the SD card temporarily in order to execute the script named "init.sh" on it if found.
2. Start the FOC server if not started by the the script "init.sh" beforehand.
3. Wait 10 seconds before setting the IP address to the default of 192.168.42.123
4. Start the FOC server if the file "init.sh" was not found on the SD card.

1.6 Building from the source

By including "focserver" in a Petalinux project it will be automatically rebuilt from the source as needed.

To regenerate the Doxygen documentation, run the script "run_doxygen.bat".

1.7 Tools

The tools required are listed in the Table 1.4. For the documentation Doxygen is used; the documentation is generated from the doxygen-formatted comments in the the source code files.

Table 1.4 Tools

Tool	Version	Notes
Xilinx SDK	2017.1	Development environment for developing bare-metal and Linux software
PetaLinux	2017.1	Xilinx tool for building embedded Linux systems
Doxygen	1.8.11	Documentation extraction
MiKTeX	2.9	PDF generation

Chapter 2

Class Index

2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

FocServer::BinaryHeader
Layout of the binary header . . . 9

DeviceTreeDevice
Fetch information from the Linux Device Tree . . . 9

FocConfiguration
Configuration of the FOC server . . . 14

FocDevice
Access to the FOC IP core . . . 16

FocServer
FOC server implementing the *Network API* and a web server, which permits control and monitor of the FOC system from the Web UI . . . 30

FocConfiguration::ParameterValue
Value of a parameter in the configuration file . . . 32

FocDevice::RegisterAccess
Description of access to a register in a register bank . . . 33

WebsocketBuffer
Write buffer, consisting of a queue of the messages to be written and a write buffer for the libwebsockets . . . 34

Chapter 3

File Index

3.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 4

Class Documentation

4.1 FocServer::BinaryHeader Struct Reference

Layout of the binary header.

```
#include <FocServer.h>
```

Public Attributes

- uint16_t nchannels

  Bytes 0..1: Number of channels.
- uint16_t nsamples

  Bytes 2..3: Number of samples.
- uint32_t sample_rate

  Bytes 4..7: Sample rate.
- uint8_t name [BINARY_HEADER_SIZE - 2u - 2u - 4u]

  Name of the data.

4.1.1 Detailed Description

Layout of the binary header.

Definition at line 84 of file FocServer.h.

The documentation for this struct was generated from the following file:

- src/FocServer.h

4.2 DeviceTreeDevice Class Reference

Fetch information from the Linux Device Tree.

```
#include <DeviceTreeDevice.h>
```

Public Member Functions

- DeviceTreeDevice (const std::string &pDeviceDirectoryPath, const std::string &pName, const std::string &pCompatible, const uintptr_t pAddress, const unsigned int pLength)

  Create new object for fetching data for the given device.
- int readUInt32Array (uint32_t *value, const unsigned int nValues, const char *propertyName) const

  Read one or more UInt32-s.
- int readUInt32 (uint32_t &value, const char *propertyName) const

  Read property as unsigned 32-bit integer.

Static Public Member Functions

- static std::shared_ptr< DeviceTreeDevice > findByProperty (const char *propertyName, const char *propertyValue)

  Find the device by the given property value.
- static void demo ()

  Small demo program of the capabilities, specific to ARTY-Z7 FOC project.

Public Attributes

- const std::string deviceDirectoryPath

  Path to the device directory in the device tree.
- const std::string name

  Name.
- const std::string compatible

  Compatible string.
- const uintptr_t address

  HW-address.
- const unsigned int length

  Length of the memory area that can be mapped.

Static Public Attributes

- static const char *const PROPERTY_COMPATIBLE = "compatible"

  Name of the compatible property: "compatible".
- static const char *const PROPERTY_TRENZ_BIZ_NAME = "trenz.biz,name"

  Name of the name property: "trenz.biz,name".

4.2.1 Detailed Description

Fetch information from the Linux Device Tree.

```
std::shared_ptr<DeviceTreeDevice>   dev = DeviceTreeDevice::findByProperty(
      "compatible", "foc");
if (dev) {
    printf("Device found at %p\n", (void*)dev->address);
} else {
    printf("FOC device not found.\n");
}
```

Definition at line 43 of file DeviceTreeDevice.h.

4.2.2 Constructor & Destructor Documentation

4.2.2.1 DeviceTreeDevice()

```
DeviceTreeDevice::DeviceTreeDevice (
            const std::string & pDeviceDirectoryPath,
            const std::string & pName,
            const std::string & pCompatible,
            const uintptr_t pAddress,
            const unsigned int pLength )
```

Create new object for fetching data for the given device.

Normally, this should not be called directly.

Parameters

pDeviceDirectoryPath	Absolute path to the device in the device tree.
pName	Name of the device.
pCompatible	Value of the device tree property "compatible".
pAddress	First value in the device tree property "reg".
pLength	Second value in the device tree property "reg".

Definition at line 104 of file DeviceTreeDevice.cpp.

```
: deviceDirectoryPath(pDeviceDirectoryPath),
  name(pName),
  compatible(pCompatible),
  address(pAddress),
  length(pLength)
{
}
```

4.2.3 Member Function Documentation

4.2.3.1 findByProperty()

```
std::shared_ptr< DeviceTreeDevice > DeviceTreeDevice::findByProperty (
            const char * propertyName,
            const char * propertyValue )   [static]
```

Find the device by the given property value.

This doesn't throw exceptions, just returns empty shared_ptr on errors.

Parameters

propertyName	Name of the property to search for.
propertyValue	Value of the property to search for.

Returns

Smart pointer to the device; in the case none found, the pointer will be empty.

Definition at line 115 of file DeviceTreeDevice.cpp.

```
{
    const unsigned int  value_length = strlen(propertyValue);

    // Scan the device tree directory for files.
    DIR*    dir = opendir(DEVICE_TREE_DIR);
    if (dir == nullptr) {
        return std::shared_ptr<DeviceTreeDevice>();
    }

    std::string device_dir;
    std::string p_value;
    std::string p_name;
    uint32_t    p_reg[2];

    struct stat st;

    for (struct dirent* ent=readdir(dir); ent!=nullptr; ent=readdir(dir)) {
        ssprintf(device_dir, "%s/%s", DEVICE_TREE_DIR, ent->d_name);

        // Must be directory.
        if (stat(device_dir.c_str(), &st) != 0 || (st.st_mode & S_IFDIR)==0) {
            continue;
        }
        // Compatible string must match.
        if (read_all_text(p_value, device_dir, propertyName) <= 0  || strncmp(p_value.c_str(), propertyValue, std::min<unsigned int>(value_length, p_value.size())) != 0) {
            continue;
        }
        // Parameters must be readable.
        if (read_all_text(p_name, device_dir, "name") <= 0
                || read_uint32_array(p_reg, sizeof(p_reg), device_dir, "reg") < 2) {
            continue;
        }
        closedir(dir);
        std::string p_compatible;
        if (strcmp(propertyName, PROPERTY_COMPATIBLE)==0) {
            p_compatible = propertyValue;
        }
        else {
            read_all_text(p_compatible, device_dir, PROPERTY_COMPATIBLE);
        }
        return std::make_shared<DeviceTreeDevice>(device_dir, p_name, p_compatible, p_reg[0], p_reg[1]);
    }
    closedir(dir);

    // Nothing found :(
    return std::shared_ptr<DeviceTreeDevice>();
}
```

4.2.3.2 readUInt32()

```
int DeviceTreeDevice::readUInt32 (
            uint32_t & value,
            const char * propertyName ) const
```

Read property as unsigned 32-bit integer.

Parameters

value	Buffer to store the value read.
propertyName	Name of the property to read the value from.

Returns

1 on success, 0 when the property doesn't contain enough data, -1 on failure.

Definition at line 171 of file DeviceTreeDevice.cpp.

```
{
    return read_uint32_array(&value, 1, deviceDirectoryPath, propertyName);
}
```

4.2.3.3 readUInt32Array()

```
int DeviceTreeDevice::readUInt32Array (
             uint32_t * value,
             const unsigned int nValues,
             const char * propertyName ) const
```

Read one or more UInt32-s.

Parameters

value	Buffer to store values read.
nValues	Number of values to be read.
propertyName	Name of the property to read values from.

Returns

Number of values read, or -1 on failure.

Definition at line 165 of file DeviceTreeDevice.cpp.

```
{
    return read_uint32_array(value, nValues, deviceDirectoryPath, propertyName);
}
```

The documentation for this class was generated from the following files:

- src/DeviceTreeDevice.h
- src/DeviceTreeDevice.cpp

4.3 FocConfiguration Class Reference

Configuration of the FOC server.

