IODA Product Software Library Specification

Project Number: **IODA**
Document Number: CTS-IODA-LIBRARY

Revision 1.04
April 30, 2010

Updated By: Young Teak Lim

Approval Signatures:	☐ Name Firmware Engineer Date:	☐ Name Software Engineer Date:	☐ Name (Job Position) Date:

(page intentionally left blank)

Revision History

Rev.	Date/Initials	Location	Description of Change
1.00	March 31, 2010 - YT	All	-Initial Revision
1.01	April 6, 2010 -YT	All	-Sample Code Added
1.02	April 9, 2010 -YT	All	-memory_read() function description Added -Error Code Added -get_library_version() Added
1.03	April 29, 2010 -YT	All	-get_ad_sampling_ratio() function added -set_ad_sampling_ratio() function added
1.04	April 30, 2010 -YT	All	- API description modified. ;set_pattern_addr () ;set_pattern_memory_info()

1. INTRODUCTION

1.1. Scope

This document describes the IODA Product Software Library features.

1.2. Objective

The purpose of this document is to provide user library interface information to user.

1.3. Overview

The primary objective of this project is to describe the internal feature of library.

1.4. Definition of Terms

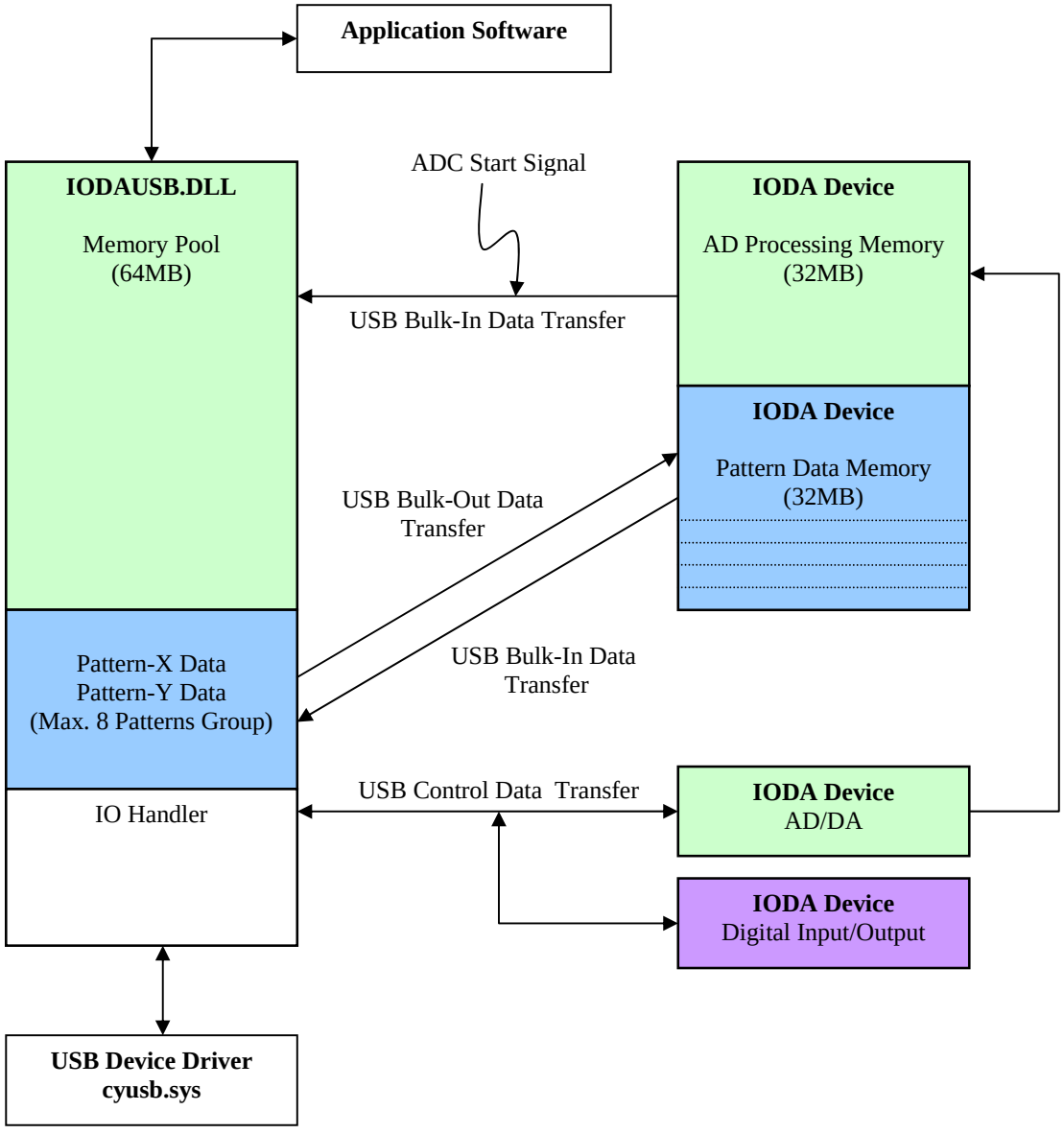
Abbreviations

FRS	Functional Requirement Specification
MRD	Market Requirement Specification
PFSD	Product Feature Specification Document
PISD	Product Interface Specification Document
PSSD	Product Software Specification Document

2. OVERVIEW

2.1. Memory Interface Architecture

Block Diagram



3. IODAUSB.DLL

3.1. Introduction

본 Library 는 IODA USB Device Driver 와 그 장치간의 Interface 를 위한 Library 로서 Visual C Compiler 환경에서 Multi-Byte Code System(MBCS), Shared MFC DLL, Release Type 으로 Build 되었다.
라이브러리 사용을 위해서는 아래의 Resource 가 반드시 필요하며, 버전 별 호환여부는 Release Note 혹은 본 문서의 Revision History 를 참고하여 확인 가능하다.

- IODAUSB.DLL ; Dynamic Link Library
- IODAUSB.lib ; Library for Win32 typed exported, functions position and the prototype.
- IODAUSB.h ; Exported functions list and predefined variable

DLL 내 모든 함수는 Microsoft 의 C/C++ Language 확장 스펙의 __declspec(dllexport)으로 export 되었으며, 함수 구현 및 호출 규약(Function Calling Convention) 은 __stdcall 을 준수하였다.

