Software Design Document

*for*

IO Extender DLL

COPYRIGHT 2014VCortex Ltd ©

This document contains valuable and confidential information proprietary to VCortex Ltd. The recipient of this document shall not disclose its contents, in whole or in part, to any third party without VCortex’ prior written approval.

|  |  |  |
| --- | --- | --- |
| *Version* | *Date* | *Description* |
| 1 | Nov 20, 2014 | Basic version |

Contents

[1. General 3](#_Toc414549990)

[1.1. Scope 3](#_Toc414549991)

[1.2. Overview 3](#_Toc414549992)

[2. Source Files 4](#_Toc414549993)

[3. Detailed Design 4](#_Toc414549994)

# General

## Scope

This document describes the DLL that was developed for the ETHIIC module, which runs on the host computer and connects to the application. The DLL has C interfaces to comply with C or C++ applications.

## Overview

The ETHIIC (Ethernet Integrated InterCom) is a bridge that converts Ethernet communication to I2C communication, to enable SW running on the main computer to control I2C controlled devices.

The ETHIIC is a partial implementation of the IOX (IO Extender module) that includes some more interfaces implementation. The GPIO and IIC are fully implemented in the ETHIIC board, as a derivative of the IOX module.

The ETHIIC can be connected as a single or multiple configuration, controlled by host computer through UDP commands. Therefore, the ETHIIC module bears a unique logical name, unique IP and unique MAC address.

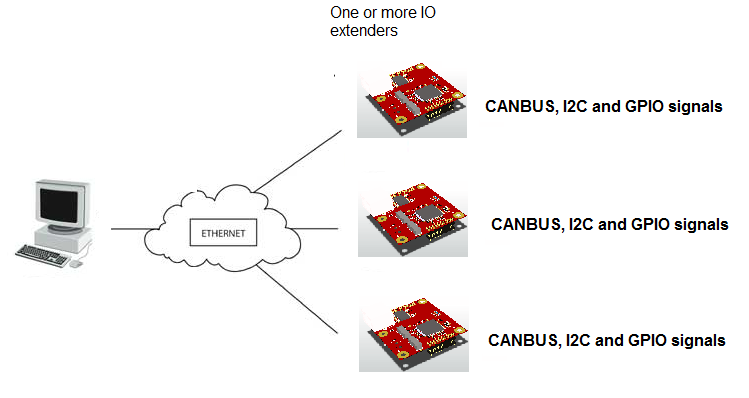


Figure 1: IOX multiple board configuration

# Source Files

Following table describes the source files of the IO\_Extender DLL:

|  |  |
| --- | --- |
| File | Description |
| IO\_Extender.h | Definition of interface functions and types |
| IO\_Extender.cpp | Implementation of interface functions |

# Detailed Design

The IO\_Extender interface routines are described herein.

**IOX\_init():**

This function initializes the UDP/IP resources.

* Initialize logger
* Initialize Winsock
* Create a receiver socket to receive datagrams
* Create a sending socket to send datagrams
* Bind the receiver socket to any address and the specified port (4478)

**IOX\_initdevice():**

This function inits a device using its logical name. The returned handle will be used in sending and receiving data from the device.

* Create a UDP message with message type = INIT, and device logic name.
* Retry 3 times, until an answer received from device
  + Send the message in broadcast mode, on port 4477
  + Set the receiving timeout on 1000 mSec
  + Receive device answer
* If an answer received, put the device on the device list, with its IP address (will be used for later transmissions)

**IOX\_initIIC():**

This function inits an I2C connection on a device.

* Look for the handle in the device list
* Build the UDP message including the requested speed
* Retry 3 times, until an answer received from device
  + Send the message to the device IP, on port 4477
  + Set the receiving timeout on 1000 mSec
  + Receive device answer
* If an answer received, return OK, else return error

**IOX\_writeBytes():**

This function writes bytes into a slave connected to the device. buff holds the bytes to write, and buffLen is the number of bytes to write.

* Look for the handle in the device list
* Build the UDP message including the requested buff and num of bytes
* Retry 3 times, until an answer received from device
  + Send the message to the device IP, on port 4477
  + Set the receiving timeout on 1000 mSec
  + Receive device answer
* If an answer received, return OK, else return error

**IOX\_writeAndReadByte():**

This function writes byte into a slave and then reads one back.

* Look for the handle in the device list
* Build the UDP message including the requested byte to write
* Retry 3 times, until an answer received from device
  + Send the message to the device IP, on port 4477
  + Set the receiving timeout on 1000 mSec
  + Receive device answer
* If an answer received, fetch the returned byte and return OK. Else, return error.

**IOX\_readBytes():**

This function reads bytes from a slave connected to the device. buff will holds the bytes that were read, and buffLen is the number of bytes to read

* Look for the handle in the device list
* Build the UDP message including the requested number of bytes to read
* Retry 3 times, until an answer received from device
  + Send the message to the device IP, on port 4477
  + Set the receiving timeout on 1000 mSec
  + Receive device answer
* If an answer received, fetch the returned bytes into buff and return OK. Else, return error.

**IOX\_getCommStat():**

This function returns some statistics on the communication performed to and from the device.

**IOX\_wakeupIIC():**

Special function for Stratasys device to start IIC bus

* Look for the handle in the device list
* Build the UDP message including the message code WAKEUP\_IIC
* Retry 3 times, until an answer received from device
  + Send the message to the device IP, on port 4477
  + Set the receiving timeout on 1000 mSec
  + Receive device answer
* If an answer received, return OK. Else, return error.

**IOX\_setLog():**

This function sets the log level and output media. The DLL is using log4cplus

<http://sourceforge.net/p/log4cplus/wiki/Home/>