`#include <FocConfiguration.h>`

Classes

- struct ParameterValue

  Value of a parameter in the configuration file.

Public Member Functions

- FocConfiguration ()

  Create new configuration with default values.
- FocConfiguration (const std::string &jsonString)

  Construct configuration from a JSON string.
- void dump ()

  Dump configuration to standard output.

Static Public Member Functions

- static std::shared_ptr< FocConfiguration > fromFile (const std::string &filepath)

  Load configuration from a file.

Public Attributes

- unsigned int ppr

  Pulses per revolution. 0 when undetermined.
- double adc2A

  Conversion factor from ADC units to mA.
- double pwm2V

  Conversion factor from PWM factors to voltages.
- std::vector< ParameterValue > init

  Initialization sequence.
- std::vector< ParameterValue > speed

  Sequence for changing to the speed mode.
- std::vector< ParameterValue > torque

  Sequence for changing to the torque mode.

Static Public Attributes

- static constexpr int INDEX_NOT_KNOWN_YET = -1

  The index corresponding to the name is not known yet.
- static constexpr int INDEX_INVALID_NAME = -2

  The name of the ParameterValue was invalid and no index can be determined.
- static constexpr const char * FILENAME = "/etc/focserver.conf"

  Default name for the configuration file.
- static constexpr double DEFAULT_ADC2A = 0.00039

  Default value for adc2A.
- static constexpr double DEFAULT_PWM2V = 0.0003662

  Default value for pwm2V.

4.3.1 Detailed Description

Configuration of the FOC server.

Definition at line 34 of file FocConfiguration.h.

4.3.2 Constructor & Destructor Documentation

4.3.2.1 FocConfiguration()

```
FocConfiguration::FocConfiguration (
              const std::string & jsonString )
```

Construct configuration from a JSON string.

Throws an exception when the JSON string is faulty.

Parameters

jsonString	String in the JSON format.

Definition at line 96 of file FocConfiguration.cpp.

```
{
    if (jsonString.size() == 0u) {
        throw std::runtime_error("Empty configuration not permitted");
    }
    Json::Reader    reader;
    Json::Value     root;

    if (!reader.parse(&jsonString[0], &jsonString[0] + jsonString.size(), root)) {
        throw std::runtime_error("Invalid JSON");
    }

    // Load the values from JSONCPP.
    ppr = root.get(NAME_PPR, PPR).asInt();
    adc2A = root.get(NAME_ADC2A, DEFAULT_ADC2A).asDouble();
    pwm2V = root.get(NAME_PWM2V, DEFAULT_PWM2V).asDouble();
    load_params(init, root, NAME_INIT);
    load_params(speed, root, NAME_SPEED);
    load_params(torque, root, NAME_TORQUE);
}
```

4.3.3 Member Function Documentation

4.3.3.1 fromFile()

```
std::shared_ptr< FocConfiguration > FocConfiguration::fromFile (
              const std::string & filepath )  [static]
```

Load configuration from a file.

Throws exceptions when the file is faulty or non-existent.

Parameters

filepath	Path to the file to be read.

Returns

Smart pointer to the configuration.

Definition at line 118 of file FocConfiguration.cpp.

```
{
    std::string s = File::readAllText(filepath);
    return std::make_shared<FocConfiguration>(s);
}
```

The documentation for this class was generated from the following files:

- src/FocConfiguration.h
- src/FocConfiguration.cpp

4.4 FocDevice Class Reference

Access to the FOC IP core.

`#include <FocDevice.h>`

Classes

- struct RegisterAccess

    Description of access to a register in a register bank.

Public Types

- enum PSEUDO_PARAMETER : unsigned int { MODE = PSEUDO_PARAMETER_OFFSET, FIXED_PERIOD, SPREAD_SPECTRUM }

    Pseudo register indices.
- enum RegisterType : uint32_t { RegisterType::INT32, RegisterType::UINT32 }

    Type of a register.

Public Member Functions

- FocDevice (std::shared_ptr< FocConfiguration > pConfig)

  Create new FOC device object.
- FocDevice ()

  Create new FOC device object with the default configuration.
- uintptr_t getBaseAddress () const

  Get the base address.
- void writeParameter (const unsigned int parameterIndex, const uint32_t parameterValue)

  Write parameter register.
- uint32_t readParameter (const unsigned int parameterIndex)

  Read parameter register.
- void readParameterString (std::string &buffer, const unsigned int parameterIndex)

  String representation of the parameter register in the following format: NAME CONVERTED_VALUE REGISTER_↩ VALUE.
- uint32_t readStatus (const unsigned int statusIndex)

  Read status register.
- void readStatusString (std::string &buffer, const unsigned int statusIndex)

  String representation of the status register in the following format: NAME VALUE VALUE.
- void defaultInit ()

  Perform default initialization.
- void startMotor (const unsigned int mode, smart::hw::AxiDataCapture ∗capture)

  Start the motor in the given mode.
- void stopMotor ()

  Stop the motor.
- void writeCaptureSource (const unsigned int sourceIndex)

  Set new capture source.
- unsigned int readCaptureSource () const

  Read the capture source index.
- unsigned int readLeds ()

  Read LED-s state.
- void resetError ()

  Reset the error flag of the speed monitor.
- void writeLeds (const uint32_t leds)

  Write led state.
- void writeErrorLimit (const unsigned int error_limit)

  Write error limit.
- unsigned int readErrorLimit ()

  Read the test error limit.
- void writeDecimate (const unsigned int decimationFactor)

  Write decimation factor (number of samples to skip for every sample captured).
- unsigned int readDecimate () const

  Read decimation factor.
- void writeSpreadSpectrum (const bool enableSpreadSpectrum)

  Write the spread spectrum flag.
- bool readSpreadSpectrum ()

  Read the spread spectrum flag.

Public Attributes

- const char ∗ designName

  Name of the HW design the software is running on.
- std::shared_ptr< FocConfiguration > config

  Configuration. This will be created anew if not existing.
- unsigned int parameterCount

  Number of parameter registers.
- const RegisterAccess ∗ parameterRegisters

  List of the known parameter registers. End marker: nullptr as name.
- unsigned int statusCount

  Number of status registers.
- const RegisterAccess ∗ statusRegisters

  List of the known status registers. End marker: nullptr as name.

Static Public Attributes

- static const char ∗const NAME_SDSOC = "SDSoC"

  Name of the SDSoC design, constant string "SDSoc".
- static const char ∗const NAME_HLS = "HLS"

  Name of the HLS design, constant string "HLS".
- static const char ∗const NAME_UNKNOWN = "Unknown"

  Name of an unknown design.
- static constexpr unsigned int PSEUDO_PARAMETER_OFFSET = 16u

  Offset to the pseudo registers.

4.4.1 Detailed Description

Access to the FOC IP core.

Example code:

```
FocDevice   dev;

dev.writeParameter(RPM_SP_REG, 1000);
dev.startMotor(CONTROL_SPEED);
```

Definition at line 49 of file FocDevice.h.

4.4.2 Member Enumeration Documentation

4.4.2.1 PSEUDO_PARAMETER

```
enum FocDevice::PSEUDO_PARAMETER : unsigned int
```

Pseudo register indices.

This is common for both parameters and status registers.

Enumerator

MODE	Operating mode of the FOC.
FIXED_PERIOD	Fixed speed increment.
SPREAD_SPECTRUM	Spread spectrum register.

Definition at line 65 of file FocDevice.h.

```
                        : unsigned int {
        /// Operating mode of the FOC.
        MODE = PSEUDO_PARAMETER_OFFSET,
        /// Fixed speed increment.
        FIXED_PERIOD,
        /// Spread spectrum register.
        SPREAD_SPECTRUM,
    };
```

4.4.2.2 RegisterType

enum FocDevice::RegisterType : uint32_t [strong]

Type of a register.

Enumerator

INT32	Signed 32-bit integer.
UINT32	Unsigned 32-bit integer.

Definition at line 76 of file FocDevice.h.

```
                          : uint32_t {
        /// Signed 32-bit integer.
        INT32,
        /// Unsigned 32-bit integer.
        UINT32
    };
```

4.4.3 Constructor & Destructor Documentation

4.4.3.1 FocDevice()

```
FocDevice::FocDevice (
            std::shared_ptr< FocConfiguration > pConfig )
```

Create new FOC device object.

Setup the default configuration.