3.2. Function & Variable Lists

3.2.1. Header File

```
//////////////////////////////////////////////////////////////////////////////////////////////////
//
// IODAUSB.h : IODAUSB.DLL header
//
//                              Copyright (c) 2010
//                      CANTOPS - SEOUL, REPUBLIC OF KOREA
//                                                              www.cantops.biz
//
// This program is the property of the CANTOPS. and it shall not be reproduced, distributed or
// used without permission of an authorized Company official.
// This is an unpublished work subject to Trade Secret and Copyright protection.
//
// DLL Version: V1.1.0.0
// Date: March 31, 2010
//
//////////////////////////////////////////////////////////////////////////////////////////////////
//
#ifdef __cplusplus
extern "C"
{
#endif

#ifndef __IODAUSB_LIB_H__
#define __IODAUSB_LIB_H__

#ifdef IODAUSB_EXPORTS
        #define IODAUSB_API    __declspec(dllexport)
#else
        #define IODAUSB_API    __declspec(dllimport)
#endif


//////////////////////////////////////////////////////////////////////////////////////////////////
//
// type definition
//
typedef unsigned char               UCHAR;
typedef  char                       CHAR;
typedef unsigned short              USHORT;
typedef unsigned short              WORD;
typedef signed   short              SHORT;
typedef unsigned int                UINT;
typedef signed   int                SINT;
typedef unsigned long               ULONG;
typedef signed   long               LONG;

//////////////////////////////////////////////////////////////////////////////////////////////////
//
// USB communication Definition
//
#define IODAUSB_MEMORY_POOL_SIZE            (0x04000000) //64 MB

#define IODAUSB_BULK_IN_BUF_SIZE            1024
#define IODAUSB_BULK_OUT_BUF_SIZE           512

#define IODAUSB_ADDA_STATUS_ADDA_STOP 0
#define IODAUSB_ADDA_STATUS_ADDA_START      1
#define IODAUSB_ADDA_STATUS_BULK_STOP 0
#define IODAUSB_ADDA_STATUS_BULK_BUSY 1
#define IODAUSB_ADDA_STATUS_BUF_EMPTY 0
#define IODAUSB_ADDA_STATUS_BUF_FULL  1

#define IODAUSB_INTERFACE_MODE_BULKIN 0
#define IODAUSB_INTERFACE_MODE_BULKOUT      1

#define IODAUSB_PATTERN_MAX                 8


//////////////////////////////////////////////////////////////////////////////////////////////////
//
// error number definition
//
```

```
enum IoDaUsbErrNo
{
	IODAUSB_NOERR=0,
	IODAUSB_ERR_DEV_INIT,
	IODAUSB_ERR_DEV_CLOSE,
	IODAUSB_ERR_DEV_RESET,
	IODAUSB_ERR_DEV_ABORT,
	IODAUSB_ERR_CONTROL_READ,
	IODAUSB_ERR_CONTROL_WRITE,
	IODAUSB_ERR_BULK_READ,
	IODAUSB_ERR_BULK_WRITE,
	IODAUSB_ERR_IO_READ,
	IODAUSB_ERR_IO_WRITE,
	IODAUSB_ERR_AD_READ,
	IODAUSB_ERR_DA_READ,
	IODAUSB_ERR_DA_WRITE,
	IODAUSB_ERR_INTERFACE_MODE_CHANGE,
	IODAUSB_ERR_READ_FRAME_DATA,
	IODAUSB_ERR_WRITE_FRAME_DATA,
	IODAUSB_ERR_PATTERN_INFO_READ,
	IODAUSB_ERR_PATTERN_INFO_WRITE,
	IODAUSB_ERR_PATTERN_INFO_SELECT,
	IODAUSB_ERR_READ_LINE_DATA,
	IODAUSB_ERR_READ_STACK_DATA_LENGTH,
	IODAUSB_ERR_PATTERN_ADDRESS,
};

////////////////////////////////////////////////////////////////////////////////////////////////////
//
// error string definition
//
char gcIoDaUsbErrorString[][256] =
{
	"",
	"Device is not initialized.",
	"Device is not closed.",
	"Device is not reset.",
	"Device is not abort.",
	"USB Communication error. control register reading.",
	"USB Communication error. control register writing.",
	"USB Communication error. bulk data reading.",
	"USB Communication error. bulk data writing.",
	"USB Communication error. I/O data reading.",
	"USB Communication error. I/O data writing.",
	"IODA AD value read failed.",
	"IODA DA value read failed.",
	"IODA DA value write failed.",
	"USB Communication error. Interface mode change.",
	"IODA frame data read failed.",
	"IODA frame data write failed.",
	"IODA pattern image information read failed.",
	"IODA pattern image information write failed.",
	"IODA pattern image information select failed.",
	"IODA frame data read failed.",
	"IODA stack data read length is too larger than bulk-in size.",
	"IODA pattern address error. It should be the value of 4 times.",
	"",
};

////////////////////////////////////////////////////////////////////////////////////////////////////
//
// USER FUNCTIONS
//
// library and USB communication
IODAUSB_API void ioda_init();
IODAUSB_API void ioda_exit();

IODAUSB_API    bool ioda_usb_open(UCHAR devID);
IODAUSB_API    void ioda_usb_close();
IODAUSB_API    void ioda_usb_reset();
IODAUSB_API    void ioda_usb_abort();

IODAUSB_API    bool get_library_version(ULONG *version);
IODAUSB_API    bool get_fpga_version(UCHAR* version);
IODAUSB_API    bool get_sampling_freq(ULONG* freq);
IODAUSB_API    bool set_sampling_freq(ULONG freq);

// pattern management
IODAUSB_API    bool get_pattern_addr(ULONG* addr);
IODAUSB_API    bool set_pattern_addr(ULONG addr);
IODAUSB_API    bool get_pattern_metrics(USHORT* width, USHORT* height);
```

```
IODAUSB_API    bool set_pattern_metrics(USHORT width, USHORT height);
IODAUSB_API    bool select_pattern(UCHAR id);
IODAUSB_API    bool get_selected_pattern(UCHAR* id);
IODAUSB_API    bool write_frame_data(USHORT width, USHORT height, void *pSrcImage);
IODAUSB_API    bool get_pattern_memory_info(UCHAR id, ULONG* addr,
                                            USHORT* width, USHORT* height);
IODAUSB_API    bool set_pattern_memory_info(UCHAR id, ULONG addr, USHORT width, USHORT height);
IODAUSB_API    bool get_upload_data_type(UCHAR* value);
IODAUSB_API    bool set_upload_data_type(UCHAR value);

// AD/DA management
IODAUSB_API    bool get_ad_start_delay(ULONG* delay);
IODAUSB_API    bool set_ad_start_delay(ULONG delay);
IODAUSB_API    bool get_ad_capture_select(UCHAR* channel);
IODAUSB_API    bool set_ad_capture_select(UCHAR channel);
IODAUSB_API    bool get_ad_input_source(UCHAR* type);
IODAUSB_API    bool set_ad_input_source(UCHAR type);
IODAUSB_API    bool get_ad_gain_type(UCHAR ch, UCHAR* type);
IODAUSB_API    bool set_ad_gain_type(UCHAR ch, UCHAR type);
IODAUSB_API    bool get_ad_value(UCHAR ch, SHORT* value);
IODAUSB_API    bool get_da_value(UCHAR ch, SHORT* value);
IODAUSB_API    bool set_da_value(UCHAR ch, SHORT value);

IODAUSB_API    bool capture_start();
IODAUSB_API    bool capture_stop();
IODAUSB_API    bool capture_adda_status(UCHAR *status);
IODAUSB_API    bool capture_bulkin_status(UCHAR *status);
IODAUSB_API    bool capture_buffer_status(UCHAR *status);

// Digital IO management
IODAUSB_API    bool get_digital_input(UCHAR *indata, UCHAR nBytesToRead);
IODAUSB_API    bool get_digital_output(UCHAR *outdata, UCHAR nBytesToRead);
IODAUSB_API    bool set_digital_output(UCHAR *outdata, UCHAR nBytesToWrite);
IODAUSB_API    bool get_digital_input_bit(UCHAR *indata, UCHAR bitnum);
IODAUSB_API    bool get_digital_output_bit(UCHAR *outdata, UCHAR bitnum);
IODAUSB_API    bool set_digital_output_bit(UCHAR outdata, UCHAR bitnum);


////////////////////////////////////////////////////////////////////////////////////////////////
//
// error handling function
//
IODAUSB_API int get_last_error();
IODAUSB_API void get_last_error_string(CHAR* cErr);


////////////////////////////////////////////////////////////////////////////////////////////////
//
// AD memory pool management function
//
IODAUSB_API    bool memory_clear();
IODAUSB_API    bool memory_read(UCHAR *pDst, ULONG nBytesToRead, ULONG *nBytesRead);
IODAUSB_API    bool memory_stacked_size(ULONG *nSize);


////////////////////////////////////////////////////////////////////////////////////////////////
//
// special functions for library interface test
//
IODAUSB_API    bool add_1b(CHAR a, CHAR* b, LONG* sum);
IODAUSB_API    bool add_2b(SHORT a, SHORT* b, LONG* sum);
IODAUSB_API    bool add_4b(LONG a, LONG* b, LONG* sum);


////////////////////////////////////////////////////////////////////////////////////////////////
//
// library management function
// note. !!! Don't use the following functions, it's served only for this library developers !!!
//
IODAUSB_API    void MutexLock();
IODAUSB_API    void MutexUnlock();
IODAUSB_API    void USBInformationPrint(char * sTitle);
IODAUSB_API    bool get_vsync_low_width(USHORT* clockcount);
IODAUSB_API    bool set_vsync_low_width(USHORT clockcount);
IODAUSB_API    bool ioda_usb_change_bulk_mode(UCHAR mode);
IODAUSB_API    bool get_upload_data_gray(SHORT* graylevel);
IODAUSB_API    bool set_upload_data_gray(SHORT graylevel);
IODAUSB_API    bool pattern_download_start();
IODAUSB_API    bool pattern_download_stop();
IODAUSB_API    bool pattern_download_status(UCHAR *status);
```

```
IODAUSB_API    bool READ_1BYTE(UCHAR cmd, UCHAR addr, UCHAR *pDst);
IODAUSB_API    bool READ_2BYTE(UCHAR cmd, UCHAR addr, USHORT *pDst);
IODAUSB_API    bool READ_4BYTE(UCHAR cmd, UCHAR addr, ULONG *pDst);
IODAUSB_API    bool READ_NBYTE(UCHAR cmd, UCHAR addr, UCHAR *pDst,
                               LONG nBytesToRead, LONG *nBytesRead);
IODAUSB_API    bool WRITE_1BYTE(UCHAR cmd, UCHAR addr, UCHAR data);
IODAUSB_API    bool WRITE_2BYTE(UCHAR cmd, UCHAR addr, USHORT data);
IODAUSB_API    bool READ_BULK(UCHAR *pDst, ULONG nBytesToRead, ULONG *nBytesRead);
IODAUSB_API    bool WRITE_BULK(UCHAR *pSrc, ULONG nByteToWrite, int EventID);

#endif // end of __IODAUSB_LIB_H__

#ifdef __cplusplus
}
#endif
```