Definition at line 121 of file FocDevice.cpp.

```
: designName(NAME_UNKNOWN),
  config(pConfig),
  parameterCount(0),
  parameterRegisters(parameter_registers),
  statusCount(0),
  statusRegisters(status_registers),
  _parameter_registers_offset(0x10),
  _status_registers_offset(0x20)
{
    // Are we running on SDSoC or HLS?
    _sdsoc_info = DeviceTreeDevice::findByProperty(
  DeviceTreeDevice::PROPERTY_COMPATIBLE, FOC_COMPATIBLE_DEVICE_PREFIX);
    if (_sdsoc_info) {
        designName = NAME_SDSOC;
        _sdsoc_device = std::unique_ptr<smart::MappedFile>(new smart::MappedFile(FILENAME_DEV_MEM,
  _sdsoc_info->address, MappedFile::pageSize()));
        _registers = _sdsoc_device.get();
        _hw_address = _sdsoc_info->address;
    }
    else {
        designName = NAME_HLS;
        _hls_device = std::unique_ptr<smart::UioDevice>(new smart::UioDevice(UIO_FOC_DEVICE_NAME));
        _registers = _hls_device->getRequiredMap(0);
        _hw_address = _hls_device->maps[0].addr;
    }

    unsigned int i;

    for (i=0; parameterRegisters[i].name!=nullptr; ++i) {
    }
    parameterCount = i;

    for (i=0; statusRegisters[i].name!=nullptr; ++i) {
    }
    statusCount = i;

    /// Setup the default configuration.
    if (!config) {
        config = std::make_shared<FocConfiguration>();
        add_parameter_value(config->init, parameterRegisters, CONTROL_REG, 0);
  // Motor OFF
        add_parameter_value(config->init, parameterRegisters,
  PSEUDO_PARAMETER::FIXED_PERIOD, 50);          // Reasonably slow rotation.
        add_parameter_value(config->init, parameterRegisters, FLUX_SP_REG, 0);
  // Flux Sp = 0
        add_parameter_value(config->init, parameterRegisters, FLUX_KP_REG, 0
  xFFFFF000); // Flux Kp = -4096
        add_parameter_value(config->init, parameterRegisters, FLUX_KI_REG, 0);
  // Flux Ki = 0
        add_parameter_value(config->init, parameterRegisters, TORQUE_SP_REG, 0);
  // Torque Sp (used only in debug modes)
        add_parameter_value(config->init, parameterRegisters, TORQUE_KP_REG, 5000);
  // Torque Kp = 1.0
        add_parameter_value(config->init, parameterRegisters, TORQUE_KI_REG, 0);
  // Torque Ki = 0
        add_parameter_value(config->init, parameterRegisters, RPM_SP_REG, 3000);
  // Speed Sp = 3000 RPM
        add_parameter_value(config->init, parameterRegisters, RPM_KP_REG, -200);
  // Speed Kp = 2.88
        add_parameter_value(config->init, parameterRegisters, RPM_KI_REG, -5);
  // Speed Ki
        add_parameter_value(config->init, parameterRegisters, ANGLE_SH_REG, 719);
  // Angle between encoder index and Phase A
        add_parameter_value(config->init, parameterRegisters, VD_REG, 0xFFFFE300);
  // Vd (used only in debug modes)
        add_parameter_value(config->init, parameterRegisters, VQ_REG, 0xFFFFc100);
  // Vq (used only in debug modes)
        add_parameter_value(config->init, parameterRegisters, FA_REG, 18120);
  // Filter coefficient A = 0.553
        add_parameter_value(config->init, parameterRegisters, FB_REG, 14647);
  // Filter coefficient A = 0.447

        // The last registers already have suitable default values.
        add_parameter_value(config->init, parameterRegisters, CONTROL2_REG,
  CONTROL2_BV_RESET_ERROR);
        add_parameter_value(config->init, parameterRegisters, CONTROL2_REG, 100u <<
   CONTROL2_BIT_ERROR_LIMIT);

        add_parameter_value(config->speed, parameterRegisters, TORQUE_SP_REG, 0);
        add_parameter_value(config->speed, parameterRegisters, TORQUE_KP_REG, 5000)
  ;
        add_parameter_value(config->speed, parameterRegisters, TORQUE_KI_REG, 0);

        add_parameter_value(config->torque, parameterRegisters, TORQUE_KP_REG, -200
  00);
        add_parameter_value(config->torque, parameterRegisters, TORQUE_KI_REG, -500
  0);
```

```
    }
}
```

4.4.4 Member Function Documentation

4.4.4.1 defaultInit()

```
void FocDevice::defaultInit ( )
```

Perform default initialization.

This does not start the motor.

Definition at line 314 of file FocDevice.cpp.

```
{
    write_parameter(CONTROL_REG, 0);
    if (config) {
        writeParameterValues(config->init);
    }
}
```

4.4.4.2 readCaptureSource()

```
unsigned int FocDevice::readCaptureSource ( ) const
```

Read the capture source index.

Returns

Capture source index.

Definition at line 422 of file FocDevice.cpp.

```
{
    const uint32_t  control2 = read_parameter(CONTROL2_REG);
    return control2 & 7u;
}
```

4.4.4.3 readDecimate()

```
unsigned int FocDevice::readDecimate ( ) const
```

Read decimation factor.

Returns

The current decimation factor.

Definition at line 477 of file FocDevice.cpp.

```
{
    const uint32_t  control2 = read_parameter(CONTROL2_REG);
    return (control2 & CONTROL2_BITMASK_DECIMATION) >> CONTROL2_BIT_DECIMATION;
}
```

4.4.4.4 readErrorLimit()

```
unsigned int FocDevice::readErrorLimit ( )
```

Read the test error limit.

Returns

Error limit for the speed monitor.

Definition at line 463 of file FocDevice.cpp.

```
{
    const uint32_t  m = read_parameter(CONTROL2_REG);
    return (m & CONTROL2_BV_ERROR_LIMIT) >> CONTROL2_BIT_ERROR_LIMIT;
}
```

4.4.4.5 readLeds()

```
unsigned int FocDevice::readLeds ( )
```

Read LED-s state.

Returns

Bitfield of the leds `LD0` ... `LD3` on the ARTY Z7 platform.

Definition at line 429 of file FocDevice.cpp.

```
{
    return _registers->read32(4);
}
```

4.4.4.6 readParameter()

```
uint32_t FocDevice::readParameter (
              const unsigned int parameterIndex )
```

Read parameter register.

Parameters

parameterIndex	Index of the parameter register to be read from.

Returns

Value of the parameter register.

Definition at line 240 of file FocDevice.cpp.

```
{
    CHECK_PARAMETER_INDEX(argumentIndex);
    const RegisterAccess*   ra = &parameterRegisters[argumentIndex];
    const unsigned int      index = ra->index;

    const uint32_t r = _registers->read32(_parameter_registers_offset + index);
    if (argumentIndex == RPM_SP_REG && !_sdsoc_info) {
        return negative_of_uint32(r);
    }
    else {
        return (r >> ra->shift) & ra->mask;
    }
}
```

4.4.4.7 readParameterString()

```
void FocDevice::readParameterString (
            std::string & buffer,
            const unsigned int parameterIndex )
```

String representation of the parameter register in the following format: NAME CONVERTED_VALUE REGISTER_VALUE.

Parameters

buffer	Buffer to store the the string to.
parameterIndex	Index of the parameter register to be formatted.

Definition at line 256 of file FocDevice.cpp.

```
{
    CHECK_PARAMETER_INDEX(argumentIndex);
    const RegisterAccess*   ra = &parameterRegisters[argumentIndex];
    const uint32_t          u_reg_0 = read_parameter(ra->index);
    uint32_t                u_reg = (u_reg_0 >> ra->shift) & ra->mask;

    if (argumentIndex == RPM_SP_REG && !_sdsoc_info) {
        u_reg = negative_of_uint32(u_reg);
    }

    switch (ra->registerType) {
    case RegisterType::UINT32:
        ssprintf(buffer, "%s %u 0x%X", ra->name, (unsigned int)u_reg, u_reg);
        break;
    case RegisterType::INT32:
        ssprintf(buffer, "%s %d 0x%X", ra->name, (int)u_reg, u_reg);
        break;
    default:
        buffer = "Error: Internal #1";
    }
}
```

4.4.4.8 readSpreadSpectrum()

```
bool FocDevice::readSpreadSpectrum ( )
```

Read the spread spectrum flag.

Returns

true if the spread spectrum is enabled, false otherwise.

Definition at line 498 of file FocDevice.cpp.

```
{
    if (designName == NAME_HLS) {
        const uint32_t  m = read_parameter(CONTROL2_REG);
        return (m & CONTROL2_BV_SPREAD_SPECTRUM) != 0u;
    }
    else {
        return false;
    }
}
```

4.4.4.9 readStatus()

```
uint32_t FocDevice::readStatus (
              const unsigned int statusIndex )
```

Read status register.

Parameters

statusIndex	Index of the status register to be read.

Returns

Value of the status register.

Definition at line 288 of file FocDevice.cpp.

```
{
    CHECK_STATUS_INDEX(statusIndex);
    return _registers->read32(_status_registers_offset + statusIndex);
}
```

4.4.4.10 readStatusString()

```
void FocDevice::readStatusString (
             std::string & buffer,
             const unsigned int statusIndex )
```

String representation of the status register in the following format: NAME VALUE VALUE.

Parameters

buffer	Buffer to store the string to.
statusIndex	Index of the status register to be formatted.

Definition at line 295 of file FocDevice.cpp.

```
{
    CHECK_STATUS_INDEX(statusIndex);
    const RegisterAccess*       ra = &statusRegisters[statusIndex];
    const uint32_t              u_reg = _registers->read32(_status_registers_offset + statusIndex);

    switch (ra->registerType) {
    case RegisterType::UINT32:
        ssprintf(buffer, "%s %u 0x%X", ra->name, u_reg, u_reg);
        break;
    case RegisterType::INT32:
        ssprintf(buffer, "%s %d 0x%X", ra->name, static_cast<int32_t>(u_reg), u_reg);
        break;
    default:
        buffer = "Error: Internal #2";
    }
}
```

4.4.4.11 startMotor()

```
void FocDevice::startMotor (
             const unsigned int mode,
             smart::hw::AxiDataCapture * capture )
```

Start the motor in the given mode.

Parameters

mode	Mode to start the motor in. See the control register in the user manual for the FOC SDSoC project for the applicable values.
capture	Data capture device used for capturing the

Definition at line 325 of file FocDevice.cpp.

```
{
    const uint32_t  old_mode = readParameter(PSEUDO_PARAMETER::MODE);
    if (old_mode == newMode) {
        // Already started, do nothing.
        return;
    }
    if (newMode != MODE_STOPPED) {
        // Stopping the motor resets various internal variables in the FOC.
        writeParameter(PSEUDO_PARAMETER::MODE, MODE_STOPPED);
        if (config) {
            if (newMode == MODE_SPEED
                || newMode == MODE_SPEED_WITHOUT_TORQUE) {
                writeParameterValues(config->speed);
            }
            else if (newMode == MODE_TORQUE_WITHOUT_SPEED) {
                writeParameterValues(config->torque);
            }
        }
        if (old_mode == MODE_STOPPED) {
            const unsigned int  old_fixed_period = readParameter(PSEUDO_PARAMETER::FIXED_PERIOD);
            const unsigned int  fixed_period = std::max<unsigned int>(old_fixed_period + 1u, 200u);
            const float         clocks_per_rev = static_cast<float>(fixed_period) * static_cast<float>(CPR * CPR);
            const unsigned int  ms_to_sleep = static_cast<unsigned int>(2.0 * (1000.0 / FOC_CLOCK_HZ) * clocks_per_rev);

            write_parameter(ANGLE_SH_REG, 0u);
            msleep(100);
            // The forced rotation mode ensures that the encoder index is reset at least once.
            writeParameter(PSEUDO_PARAMETER::MODE, MODE_MANUAL_TORQUE_FLUX_FIXED_SPEED);
            writeParameter(PSEUDO_PARAMETER::FIXED_PERIOD, fixed_period);

            const unsigned int  old_capture_source = readCaptureSource();
            const unsigned int  old_decimate = readDecimate();
            while (capture->isCaptureInProgress()) {
                msleep(10);
            }

            writeDecimate(0);
            writeCaptureSource(DATASOURCE_V_A_B_C);

            const uint64_t              t_end = time_us() + ms_to_sleep * 1000ull;
            volatile int16_t*           buffer16 = reinterpret_cast<volatile int16_t*>(capture->buffer->data());
            // Count of Va samples.
            unsigned int                count_va = 0u;
            // Last Va sample.
            int16_t                     last_va = 0;
            // Zero crossings.
            std::vector<unsigned int>   zero_cross;
            constexpr unsigned int      NSAMPLES = 32u;
            do {
                const unsigned int  current_angle = _registers->read32(_status_registers_offset + ANGLE_REG);
                capture->startCapture(NSAMPLES * sizeof(uint64_t));
                do {
                    msleep(1);
                } while (capture->isCaptureInProgress());

                const int16_t       this_va = *buffer16;
                if (count_va>0 && last_va <= 0 && this_va>0) {
                    zero_cross.push_back(current_angle);
                }

                ++count_va;
                last_va = this_va;
            } while (time_us() < t_end);


            writeCaptureSource(old_capture_source);
            writeDecimate(old_decimate);
            writeParameter(PSEUDO_PARAMETER::FIXED_PERIOD, old_fixed_period);

            if (zero_cross.size()>0u) {
                // The offset of 150 works for the EDDP Kit.
                constexpr unsigned int  offset = (3*CPR)/(10*PPR) - 15;
                const unsigned int      new_shift = (zero_cross.back() + offset) % CPR;
                write_parameter(ANGLE_SH_REG, new_shift);
            }
            else {
                // TODO: How to report the failure?
            }
        }
    }
    writeParameter(PSEUDO_PARAMETER::MODE, newMode);          // Run motor in speed loop
}
```

4.4.4.12 writeCaptureSource()

```
void FocDevice::writeCaptureSource (
             const unsigned int sourceIndex )
```

Set new capture source.

Parameters

sourceIndex	New capture source index.

Definition at line 416 of file FocDevice.cpp.

```
417 {
418     _registers->write32Masked(_parameter_registers_offset + CONTROL2_REG, 0x7, sourceIndex);
419 }
```

4.4.4.13 writeDecimate()

```
void FocDevice::writeDecimate (
             const unsigned int decimationFactor )
```

Write decimation factor (number of samples to skip for every sample captured).

Parameters

decimationFactor	New decimation factor.

Definition at line 470 of file FocDevice.cpp.

```
471 {
472     const unsigned int  df = std::min(CONTROL2_MAX_DECIMATION, decimationFactor);
473     _registers->write32Masked(_parameter_registers_offset + CONTROL2_REG, CONTROL2_BITMASK_DECIMATION, df
      << CONTROL2_BIT_DECIMATION);
474 }
```

4.4.4.14 writeErrorLimit()

```
void FocDevice::writeErrorLimit (
             const unsigned int error_limit )
```

Write error limit.

Parameters

error_limit	New error limit for the speed monitor.

Definition at line 456 of file FocDevice.cpp.

```
{
    const uint32_t  m = read_parameter(CONTROL2_REG);
    write_parameter(CONTROL2_REG, (m & ~CONTROL2_BV_ERROR_LIMIT) | ((error_limit <<
CONTROL2_BIT_ERROR_LIMIT) & CONTROL2_BV_ERROR_LIMIT));
}
```

4.4.4.15 writeLeds()

```
void FocDevice::writeLeds (
              const uint32_t leds )
```

Write led state.

At the moment only 1 led is supported.

Parameters

leds	0 to turn the led `LD0` on the ARTY Z7 platform off, 1 to turn it on.

Definition at line 444 of file FocDevice.cpp.

```
{
    const uint32_t  m = read_parameter(CONTROL2_REG);
    if (leds == 0) {
        write_parameter(CONTROL2_REG, m | CONTROL2_BV_LED);
    }
    else {
        write_parameter(CONTROL2_REG, m & ~CONTROL2_BV_LED);
    }
}
```

4.4.4.16 writeParameter()

```
void FocDevice::writeParameter (
              const unsigned int parameterIndex,
              const uint32_t parameterValue )
```

Write parameter register.

Parameters

parameterIndex	Index of the parameter register to be written to.
parameterValue	Value of the parameter to be written.

Definition at line 217 of file FocDevice.cpp.

```
{
    CHECK_PARAMETER_INDEX(argumentIndex);
    const RegisterAccess*   ra = &parameterRegisters[argumentIndex];
    const unsigned int      index = ra->index;

    if (argumentIndex == RPM_SP_REG && !_sdsoc_info) {
        write_parameter(index, negative_of_uint32(argumentValue));
    }
    else {
        const uint32_t          shift = ra->shift;
        const uint32_t          mask = ra->mask;
        if (shift==0 && mask==UINT32_MAX) {
            // just reading registers can be expensive, too.
            write_parameter(index, argumentValue);
        }
        else {
            _registers->write32Masked(_parameter_registers_offset + index, mask << shift, argumentValue << shift);
        }
    }
}
```

4.4.4.17 writeSpreadSpectrum()

```
void FocDevice::writeSpreadSpectrum (
            const bool enableSpreadSpectrum )
```

Write the spread spectrum flag.