3.2.2. Detailed View

라이브러리 소스 및 헤더에 대한 저작권/정보를 나타낸다.

```
//////////////////////////////////////////////////////////////////////////////////////////////////
//
// IODAUSB.h : IODAUSB.DLL header
//
//                                Copyright (c) 2010
//                        CANTOPS - SEOUL, REPUBLIC OF KOREA
//                                                              www.cantops.biz
//
// This program is the property of the CANTOPS. and it shall not be reproduced, distributed or
// used without permission of an authorized Company official.
// This is an unpublished work subject to Trade Secret and Copyright protection.
//
// DLL Version: V1.1.0.0
// Date: March 31, 2010
//
//////////////////////////////////////////////////////////////////////////////////////////////////
//
```

C/C++ 언어를 사용하여 프로젝트 개발하는 경우 라이브러리 링크 function name mangling 을 위한 코드이다.

```
#ifdef __cplusplus
extern "C"
{
#endif

#ifndef __IODAUSB_LIB_H__
#define __IODAUSB_LIB_H__

#ifdef IODAUSB_EXPORTS
        #define IODAUSB_API    __declspec(dllexport)
#else
        #define IODAUSB_API    __declspec(dllimport)
#endif
```

본 라이브러리에 사용된 타입 및 새롭게 정의된 데이터 형에 대한 표시이다.

```
//////////////////////////////////////////////////////////////////////////////////////////////////
//
// type definition
//
typedef unsigned char                UCHAR;
typedef  char                        CHAR;
typedef unsigned short               USHORT;
typedef unsigned short               WORD;
typedef signed   short               SHORT;
typedef unsigned int                 UINT;
typedef signed   int                 SINT;
typedef unsigned long                ULONG;
typedef signed   long                LONG;
```