Parameters

enableSpreadSpectrum	True if spread spectrum is to be enabled, false otherwise.

Definition at line 484 of file FocDevice.cpp.

```
{
    if (designName == NAME_HLS) {
        const uint32_t  m = read_parameter(CONTROL2_REG);
        if (enableSpreadSpectrum) {
            write_parameter(CONTROL2_REG, m | CONTROL2_BV_SPREAD_SPECTRUM);
        }
        else {
            write_parameter(CONTROL2_REG, m & ~CONTROL2_BV_SPREAD_SPECTRUM);
        }
    }
}
```

4.4.5 Member Data Documentation

4.4.5.1 designName

```
const char* FocDevice::designName
```

Name of the HW design the software is running on.

This is detected automatically. One of NAME_SDSOC or NAME_HLS.

Definition at line 99 of file FocDevice.h.

4.4.5.2 PSEUDO_PARAMETER_OFFSET

```
constexpr unsigned int FocDevice::PSEUDO_PARAMETER_OFFSET = 16u  [static]
```

Offset to the pseudo registers.

Important: this should match ARGS_SIZE in foc.h

Definition at line 61 of file FocDevice.h.

The documentation for this class was generated from the following files:

- src/FocDevice.h
- src/FocDevice.cpp

4.5 FocServer Class Reference

FOC server implementing the *Network API* and a web server, which permits control and monitor of the FOC system from the Web UI.

```
#include <FocServer.h>
```

Classes

- struct BinaryHeader

  Layout of the binary header.

Public Member Functions

- FocServer (std::shared_ptr< FocConfiguration > config)

  Create new FOC server object.
- ~FocServer ()

  Destruct server object.
- void run ()

  Run the server until either stopped by a signal or by closing the underlying event loop.
- void setTestMode (const bool pTestMode)

  Set or reset the test mode flag.
- void setWwwDirectory (const std::string &newWebDirectory)

  Set the new document root directory for the web server.
- const std::string & getWwwDirectory () const

  Get the docuemnt root directory of the web server.
- FocDevice ∗ device ()

  Access to the underlying FOC device.
- smart::hw::AxiDataCapture ∗ deviceCapture ()

  Access to the underlying data capture device.

Static Public Attributes

- static constexpr unsigned int BINARY_HEADER_SIZE = 32u

  Size of the header, in bytes.

4.5.1 Detailed Description

FOC server implementing the *Network API* and a web server, which permits control and monitor of the FOC system from the Web UI.

Example code:

```
FocServer   server;

server.run();
```

Definition at line 51 of file FocServer.h.

4.5.2 Member Function Documentation

4.5.2.1 device()

```
FocDevice * FocServer::device ( )
```

Access to the underlying FOC device.

Returns

Definition at line 215 of file FocServer.cpp.

```
{
    return &_device;
}
```

4.5.2.2 setTestMode()

```
void FocServer::setTestMode (
             const bool pTestMode )
```

Set or reset the test mode flag.

Parameters

pTestMode	New test mode flag.

Definition at line 196 of file FocServer.cpp.

```
{
    _test_mode = pTestMode;
    _configuration_reply.clear();
}
```

4.5.2.3 setWwwDirectory()

```
void FocServer::setWwwDirectory (
             const std::string & newWebDirectory )
```

Set the new document root directory for the web server.

Parameters

newWebDirectory	New document root directory to serve files from.

Definition at line 203 of file FocServer.cpp.

```
{
    _www_directory = newWwwDirectory;
}
```

The documentation for this class was generated from the following files:

- src/FocServer.h
- src/FocServer.cpp

4.6 FocConfiguration::ParameterValue Struct Reference

Value of a parameter in the configuration file.

```
#include <FocConfiguration.h>
```

Public Attributes

- std::string name

  Name of the register.
- int index

  Index of the parameter, <0 when unknown.
- uint32_t value

  Value of the parameter.

4.6.1 Detailed Description

Value of a parameter in the configuration file.

Definition at line 37 of file FocConfiguration.h.

4.6.2 Member Data Documentation

4.6.2.1 index

```
int FocConfiguration::ParameterValue::index
```

Index of the parameter, <0 when unknown.

See also INDEX_NOT_KNOWN_YET and INDEX_INVALID_NAME.

Definition at line 42 of file FocConfiguration.h.

The documentation for this struct was generated from the following file:

- src/FocConfiguration.h

4.7 FocDevice::RegisterAccess Struct Reference

Description of access to a register in a register bank.

```
#include <FocDevice.h>
```

Public Attributes

- const char ∗ name

  Name of the register.
- const unsigned int index

  Register index in the register bank (parameter or status).
- const RegisterType registerType

  Type of the register.
- const int shift

  Bit shift, if any.
- const uint32_t mask

  Mask of the value in the original position.

4.7.1 Detailed Description

Description of access to a register in a register bank.

Definition at line 84 of file FocDevice.h.

The documentation for this struct was generated from the following file:

- src/FocDevice.h

4.8 WebsocketBuffer Class Reference

Write buffer, consisting of a queue of the messages to be written and a write buffer for the libwebsockets.

```
#include <WebsocketBuffer.h>
```

Public Member Functions

- WebsocketBuffer (struct lws ∗wsi)

  Create new write buffer.
- void writeMessage (const std::string &msg)

  Write a message to the write queue.
- void writeBinary (const void ∗message1, unsigned int size1,...)

  Write a binary message to the write queue.
- int onWriteable ()

  Call this from the libwebsockets callback.

4.8.1 Detailed Description

Write buffer, consisting of a queue of the messages to be written and a write buffer for the libwebsockets.

This simplifies handling of pre- and postpadding as required by libweboscktets.

Important: A WebsocketBuffer is safe to use from one thread at a time only.

Usage: Call writeMessage() any number of times. In the libwebsockets callback, call onWriteable() as needed.

Definition at line 41 of file WebsocketBuffer.h.

4.8.2 Constructor & Destructor Documentation

4.8.2.1 WebsocketBuffer()

```
WebsocketBuffer::WebsocketBuffer (
              struct lws * wsi )
```

Create new write buffer.

Parameters

wsi	Pointer to the libwebsockets object.

Definition at line 57 of file WebsocketBuffer.cpp.

```
:
 _write_buffer((PRE_PADDING + FRAGMENT_SIZE + POST_PADDING) / sizeof(_write_buffer[0])),
 _wsi(wsi),
 _max_queue_size(0),
 _max_write_size(0),
 _was_write_error(false)
{
}
```

4.8.3 Member Function Documentation

4.8.3.1 onWriteable()

```
int WebsocketBuffer::onWriteable ( )
```

Call this from the libwebsockets callback.

This will flush the write queue to the extent possible and schedule new callback if there was some data remaining in the queue.

Definition at line 158 of file WebsocketBuffer.cpp.

```
{
    // NB! Fragments:
    //
    // The write_mode should be set as below:
    // int write_mode;
    // write_mode = LWS_WRITE_BINARY; // single frame, no fragmentation
    // write_mode = LWS_WRITE_BINARY | LWS_WRITE_NO_FIN; // first fragment
    // write_mode = LWS_WRITE_CONTINUATION | LWS_WRITE_NO_FIN; // all middle fragments
    // write_mode = LWS_WRITE_CONTINUATION; // last fragment
    //
    // More details can be found in the fragmentation section of the WebSocket RFC: https://tools.ietf.org/html/rfc6455#section-5.4
    //
    // Source: http://stackoverflow.com/questions/33916549/libwebsocket-send-big-messages-with-limited-payload
    bool            stop_sending = false;
    unsigned char*  write_buffer = reinterpret_cast<unsigned char*>(&_write_buffer[PRE_PADDING / sizeof(_write_buffer[0])]);

    while (!stop_sending && !_write_queue.empty()) {
        WriteRecord&                msg = _write_queue.front();
        const unsigned int          msg_size = msg.buffer.size();

        do {
            unsigned int                todo;
            int                         write_protocol;

            if (msg_size <= FRAGMENT_SIZE) {
                todo = msg_size;
                write_protocol = msg.type;
            }
            else {
                // Fragmented write.
                if (msg.bytesWritten == 0u) {
                    // First fragment.
                    todo = FRAGMENT_SIZE;
                    write_protocol = msg.type | LWS_WRITE_NO_FIN;
                }
                else {
                    const unsigned int  real_todo = msg_size - msg.bytesWritten;
                    if (real_todo > FRAGMENT_SIZE) {
                        // Middle fragments.
                        todo = FRAGMENT_SIZE;
                        write_protocol = LWS_WRITE_CONTINUATION | LWS_WRITE_NO_FIN;
                    }
                    else {
                        todo = real_todo;
                        write_protocol = LWS_WRITE_CONTINUATION;
                    }
                }
            }

            // sorry, have to memcpy. Memcpy is cheap, guys :)
            memcpy(write_buffer, &msg.buffer[0] + msg.bytesWritten, todo);
            const auto r = lws_write(_wsi, write_buffer, todo, (lws_write_protocol)write_protocol);
            if (static_cast<unsigned int>(r) == todo) {
                if (todo > _max_write_size)
                {
                    _max_write_size = todo;
                }
                _was_write_error = false;
                msg.bytesWritten += todo;
            }
            else {
                if (r > 0) {
                    msg.bytesWritten += r;
                    _was_write_error = false;
                    break;
                }
                else {
                    if (!_was_write_error) {
                        lwsl_err("Write error: %d.\n", r);
                    }
                    _was_write_error = true;
                }
                stop_sending = true;
                break;
            }
            if (lws_partial_buffered(_wsi)) {
                stop_sending = true;
                break;
            }
            if (lws_send_pipe_choked(_wsi)) {
                stop_sending = true;
                break;
            }
        } while (!stop_sending && msg.bytesWritten!=msg_size);

        if (msg.bytesWritten == msg_size) {
            _write_queue.pop_front();
        }
    }

    if (!_write_queue.empty()) {
        lws_callback_on_writable(_wsi);
    }
    return _was_write_error ? -1 : 0;
}
```

4.8.3.2 writeBinary()

```
void WebsocketBuffer::writeBinary (
              const void * message1,
              unsigned int size1,
               ... )
```

Write a binary message to the write queue.

It will schedule a callback when the queue was not empty before the call.

Parameters

message1	First message to be written.
size1	Size of the first message to be written.

Definition at line 95 of file WebsocketBuffer.cpp.

```
{
    unsigned int    queue_count = 0;
    unsigned int    queue_bytes = 0;
    unsigned int    total_size = size1;
    const void*     message2;
    unsigned int    size2;
    unsigned int    so_far = size1;
    va_list         ap;

    // Check the queue.
    if (!_checkQueue(queue_count, queue_bytes)) {
        return;
    }
    const bool was_empty = queue_count==0u;

    // Count the total number of bytes.
    va_start(ap, size1);
    for (;;) {
        message2 = va_arg(ap, const void*);
        if (message2 == nullptr) {
            break;
        }
        size2 = va_arg(ap, unsigned int);
        total_size += size2;
    }
    va_end(ap);

    // Create new write record.
    _write_queue.emplace_back();
    WriteRecord& packet = _write_queue.back();
    packet.type = LWS_WRITE_BINARY;
    packet.buffer.resize(total_size);

    // Copy stuff over.
    memcpy(&packet.buffer[0], message1, size1);
    va_start(ap, size1);
    for (;;) {
        message2 = va_arg(ap, const void*);
        if (message2 == nullptr) {
            break;
        }
        size2 = va_arg(ap, unsigned int);
        memcpy(&packet.buffer[so_far], message2, size2);
        so_far += size2;
    }
    va_end(ap);

    packet.bytesWritten = 0;

    // Statistics.
    const unsigned int  qsize = _write_queue.size();
    if (qsize > _max_queue_size) {
        _max_queue_size = qsize;
    }

    // To start writing again, mark us as writable.
    if (was_empty) {
        lws_callback_on_writable(_wsi);
    }
}
```

4.8.3.3 writeMessage()

```
void WebsocketBuffer::writeMessage (
              const std::string & msg )
```

Write a message to the write queue.

It will schedule a callback when the queue was not empty before the call.

Parameters

msg	Message to be written.

Definition at line 68 of file WebsocketBuffer.cpp.

```
{
    unsigned int    queue_count = 0;
    unsigned int    queue_bytes = 0;
    if (!_checkQueue(queue_count, queue_bytes)) {
        return;
    }
    const bool was_empty = queue_count==0u;

    _write_queue.emplace_back();
    WriteRecord& packet = _write_queue.back();
    packet.type = LWS_WRITE_TEXT;
    packet.buffer.resize(msg.size());
    memcpy(&packet.buffer[0], msg.c_str(), msg.size());
    packet.bytesWritten = 0;

    const unsigned int  qsize = _write_queue.size();
    if (qsize > _max_queue_size) {
        _max_queue_size = qsize;
    }

    if (was_empty) {
        lws_callback_on_writable(_wsi);
    }
}
```

The documentation for this class was generated from the following files:

- src/WebsocketBuffer.h
- src/WebsocketBuffer.cpp

Chapter 5

File Documentation

5.1 main.cpp File Reference

Implementation of the main function of the focserver.

```
#include "src/focserver_main.h"
```

Functions

- int main (int argc, char ∗argv[ ])
  Entry point to the program focserver.

5.1.1 Detailed Description

Implementation of the main function of the focserver.

Webserver control program for the Field-Oriented Control demo.

Version

1.0

Date

2017

Copyright

SPDX: BSD-3-Clause 2016-2017 Trenz Electronic GmbH

5.1.2 Function Documentation

5.1.2.1 main()

```
int main (
            int argc,
            char * argv[] )
```

Entry point to the program focserver.

This just calls Main function focserver_main. See section Introduction for the description.

Definition at line 15 of file main.cpp.

```
16 {
17 
18     const int   r = focserver_main(argc, argv);
19     return r;
20 }
```

5.2 src/DeviceTreeDevice.h File Reference

Implementation of the class DeviceTreeDevice.

```
#include <memory>
#include <string>
#include <map>
#include <stdint.h>
```

Classes

- class DeviceTreeDevice

  Fetch information from the Linux Device Tree.

5.2.1 Detailed Description

Implementation of the class DeviceTreeDevice.

Interface of the class DeviceTreeDevice.

Version

1.0

Date

2017

Copyright

SPDX: BSD-3-Clause 2016-2017 Trenz Electronic GmbH

5.3 src/FocConfiguration.cpp File Reference

Implementation of the class FocConfiguration.

```
#include <stdexcept>
#include <string>
#include <vector>
#include <stdio.h>
#include <json/reader.h>
#include <json/value.h>
#include <smart/File.h>
#include <smart/string.h>
#include "foc.h"
#include "FocConfiguration.h"
```

5.3.1 Detailed Description

Implementation of the class FocConfiguration.

Version

1.0

Date

2017

Copyright

SPDX: BSD-3-Clause 2016-2017 Trenz Electronic GmbH

5.4 src/FocConfiguration.h File Reference

Interface of the class FocConfiguration.

```
#include <memory>
#include <string>
#include <vector>
#include <stdint.h>
```

Classes

- class FocConfiguration

  Configuration of the FOC server.
- struct FocConfiguration::ParameterValue

  Value of a parameter in the configuration file.

5.4.1 Detailed Description

Interface of the class FocConfiguration.

Version

1.0

Date

2017

Copyright

SPDX: BSD-3-Clause 2016-2017 Trenz Electronic GmbH

5.5 src/FocDevice.cpp File Reference

Implementation of the class FocDevice.

```
#include <stdexcept>
#include <string.h>
#include <smart/string.h>
#include <smart/time.h>
#include "FocDevice.h"
#include "foc.h"
```

Macros

- #define write_parameter(index, value) _registers->write32(_parameter_registers_offset + (index), (value))

  Write a parameter register.
- #define read_parameter(index) _registers->read32(_parameter_registers_offset + (index))

  Read a parameter register.
- #define CHECK_PARAMETER_INDEX(parameter_index)

  Check the parameter register index and throw an exception when it is not in the permitted range.
- #define CHECK_STATUS_INDEX(status_index)

  Check the status register index and throw an exception when it is not in the permitted range.
- #define **GET_INT16**(data64, int16index) ((int16_t)((data64 >> (int16index*16)) & 0xFFFF))

5.5.1 Detailed Description

Implementation of the class FocDevice.

Version

1.0

Date

2017

Copyright

SPDX: BSD-3-Clause 2016-2017 Trenz Electronic GmbH

5.5.2 Macro Definition Documentation

5.5.2.1 CHECK_PARAMETER_INDEX

```
#define CHECK_PARAMETER_INDEX(
                parameter_index )
```

Value:

```
do {                                            \
        if ((parameter_index) >= parameterCount) {
      \
            throw std::runtime_error(ssprintf("FocDevice: parameter index %u outside range 0 ... %u", (
      parameter_index), parameterCount-1u));    \
        }
      \
} while (0)
```

Check the parameter register index and throw an exception when it is not in the permitted range.

Definition at line 203 of file FocDevice.cpp.