메모리크기, 함수인자에 사용될 미리 정의된 값들이다.

```
////////////////////////////////////////////////////////////////////////////////////////////////////
//
// USB communication Definition
//
#define IODAUSB_MEMORY_POOL_SIZE            (0x04000000) //64 MB

#define IODAUSB_BULK_IN_BUF_SIZE            1024
#define IODAUSB_BULK_OUT_BUF_SIZE           512

#define IODAUSB_ADDA_STATUS_ADDA_STOP 0
#define IODAUSB_ADDA_STATUS_ADDA_START      1
#define IODAUSB_ADDA_STATUS_BULK_STOP 0
#define IODAUSB_ADDA_STATUS_BULK_BUSY 1
#define IODAUSB_ADDA_STATUS_BUF_EMPTY 0
#define IODAUSB_ADDA_STATUS_BUF_FULL  1

#define IODAUSB_INTERFACE_MODE_BULKIN 0
#define IODAUSB_INTERFACE_MODE_BULKOUT      1

#define IODAUSB_PATTERN_MAX                 8
```

라이브러리 사용 중 발생한 에러코드 및 에러내용

```
////////////////////////////////////////////////////////////////////////////////////////////////////
//
// error number definition
//
enum IoDaUsbErrNo
{
        IODAUSB_NOERR=0,
        IODAUSB_ERR_DEV_INIT,
        IODAUSB_ERR_DEV_CLOSE,
        IODAUSB_ERR_DEV_RESET,
        IODAUSB_ERR_DEV_ABORT,
        IODAUSB_ERR_CONTROL_READ,
        IODAUSB_ERR_CONTROL_WRITE,
        IODAUSB_ERR_BULK_READ,
        IODAUSB_ERR_BULK_WRITE,
        IODAUSB_ERR_IO_READ,
        IODAUSB_ERR_IO_WRITE,
        IODAUSB_ERR_AD_READ,
        IODAUSB_ERR_DA_READ,
        IODAUSB_ERR_DA_WRITE,
        IODAUSB_ERR_INTERFACE_MODE_CHANGE,
        IODAUSB_ERR_READ_FRAME_DATA,
        IODAUSB_ERR_WRITE_FRAME_DATA,
        IODAUSB_ERR_PATTERN_INFO_READ,
        IODAUSB_ERR_PATTERN_INFO_WRITE,
        IODAUSB_ERR_PATTERN_INFO_SELECT,
        IODAUSB_ERR_READ_LINE_DATA,
        IODAUSB_ERR_READ_STACK_DATA_LENGTH,
};

////////////////////////////////////////////////////////////////////////////////////////////////////
//
// error string definition
//
char gcIoDaUsbErrorString[][256] =
{
        "",
        "Device is not initialized.",
        "Device is not closed.",
        "Device is not reset.",
        "Device is not abort.",
        "USB Communication error. control register reading.",
        "USB Communication error. control register writing.",
        "USB Communication error. bulk data reading.",
        "USB Communication error. bulk data writing.",
        "USB Communication error. I/O data reading.",
        "USB Communication error. I/O data writing.",
        "IODA AD value read failed.",
        "IODA DA value read failed.",
        "IODA DA value write failed.",
        "USB Communication error. Interface mode change.",
        "IODA frame data read failed.",
        "IODA frame data write failed.",
        "IODA pattern image information read failed.",
        "IODA pattern image information write failed.",
```

```
        "IODA pattern image information select failed.",
        "IODA frame data read failed.",
        "IODA stack data read length is too larger than bulk-in size.",
        "",
};
```

아래는 사용자에게 노출되는 함수리스트이다. 라이브러리 초기화/USB 통신/시스템 운용에 대한 전반적인 함수들이 기술된다.

```
////////////////////////////////////////////////////////////////////////////////////////////
//
// USER FUNCTIONS
//
// library and USB communication
```

아래는 USB 통신과 관련된 함수 리스트 로서 USB 장치의 접속 및 해제, 통신에러클리어에 사용된다.

```
IODAUSB_API    bool ioda_usb_open(UCHAR devID);
IODAUSB_API    void ioda_usb_close();
IODAUSB_API    void ioda_usb_reset();
IODAUSB_API    void ioda_usb_abort();

IODAUSB_API    bool get_fpga_version(UCHAR* version);
IODAUSB_API    bool get_sampling_freq(ULONG* freq);
IODAUSB_API    bool set_sampling_freq(ULONG freq);
```

아래는 패턴데이터 처리에 대한 함수리스트 이다. 패턴 X/Y 에 대한 크기 및 저장 Address 등을 지정할수 있다.

```
// pattern management
IODAUSB_API    bool get_pattern_addr(ULONG* addr);
IODAUSB_API    bool set_pattern_addr(ULONG addr);
IODAUSB_API    bool get_pattern_metrics(USHORT* width, USHORT* height);
IODAUSB_API    bool set_pattern_metrics(USHORT width, USHORT height);
IODAUSB_API    bool select_pattern(UCHAR id);
IODAUSB_API    bool get_selected_pattern(UCHAR* id);
IODAUSB_API    bool write_frame_data(USHORT width, USHORT height, void *pSrcImage);
IODAUSB_API    bool get_pattern_memory_info(UCHAR id, ULONG* addr,
                                            USHORT* width, USHORT* height);
IODAUSB_API    bool set_pattern_memory_info(UCHAR id, ULONG addr, USHORT width, USHORT height);
IODAUSB_API    bool get_upload_data_type(UCHAR* value);
IODAUSB_API    bool set_upload_data_type(UCHAR value);
```

아래는 AD 및 DA 동작에 관한 함수들로서 AD 지연시간, 채널 선택등이 가능하다.

```
// AD/DA management
IODAUSB_API    bool get_ad_start_delay(ULONG* delay);
IODAUSB_API    bool set_ad_start_delay(ULONG delay);
IODAUSB_API    bool get_ad_capture_select(UCHAR* channel);
IODAUSB_API    bool set_ad_capture_select(UCHAR channel);
IODAUSB_API    bool get_ad_input_source(UCHAR* type);
IODAUSB_API    bool set_ad_input_source(UCHAR type);
IODAUSB_API    bool get_ad_gain_type(UCHAR ch, UCHAR* type);
IODAUSB_API    bool set_ad_gain_type(UCHAR ch, UCHAR type);
IODAUSB_API    bool get_ad_value(UCHAR ch, SHORT* value);
IODAUSB_API    bool get_da_value(UCHAR ch, SHORT* value);
IODAUSB_API    bool set_da_value(UCHAR ch, SHORT value);
IODAUSB_API    bool get_ad_sampling_ratio(UCHAR* value);
IODAUSB_API    bool set_ad_sampling_ratio(UCHAR value);

IODAUSB_API    bool capture_start();
IODAUSB_API    bool capture_stop();
IODAUSB_API    bool capture_adda_status(UCHAR *status);
IODAUSB_API    bool capture_bulkin_status(UCHAR *status);
IODAUSB_API    bool capture_buffer_status(UCHAR *status);
```