5.5.2.2 CHECK_STATUS_INDEX

```
#define CHECK_STATUS_INDEX(
                status_index )
```

Value:

```
do {                                            \
        if ((status_index) >= statusCount) {
      \
            throw std::runtime_error(ssprintf("FocDevice: status index %u outside range 0 ... %u", (
      status_index), statusCount-1u));     \
        }
      \
} while (0)
```

Check the status register index and throw an exception when it is not in the permitted range.

Definition at line 280 of file FocDevice.cpp.

5.5.2.3 read_parameter

```
#define read_parameter(
                index ) _registers->read32(_parameter_registers_offset + (index))
```

Read a parameter register.

Ensure index is in the correct range before calling this function.

Parameters

index	Index of the parameter register to be read.

Definition at line 61 of file FocDevice.cpp.

5.5.2.4 write_parameter

```
#define write_parameter(
            index,
            value ) _registers->write32(_parameter_registers_offset + (index), (value))
```

Write a parameter register.

Ensure index is in the correct range before calling this function.

Parameters

index	Index of the parameter register.
value	Value to be written to the parameter register.

Definition at line 57 of file FocDevice.cpp.

5.6 src/FocDevice.h File Reference

Interface of the class FocDevice.

```
#include <limits>
#include <memory>
#include <stdint.h>
#include <smart/hw/AxiDataCapture.h>
#include <smart/MappedFile.h>
#include <smart/UioDevice.h>
#include "DeviceTreeDevice.h"
#include "FocConfiguration.h"
```

Classes

- class FocDevice

  Access to the FOC IP core.
- struct FocDevice::RegisterAccess

  Description of access to a register in a register bank.

5.6.1 Detailed Description

Interface of the class FocDevice.

Version

1.0

Date

2017

Copyright

SPDX: BSD-3-Clause 2016-2017 Trenz Electronic GmbH

5.7 src/FocServer.cpp File Reference

Implementation of the class FocServer.

```
#include <limits>
#include <stdexcept>
#include <string>
#include <stdio.h>
#include <string.h>
#include <errno.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <inttypes.h>
#include <libwebsockets.h>
#include <smart/string.h>
#include <smart/time.h>
#include <json/writer.h>
#include <json/value.h>
#include "FocServer.h"
#include "DeviceTreeDevice.h"
#include "Version.h"
```

Macros

- #define DEFAULT_WWW_DIRECTORY "/usr/share/focserver"

  Default document root directory for the web server.
- #define NAME_LEDS_STATUS_REG "LEDs"

  Name of the led status register.
- #define NAME_SPREAD_SPECTRUM_REG "SpreadSpectrum"

  name of the fictive spread spectrum register.
- #define COMMAND_CAPTURE "Capture"

  Name of the capture command.
- #define COMMAND_RESET_ERROR "ResetError"

  Name of the reset error command.
- #define COMMAND_ERROR_LIMIT "ErrorLimit"

  Name of the error limit parameter register.
- #define COMMAND_CONFIGURATION "Configuration"

  Command to query/set configuration.

Enumerations

- enum server_protocols { **PROTOCOL_HTTP** = 0, **PROTOCOL_FOC**, **PROTOCOL_COUNT** }

  List of the protocols supported.

5.7.1 Detailed Description

Implementation of the class FocServer.

Version

1.0

Date

2017

Copyright

SPDX: BSD-3-Clause 2016-2017 Trenz Electronic GmbH

5.8 src/FocServer.h File Reference

Interface of the class FocServer.

```
#include <list>
#include <memory>
#include <string>
#include <stdint.h>
#include <libwebsockets.h>
#include <uv.h>
#include <smart/hw/AxiDataCapture.h>
#include "FocConfiguration.h"
#include "FocDevice.h"
#include "WebsocketBuffer.h"
```

Classes

- class FocServer

  FOC server implementing the Network API and a web server, which permits control and monitor of the FOC system from the Web UI.
- struct FocServer::BinaryHeader

  Layout of the binary header.

5.8.1 Detailed Description

Interface of the class FocServer.

Version

1.0

Date

2017

Copyright

SPDX: BSD-3-Clause 2016-2017 Trenz Electronic GmbH

5.9 src/focserver_main.cpp File Reference

Implementation of the function focserver_main.

```
#include "focserver_main.h"
#include <memory>
#include <stdexcept>
#include <string>
#include <getopt.h>
#include <inttypes.h>
#include <stdlib.h>
#include <string.h>
#include <syslog.h>
#include <sys/time.h>
#include <unistd.h>
#include <uv.h>
#include <libwebsockets.h>
#include <smart/File.h>
#include <smart/string.h>
#include <smart/time.h>
#include <smart/WavFormat.h>
#include <smart/hw/AxiDataCapture.h>
#include "FocConfiguration.h"
#include "DeviceTreeDevice.h"
#include "FocDevice.h"
#include "FocServer.h"
#include "Version.h"
#include "foc.h"
```

Functions

- int focserver_main (int argc, char *argv[ ])

  Main function of the focserver, which implements the Network API and a web server.

5.9.1 Detailed Description

Implementation of the function focserver_main.

Version

1.0

Date

2017

Copyright

SPDX: BSD-3-Clause 2016-2017 Trenz Electronic GmbH

5.9.2 Function Documentation

5.9.2.1 focserver_main()

```
int focserver_main (
            int argc,
            char * argv[] )
```

Main function of the focserver, which implements the Network API and a web server.

See section Introduction for the description. Result of the write or capture operation.

Definition at line 234 of file focserver_main.cpp.

```
{
    int debug_level = 7;
    int n = 0;
    int syslog_options = LOG_PID | LOG_PERROR;
    bool daemonize = false;
    bool start_motor = false;
    std::unique_ptr<int>                    start_motor_speed;
    std::unique_ptr<FocDevice>              write_device;
    std::unique_ptr<hw::AxiDataCapture>     capture_device;
    std::shared_ptr<FocConfiguration>       configuration;
    /// Result of the write or capture operation.
    int                                     op_result = 0;
    bool                                    test_mode = false;
    const char*                             www_directory = nullptr;

    try {
        while (n >= 0) {
            n = getopt_long(argc, argv, "c:d:f:Dhps::tvw:W:", options, NULL);
            if (n < 0)
                continue;
            switch (n) {
            case 'c':
                op_result = capture(capture_device, optarg);
                if (op_result != 0) {
                    return op_result;
                }
                break;
            case 'D':
                daemonize = true;
                syslog_options &= ~LOG_PERROR;
                break;
            case 'd':
                debug_level = atoi(optarg);
                break;
            case 'f':
                configuration = FocConfiguration::fromFile(optarg);
                if (configuration) {
                    configuration->dump();
                }
                else {
                    lwsl_notice("Error: configuration file %s not found\n", optarg);
                    return 1;
                }
                break;
            case 'h':
                print_usage();
                exit(1);
            case 'p':
                print_registers();
                return 0;
            case 's':
                if (optarg != nullptr) {
                    int x = 0;
                    if (int_of(optarg, x)) {
                        start_motor_speed = std::unique_ptr<int>(new int(x));
                    }
                }
                start_motor = true;
                break;
            case 't':
                test_mode = true;
                break;
            case 'v':
                printf("Version: %s\n", Version::FOCSERVER_DATE);
                return 0;
            case 'w':
                op_result = write_register(write_device, optarg);
                if (op_result != 0) {
                    return op_result;
                }
                break;
            case 'W':
                www_directory = optarg;
                break;
            }
        }

        if (!start_motor && !daemonize && (write_device || capture_device)) {
            return op_result;
        }

        /*
         * normally lock path would be /var/lock/lwsts or similar, to
         * simplify getting started without having to take care about
         * permissions or running as root, set to /tmp/.lwsts-lockc
         */
        if (daemonize && lws_daemonize("/tmp/.lwsts-lock")) {
            fprintf(stderr, "Failed to daemonize\n");
            return 1;
        }

        /* we will only try to log things according to our debug_level */
        setlogmask(LOG_UPTO (LOG_DEBUG));
        openlog("lwsts", syslog_options, LOG_DAEMON);

        /* tell the library what debug level to emit and to send it to syslog */
        lws_set_log_level(debug_level, lwsl_emit_syslog);

        lwsl_notice("FOC webserver.\n");

        if (!configuration && File::exists(FocConfiguration::FILENAME)) {
            configuration = FocConfiguration::fromFile(
  FocConfiguration::FILENAME);
        }
        if (!configuration) {
            lwsl_notice("Configuration file %s not found\n",
  FocConfiguration::FILENAME);
        }
        FocServer                  server(configuration);
        FocDevice*                 dev = server.device();
        smart::hw::AxiDataCapture*  devcap = server.deviceCapture();

        server.setTestMode(test_mode);
        if (www_directory != nullptr) {
            server.setWwwDirectory(www_directory);
        }
        lwsl_notice("focserver version: %s\n", Version::FOCSERVER_DATE);
```

```
        lwsl_notice("FOC design:        %s\n", dev->designName);
        lwsl_notice("FOC IP core base address: 0x%08" PRIxPTR "\n", dev->getBaseAddress());
        lwsl_notice("WWW server directory:     %s\n", server.getWwwDirectory().c_str());
        lwsl_notice("Test mode:  %s\n", test_mode ? "true" : "false");

        if (start_motor) {
            if (start_motor_speed) {
                dev->writeParameter(RPM_SP_REG, (uint32_t)*start_motor_speed);
            }
            lwsl_notice("Starting the motor at speed %d RPM.\n", (int32_t)dev->readParameter(RPM_SP_REG));
            dev->startMotor(MODE_SPEED, devcap);
        }
        server.run();

        lwsl_notice("Exited cleanly\n");
    } catch (const std::exception& ex) {
        printf("Error: %s\n", ex.what());
        return 2;
    }
    return 0;
}
```

5.10 src/focserver_main.h File Reference

Declaration of the function focserver_main.