아래는 Digital 입출력에 대한 설정 확인용 함수 리스트로서 바이트 혹은 비트단위 처리가 가능하다.

```
// Digital IO management
IODAUSB_API    bool get_digital_input(UCHAR *indata, UCHAR nBytesToRead);
IODAUSB_API    bool get_digital_output(UCHAR *outdata, UCHAR nBytesToRead);
IODAUSB_API    bool set_digital_output(UCHAR *outdata, UCHAR nBytesToWrite);
IODAUSB_API    bool get_digital_input_bit(UCHAR *indata, UCHAR bitnum);
IODAUSB_API    bool get_digital_output_bit(UCHAR *outdata, UCHAR bitnum);
IODAUSB_API    bool set_digital_output_bit(UCHAR outdata, UCHAR bitnum);
```

```
////////////////////////////////////////////////////////////////////////////////////////////////
//
아래는 라이브러리 함수 수행에 대한 에러코드 확인용 함수리스트이다.
// error handling function
//
IODAUSB_API int get_last_error();
IODAUSB_API void get_last_error_string(CHAR* cErr);

////////////////////////////////////////////////////////////////////////////////////////////////
//
아래는 AD 데이터가 저장되는 라이브러리 내부의 메모리 처리에 대한 함수 리스트이다.
// AD memory pool management function
//
IODAUSB_API    bool memory_clear();
IODAUSB_API    bool memory_read(UCHAR *pDst, ULONG nBytesToRead, ULONG *nBytesRead);
IODAUSB_API    bool memory_stacked_size(ULONG *nSize);

////////////////////////////////////////////////////////////////////////////////////////////////
//
아래는 라이브러리 테스트용 함수리스트이다. Call by Value, Call by Reference 입출력테스트를 할 수 있다.
// special functions for library interface test
//
IODAUSB_API    bool add_1b(CHAR a, CHAR* b, LONG* sum);
IODAUSB_API    bool add_2b(SHORT a, SHORT* b, LONG* sum);
IODAUSB_API    bool add_4b(LONG a, LONG* b, LONG* sum);

////////////////////////////////////////////////////////////////////////////////////////////////
//
아래는 제조사 개발자를 위한 함수리스트로서 일반 사용자는 사용하면 안된다.
// library management function
// note. !!! Don't use the following functions, it's served only for this library developers !!!
//

IODAUSB_API void ioda_init();
IODAUSB_API void ioda_exit();

IODAUSB_API    void MutexLock();
IODAUSB_API    void MutexUnlock();
IODAUSB_API    void USBInformationPrint(char * sTitle);
IODAUSB_API    bool get_vsync_low_width(USHORT* clockcount);
IODAUSB_API    bool set_vsync_low_width(USHORT clockcount);
IODAUSB_API    bool ioda_usb_change_bulk_mode(UCHAR mode);
IODAUSB_API    bool get_upload_data_gray(SHORT* graylevel);
IODAUSB_API    bool set_upload_data_gray(SHORT graylevel);
IODAUSB_API    bool pattern_download_start();
IODAUSB_API    bool pattern_download_stop();
IODAUSB_API    bool pattern_download_status(UCHAR *status);

IODAUSB_API    bool READ_1BYTE(UCHAR cmd, UCHAR addr, UCHAR *pDst);
IODAUSB_API    bool READ_2BYTE(UCHAR cmd, UCHAR addr, USHORT *pDst);
IODAUSB_API    bool READ_4BYTE(UCHAR cmd, UCHAR addr, ULONG *pDst);
IODAUSB_API    bool READ_NBYTE(UCHAR cmd, UCHAR addr, UCHAR *pDst,
                               LONG nBytesToRead, LONG *nBytesRead);
IODAUSB_API    bool WRITE_1BYTE(UCHAR cmd, UCHAR addr, UCHAR data);
IODAUSB_API    bool WRITE_2BYTE(UCHAR cmd, UCHAR addr, USHORT data);
IODAUSB_API    bool READ_BULK(UCHAR *pDst, ULONG nBytesToRead, ULONG *nBytesRead);
IODAUSB_API    bool WRITE_BULK(UCHAR *pSrc, ULONG nByteToWrite, int EventID);

#endif // end of __IODAUSB_LIB_H__

#ifdef __cplusplus
}
#endif
```

3.3. Error List

Error Number	Error Message Description
IODAUSB_NOERR=0	null string. 에러 없음. 정상.
IODAUSB_ERR_DEV_INIT	Device is not initialized. 라이브러리 혹은 장치가 초기화 되지 않음.
IODAUSB_ERR_DEV_CLOSE	Device is not closed. USB 장치가 정상적으로 Close 되지 않음.
IODAUSB_ERR_DEV_RESET	Device is not reset. USB 장치가 정상적으로 Reset 되지 않음.
IODAUSB_ERR_DEV_ABORT	Device is not abort. USB 장치와의 접속 차단/에러 클리어 시도 실패.
IODAUSB_ERR_CONTROL_READ	USB Communication error. control register reading. USB 장치의 제어데이터 읽기 실패.
IODAUSB_ERR_CONTROL_WRITE	USB Communication error. control register writing. USB 장치의 제어데이터 쓰기 실패.
IODAUSB_ERR_BULK_READ	USB Communication error. bulk data reading. USB 장치의 Bulk 데이터 읽기 실패.
IODAUSB_ERR_BULK_WRITE	USB Communication error. bulk data writing. USB 장치의 Bulk 데이터 쓰기 실패.
IODAUSB_ERR_IO_READ	USB Communication error. I/O data reading. USB 장치의 IO 데이터 읽기 실패.
IODAUSB_ERR_IO_WRITE	USB Communication error. I/O data writing. USB 장치의 IO 데이터 쓰기 실패.
IODAUSB_ERR_AD_READ	IODA AD value read failed. 아나로그 입력값 읽기 실패.
IODAUSB_ERR_DA_READ	IODA DA value read failed. 아나로그 출력값 읽기 실패.
IODAUSB_ERR_DA_WRITE	IODA DA value write failed. 아나로그 출력값 쓰기 실패.
IODAUSB_ERR_INTERFACE_MODE_CHANGE	USB Communication error. Interface mode change. USB Bulk 통신 모드 변경 실패.
IODAUSB_ERR_READ_FRAME_DATA	IODA frame data read failed. 프레임 데이터 읽기 실패.
IODAUSB_ERR_WRITE_FRAME_DATA	IODA frame data write failed. 프레임 데이터 쓰기 실패.
IODAUSB_ERR_PATTERN_INFO_READ	IODA pattern image information read failed. 패턴 정보(가로 세로 크기) 읽기 실패.
IODAUSB_ERR_PATTERN_INFO_WRITE	IODA pattern image information write failed. 패턴 정보(가로 세로 크기) 쓰기 실패.
IODAUSB_ERR_PATTERN_INFO_SELECT	IODA pattern image information select failed. 활성화 패턴 선택 실패.
IODAUSB_ERR_READ_LINE_DATA	IODA frame data read failed. 실시간 처리되는 AD 값 읽기 실패.
IODAUSB_ERR_READ_STACK_DATA_LENGTH	IODA stack data read length is too larger than bulk-in size. Memory Pool 데이터 읽기 실패(파라미터 크기)

3.4. APIs

3.4.1. ioda_usb_open

Proto type
```
bool ioda_usb_open(UCHAR devID);
```

Parameter
devID
장치 고유 ID 로서 반드시 “0” 으로 고정됨. range: 0

Return
true
성공

false
실패

실패시 get_last_error 를 이용하여 error code, get_last_error_string 을 이용하여 error string 을 확인하여 적절한 조치를 하여야 한다.

Description
USB 를 이용하여 각종 파라미터 읽기/쓰기, 패턴 관련 데이터 수집을 위해서 해당 디바이스에 대한 USB 통신을 열어야 할 때 사용한다.

Example
IODA 장치와 USB 통신을 준비하는 예.

```
bool bRet;

bRet = ioda_usb_open(0);
```

3.4.2. ioda_usb_reset

Proto type
```
void ioda_usb_reset();
```

Parameter
```
None.
```

Return
```
None.
```

Description
장치를 Power-on 상태로 초기화한다. IODA 장치는 현재 작동중인 데이터가 저장되지 않으며, USB 통신이 Reset 되면 초기 Power-on 상태가 되므로 본 함수 호출 후에는 작동에 필요한 파라미터 및 데이터를 재설정 하여야 한다.

Example
IODA 장치와 USB 통신을 Reset 하는 예.

```
ioda_usb_reset();
```

3.4.3. ioda_usb_abort

Proto type
```
void ioda_usb_abort();
```

Parameter
```
None.
```

Return
```
None.
```

Description
USB 장치와 통신 중 Pending 된 통신을 취소한다.

Example
USB 통신이 Pending 된 경우 이것을 취소하는 예.

```
ioda_usb_abort();
```

3.4.4. get_library_version

Proto type
```
bool get_library_version(ULONG* version);
```

Parameter
version
사용중인 라이브러리 버전을 얻을 unsigned long 32bit 포인터.

Return
true
성공

false
실패

실패시 get_last_error 를 이용하여 error code, get_last_error_string 을 이용하여 error string 을 확인하여 적절한 조치를 하여야 한다.

Description
라이브러리 버전을 구한다. 버전번호는 4 자리로 표시되며 각 digit 는 1 바이트로 구성된다.
버전 형식: 0xabcdefgh → ab.cd.ef.gh.
ex)리턴된 값이 16908288 (;0x01020000) 이라면, 이것은 1.2.0.0 을 의미한다.

Example
라이브러리 버전을 구하는 예.

```
bool  bRet;
ULONG version;

bRet = get_library_version(&version);
```

3.4.5. get_fpga_version

Proto type
```
bool get_fpga_version(UCHAR* version);
```

Parameter
version
IODA 메인보드의 FPGA 펌웨어 버전을 얻을 unsigned 8bit 포인터.

Return
true
성공

false
실패

실패시 get_last_error 를 이용하여 error code, get_last_error_string 을 이용하여 error string 을 확인하여 적절한 조치를 하여야 한다.

Description
IODA 메인보드의 FPGA 펌웨어 버전을 구한다.

Example
IODA 메인보드의 FPGA 펌웨어 버전을 구하는 예.

```
bool  bRet;
UCHAR version;

bRet = get_fpga_version(&version);

<Result>
version: 1
```

3.4.6. get_sampling_freq

Proto type
```
bool get_sampling_freq(ULONG* freq);
```

Parameter
`freq`
설정된 ADC Sampling 주파수를 얻을 unsigned 32bit 포인터.
`Range: 0 ~ 16777215 (;0 ~ 0x00FFFFFF)`

Return
`true`
성공

`false`
실패

실패시 `get_last_error` 를 이용하여 `error code`, `get_last_error_string` 을 이용하여 `error string` 을 확인하여 적절한 조치를 하여야 한다.

Description
현재 설정된 ADC Sampling Frequency 를 얻는다. 24 비트 범위의 값이며 단위는 Hz 이다.

Example
현재 설정된 ADC Sampling Frequency 값을 얻는 예

```
bool  bRet;
ULONG freq;

bRet = get_sampling_freq(&freq);
```

3.4.7. set_sampling_freq

Proto type
```
bool set_sampling_freq(ULONG freq);
```

Parameter
`freq`
ADC Sampling 주파수 설정.
Range: 0 ~ 16777215 (;0 ~ 0x00FFFFFF)

Return
`true`
성공

`false`
실패

실패시 `get_last_error` 를 이용하여 `error code`, `get_last_error_string` 을 이용하여 `error string` 을 확인하여 적절한 조치를 하여야 한다.

Description
ADC Sampling Frequency 를 설정한다. 24 비트 범위의 값이며 단위는 Hz 이다.

Example
ADC Sampling Frequency 를 1.0 MHz 로 설정하는 예.

```
bool  bRet;
ULONG freq = 1000000;

bRet = set_sampling_freq(freq);
```

3.4.8. get_pattern_addr

Proto type
```
bool get_pattern_addr(ULONG* addr);
```

Parameter
addr
현재 작동중인 패턴이 저장된 시작 Address 값을 얻을 포인터.
Range: 0 ~ 0x1FFFFFF (;0 ~ 33554431)

Return
true
성공

false
실패

실패시 get_last_error 를 이용하여 error code, get_last_error_string 을 이용하여 error string 을 확인하여 적절한 조치를 하여야 한다.

Description
함수 호출 후 현재 작동중인 패턴이 저장된 시작 주소가 리턴 된다.