Functions

- int focserver_main (int argc, char *argv[ ])

  Main function of the focserver, which implements the Network API and a web server.

5.10.1 Detailed Description

Declaration of the function focserver_main.

Version

1.0

Date

2017

Copyright

SPDX: BSD-3-Clause 2016-2017 Trenz Electronic GmbH

5.10.2 Function Documentation

5.10.2.1 focserver_main()

```
int focserver_main (
            int argc,
            char * argv[ ] )
```

Main function of the focserver, which implements the Network API and a web server.

See section Introduction for the description. Result of the write or capture operation.

Definition at line 234 of file focserver_main.cpp.

```
{
    int debug_level = 7;
    int n = 0;
    int syslog_options = LOG_PID | LOG_PERROR;
    bool daemonize = false;
    bool start_motor = false;
    std::unique_ptr<int>                    start_motor_speed;
    std::unique_ptr<FocDevice>              write_device;
    std::unique_ptr<hw::AxiDataCapture>     capture_device;
    std::shared_ptr<FocConfiguration>       configuration;
    /// Result of the write or capture operation.
    int                                     op_result = 0;
    bool                                    test_mode = false;
    const char*                             www_directory = nullptr;

    try {
        while (n >= 0) {
            n = getopt_long(argc, argv, "c:d:f:Dhps::tvw:W:", options, NULL);
            if (n < 0)
                continue;
            switch (n) {
            case 'c':
                op_result = capture(capture_device, optarg);
                if (op_result != 0) {
                    return op_result;
                }
                break;
            case 'D':
                daemonize = true;
                syslog_options &= ~LOG_PERROR;
                break;
            case 'd':
                debug_level = atoi(optarg);
                break;
            case 'f':
                configuration = FocConfiguration::fromFile(optarg);
                if (configuration) {
                    configuration->dump();
                }
                else {
                    lwsl_notice("Error: configuration file %s not found\n", optarg);
                    return 1;
                }
                break;
            case 'h':
                print_usage();
                exit(1);
            case 'p':
                print_registers();
                return 0;
            case 's':
                if (optarg != nullptr) {
                    int x = 0;
                    if (int_of(optarg, x)) {
                        start_motor_speed = std::unique_ptr<int>(new int(x));
                    }
                }
                start_motor = true;
                break;
            case 't':
                test_mode = true;
                break;
            case 'v':
                printf("Version: %s\n", Version::FOCSERVER_DATE);
                return 0;
            case 'w':
                op_result = write_register(write_device, optarg);
                if (op_result != 0) {
                    return op_result;
                }
                break;
            case 'W':
                www_directory = optarg;
                break;
            }
        }

        if (!start_motor && !daemonize && (write_device || capture_device)) {
            return op_result;
        }

        /*
         * normally lock path would be /var/lock/lwsts or similar, to
         * simplify getting started without having to take care about
         * permissions or running as root, set to /tmp/.lwsts-lockc
         */
        if (daemonize && lws_daemonize("/tmp/.lwsts-lock")) {
            fprintf(stderr, "Failed to daemonize\n");
            return 1;
        }

        /* we will only try to log things according to our debug_level */
        setlogmask(LOG_UPTO (LOG_DEBUG));
        openlog("lwsts", syslog_options, LOG_DAEMON);

        /* tell the library what debug level to emit and to send it to syslog */
        lws_set_log_level(debug_level, lwsl_emit_syslog);

        lwsl_notice("FOC webserver.\n");

        if (!configuration && File::exists(FocConfiguration::FILENAME)) {
            configuration = FocConfiguration::fromFile(FocConfiguration::FILENAME);
        }
        if (!configuration) {
            lwsl_notice("Configuration file %s not found\n", FocConfiguration::FILENAME);
        }
        FocServer                  server(configuration);
        FocDevice*                 dev = server.device();
        smart::hw::AxiDataCapture*  devcap = server.deviceCapture();

        server.setTestMode(test_mode);
        if (www_directory != nullptr) {
            server.setWwwDirectory(www_directory);
        }
        lwsl_notice("focserver version: %s\n", Version::FOCSERVER_DATE);
        lwsl_notice("FOC design:        %s\n", dev->designName);
        lwsl_notice("FOC IP core base address: 0x%08" PRIxPTR "\n", dev->getBaseAddress());
        lwsl_notice("WWW server directory:     %s\n", server.getWwwDirectory().c_str());
        lwsl_notice("Test mode:  %s\n", test_mode ? "true" : "false");

        if (start_motor) {
            if (start_motor_speed) {
                dev->writeParameter(RPM_SP_REG, (uint32_t)*start_motor_speed);
            }
            lwsl_notice("Starting the motor at speed %d RPM.\n", (int32_t)dev->readParameter(RPM_SP_REG));
            dev->startMotor(MODE_SPEED, devcap);
        }
        server.run();

        lwsl_notice("Exited cleanly\n");
    } catch (const std::exception& ex) {
        printf("Error: %s\n", ex.what());
        return 2;
    }
    return 0;
}
```

5.11 src/Version.h File Reference

Version information.

Variables

- constexpr const char * Version::FOCSERVER_DATE = "2017-09-27"

Build date of the focserver.

5.11.1 Detailed Description

Version information.

Version

1.0

Date

2017

Copyright

SPDX: BSD-3-Clause 2016-2017 Trenz Electronic GmbH

5.12 src/WebsocketBuffer.cpp File Reference

Implementation of the class WebsocketBuffer.

```
#include <stdarg.h>
#include <string.h>
#include "WebsocketBuffer.h"
```

Macros

- #define ALIGN(x_align) (ALIGNMENT∗(((x_align) + ALIGNMENT - 1u)/ALIGNMENT))

  Align a value.

Typedefs

- typedef uint64_t buffer_element_t

  Buffer element.

Variables

- constexpr unsigned int QUEUE_LENGTH_LIMIT = 500

  Limit of the write queue length.
- constexpr unsigned int QUEUE_BYTES_LIMIT = 10 ∗ 1024 ∗ 1024

  Limit of the total data size in a queue, in bytes.
- constexpr unsigned int ALIGNMENT = (sizeof(buffer_element_t))

  Alignment, in bytes.
- constexpr unsigned int PRE_PADDING = ALIGN(LWS_SEND_BUFFER_PRE_PADDING)

  Size of pre-padding, in bytes, aligned.
- constexpr unsigned int POST_PADDING = ALIGN(LWS_SEND_BUFFER_POST_PADDING)

  Size of post-paddding, in bytes, aligned.
- constexpr unsigned int FRAGMENT_SIZE = 32∗1024

  Size above which messages will be fragmentized.

5.12.1 Detailed Description

Implementation of the class WebsocketBuffer.

Version

1.0

Date

2017

Copyright

SPDX: BSD-3-Clause 2016-2017 Trenz Electronic GmbH

5.13 src/WebsocketBuffer.h File Reference

Interface of the class WebsocketBuffer.

```
#include <stdint.h>
#include <string>
#include <vector>
#include <deque>
#include <libwebsockets.h>
```

Classes

- class WebsocketBuffer

  Write buffer, consisting of a queue of the messages to be written and a write buffer for the libwebsockets.

5.13.1 Detailed Description

Interface of the class WebsocketBuffer.

Version

1.0

Date

2017

Copyright

SPDX: BSD-3-Clause 2016-2017 Trenz Electronic GmbH

Index