Example
현재 작동중인 패턴이 저장된 시작 Address 값을 얻는 예.

```
bool  bRet;
ULONG addr;

bRet = get_pattern_addr (&addr);
```

3.4.9. set_pattern_addr

Proto type
```
bool set_pattern_addr(ULONG addr);
```

Parameter
addr
현재 작동중인 패턴이 저장될 시작 Address 값.
Range: 0 ~ 0x1FFFFFF (;0 ~ 33554431)

Return
true
성공

false
실패

실패시 get_last_error 를 이용하여 error code, get_last_error_string 을 이용하여 error string 을 확인하여 적절한 조치를 하여야 한다.

Description
패턴 다운로드 함수 호출 전에 사용하여야 하며, Address 는 반드시 32 의 배수가 되도록 하여야 한다. 0x1000000 (O), 0x1000020 (O), 0x00001234 (X), 0x00001233 (X)
본 API 는 다운로드 할 이미지에 대한 크기 및 주소를 미리 할당(;set_pattern_memory_info) 한 후, 원하는 ID 번호로 작업 할 패턴을 선택(;select_pattern) 하는 방식이 아닌 Device 에 직접 기록 방식이므로 이렇게 기록된 정보는 select_pattern 함수에 의해 작동되지 않는다. 본 API 는 고급 사용자를 위한 함수로서 일반적으로 그 사용을 추천하지 않는다.

Example
패턴사이즈 Width=500, Height=256 와 패턴주소= 0x10000 를 장치에 직접 기록 하는 예.

```
bool  bRet;
USHORT width = 500;
USHORT height = 256;
ULONG addr = 0x10000;

bRet = set_pattern_metrics(width, height);
bRet = set_pattern_addr(addr);
```

3.4.10. get_pattern_metrics

Proto type
```
bool get_pattern_metrics(USHORT* width, USHORT* height);
```

Parameter
`width`
현재 동작 중인 패턴의 X 크기를 얻을 포인터.

`height`
현재 동작 중인 패턴의 Y 크기를 얻을 포인터.

Return
`true`
성공

`false`
실패

실패시 `get_last_error` 를 이용하여 `error code`, `get_last_error_string` 을 이용하여 `error string` 을 확인하여 적절한 조치를 하여야 한다.

Description
현재 동작 중인 패턴의 X,Y 크기를 얻어 온다.

Example
현재 동작 중인 패턴의 X,Y 크기를 얻는 예.

```
bool  bRet;
USHORT width;
USHORT height;

bRet = get_pattern_metrics(&width, &height);
```

3.4.11. set_pattern_metrics

Proto type
```
bool set_pattern_metrics(USHORT width, USHORT height);
```

Parameter
`width`
패턴 X 크기 설정 값.

`height`
패턴 Y 크기 설정 값.

Return
`true`
성공

`false`
실패

실패시 `get_last_error` 를 이용하여 `error code`, `get_last_error_string` 을 이용하여 `error string` 을 확인하여 적절한 조치를 하여야 한다.

Description
현재 동작 중인 패턴의 X,Y 크기를 설정한다.

Example
패턴의 크기를 `Width=500, Height=256` 으로 설정 하는 예.

```
bool  bRet;
USHORT width=500;
USHORT height=256;

bRet = set_pattern_metrics(width, height);
```

3.4.12. select_pattern

Proto type
```
bool select_pattern(UCHAR id);
```

Parameter
`id`
선택될 패턴 `ID`.
`Range: 0 ~ 7`

Return
`true`
성공

`false`
실패

실패시 `get_last_error` 를 이용하여 `error code`, `get_last_error_string` 을 이용하여 `error string` 을 확인하여 적절한 조치를 하여야 한다.

Description
IODA 는 패턴 데이터를 최대 8 개 까지 미리 다운로드 한 후 선택적으로 사용할 수 있다. 본 함수는 이용하면 미리 저장된 ID 만을 사용하여 편리하게 패턴의 X/Y 크기 및 Address 를 설정할 수 있다.

Example
패턴 #3 을 선택하는 예.

```
bool bRet;

bRet = select_pattern(3);
```

3.4.13. get_selected_pattern

Proto type
```
bool get_selected_pattern(UCHAR* id);
```

Parameter
`id`
현재 설정된 패턴 번호를 얻어올 포인터.
`Range: 0 ~ 7`

Return
`true`
성공

`false`
실패

실패시 `get_last_error` 를 이용하여 `error code`, `get_last_error_string` 을 이용하여 `error string` 을 확인하여 적절한 조치를 하여야 한다.

Description
현재 설정된 패턴 번호를 얻어 온다.

Example
현재 설정된 패턴 번호를 얻는 예.

```
bool  bRet;
UCHAR id;

bRet = get_selected_pattern(&id);
```

3.4.14. write_frame_data

Proto type
```
bool write_frame_data(USHORT width, USHORT height, void *pSrcImage);
```

Parameter
`width`
패턴의 가로 크기.

`height`
패턴의 세로 크기.

`pSrcImage`
패턴 데이터 시작 포인터. 데이터는 `-32768 ~ +32767` 의 범위를 갖는 `signed short` 형식으로 저장되어 있어야 하며 `Byte Order` 는 `Little endian` 을 따른다.
패턴의 총 크기 = `width*height*4 (byte)`.

메모리

Return
`true`
성공

`false`
실패

실패시 `get_last_error` 를 이용하여 `error code`, `get_last_error_string` 을 이용하여 `error string` 을 확인하여 적절한 조치를 하여야 한다.

Description
본 API 는 PatternX, PatternY 데이터를 Device 로 전송한다. 함수 호출전에 각각의 패턴 데이터는 충분한 메모리에 정해진 포멧으로 저장되어 있어야 한다. 본 함수가 호출되면 실행중인 ADC 작업이 중지된다.
주의.
Library Version 1.2.0.4 이전을 사용하는 경우: 실제 패턴 데이터 기록 직전에 1024 바이트의 Dummy 패턴을 기록하여야 함.

Example
가로 100 x 세로 50 크기의 패턴 X/Y 를 Device 에 기록하는 예.

```
bool   bRet;
USHORT width=100;
USHORT height=50;
SHORT* pData;

pSrcImage = new SHORT[width*height*2];
bRet = write_frame_data(width, height, pSrcImage);
delete [] pSrcImage;
```

3.4.15. get_pattern_memory_info

Proto type

```
bool get_pattern_memory_info( UCHAR id, ULONG* addr,
                              USHORT* width, USHORT* height);
```

Parameter

`id`
구하려는 패턴 `ID. Range: 0 ~ 7`

`addr`
패턴 어드레스를 얻을 포인터.

`width`
패턴의 가로 크기를 얻을 포인터.

`height`
패턴의 세로 크기를 얻을 포인터.

Return

`true`
성공

`false`
실패

실패시 `get_last_error` 를 이용하여 `error code, get_last_error_string` 을 이용하여 `error string` 을 확인하여 적절한 조치를 하여야 한다.

Description

본 API 는 기 설정된 패턴에 대한 정보를 알고자 할 때 사용된다.

Example

패턴 #3 의 정보를 얻는 예.

```
bool   bRet;
ULONG  addr;
USHORT width,height;

bRet = get_pattern_memory_info(3, &addr, &width, &height);
```

3.4.16. set_pattern_memory_info

Proto type

```
bool set_pattern_memory_info( UCHAR id, ULONG addr,
                              USHORT width, USHORT height);
```

Parameter

`id`
설정하려는 패턴 ID. Range: 0 ~ 7

`addr`
패턴 어드레스 설정값.

`width`
패턴의 가로 크기 설정값.

`height`
패턴의 세로 크기 설정값.

Return

`true`
성공

`false`
실패

실패시 `get_last_error` 를 이용하여 `error code`, `get_last_error_string` 을 이용하여 `error string` 을 확인하여 적절한 조치를 하여야 한다.

Description

본 API 는 패턴별로 각각 설정해야 하는 가로크기, 세로크기, 패턴시작주소를 일괄 처리하도록 고안되었다.
본 API 호출 후 select_pattern 함수를 호출하여야만 설정된 데이터가 반영된다. address 지정시 설정하려는 패턴 ID 의 전후에 위치한 패턴들의 영역과 중첩여부를 신중히 고려하여야 한다.
또한, 패턴의 시작 `Address` 는 반드시 `32` 의 배수가 되도록 하여야 한다.
`0x1000000 (O), 0x1000020 (O), 0x00001234 (X), 0x00001233 (X)`

Example

4 개의 패턴 그룹을 만들고 #3 을 선택하는 예.

```
bool   bRet;
UCHAR  id=3;
ULONG  addr[4]={0,0x200000,0x400000,0x600000};
USHORT W[4]={100,200,300,400};
USHORT H[4]={150,160,170,180};

bRet = set_pattern_memory_info(id,addr[id],W[id],H[id]);
```

3.4.17. get_ad_start_delay

Proto type
```
bool get_ad_start_delay(ULONG* delay);
```

Parameter
delay
DA 출력 직후부터 AD 변환 직전까지의 지연 시간을 얻을 포인터.
단위: Sampling Clock Count

Return
true
성공

false
실패

실패시 get_last_error 를 이용하여 error code, get_last_error_string 을 이용하여 error string 을 확인하여 적절한 조치를 하여야 한다.

Description
AD 작업 지연 시간을 얻는다.

Example
AD 작업 지연 시간을 얻는 예.

```
bool bRet;
ULONG delay;

bRet = get_ad_start_delay(&delay);
```

3.4.18. set_ad_start_delay

Proto type
```
bool set_ad_start_delay(ULONG delay);
```

Parameter
DA 출력 직후부터 AD 변환 직전까지의 지연 시간 설정.
단위: Sampling Clock Count

Return
true
성공

false
실패

실패시 get_last_error 를 이용하여 error code, get_last_error_string 을 이용하여 error string 을 확인하여 적절한 조치를 하여야 한다.

Description
AD 작업 지연 시간을 설정한다.

Example
AD 작업 지연 시간을 10 sampling clock 으로 설정하는 예.

```
bool bRet;

bRet = set_ad_start_delay(10);
```

3.4.19. get_ad_capture_select

Proto type
```
bool get_ad_capture_select(UCHAR* channel);
```

Parameter
channel
AD 입력 채널을 얻을 포인터. value 별 각각의 의미는 아래와 같다.
    0: Channel 1
    1: Channel 2

Return
true
성공

false
실패

실패시 get_last_error 를 이용하여 error code, get_last_error_string 을 이용하여 error string 을 확인하여 적절한 조치를 하여야 한다.

Description
현재 동작중인 AD 입력 채널을 알고자 할 때 사용한다.

Example
현재 동작중인 AD 입력 채널을 구하는 예.

```
bool  bRet;
UCHAR channel;

bRet = get_ad_capture_select(&channel);
```

3.4.20. set_ad_capture_select

Proto type
```
bool set_ad_capture_select(UCHAR channel);
```

Parameter
channel
AD 입력 채널 설정값. value 별 각각의 의미는 아래와 같다.
    0: Channel 1
    1: Channel 2

Return
true
성공

false
실패

실패시 get_last_error 를 이용하여 error code, get_last_error_string 을 이용하여 error string 을 확인하여 적절한 조치를 하여야 한다.

Description
현재 동작중인 AD 입력 소스를 알고자 할 때 사용한다.

Example
현재 동작중인 AD 입력 채널을 CH2 로 변경하는 예.

```
bool  bRet;

bRet = set_ad_capture_select(1);
```

3.4.21. get_ad_input_source

Proto type
```
bool get_ad_input_source(UCHAR* type);
```

Parameter
channel
AD 입력 타입을 얻을 포인터. value 별 각각의 의미는 아래와 같다.
```
    0: AD Input
    1: DA Output
    2: Reference 6.95V
    3: Ground
```

Return
true
성공

false
실패

실패시 get_last_error 를 이용하여 error code, get_last_error_string 을 이용하여 error string 을 확인하여 적절한 조치를 하여야 한다.

Description
현재 동작중인 AD 입력 타입을 알고자 할 때 사용한다.

Example
현재 동작중인 AD 입력 타입을 구하는 예.

```
bool  bRet;
UCHAR type;

bRet = get_ad_input_source(&type);
```

3.4.22. set_ad_input_source

Proto type
```
bool set_ad_input_source(UCHAR type);
```

Parameter
channel
AD 입력 타입 설정. value 별 각각의 의미는 아래와 같다.
    0: AD Input
    1: DA Output
    2: Reference 6.95V
    3: Ground

Return
true
성공

false
실패

실패시 get_last_error 를 이용하여 error code, get_last_error_string 을 이용하여 error string 을 확인하여 적절한 조치를 하여야 한다.

Description
현재 동작중인 AD 입력 타입을 설정하고자 할 때 사용한다.

Example
현재 동작중인 AD 입력 타입을 AD Input (;0) 으로 설정하는 예.

```
bool  bRet;

bRet = set_ad_input_source(0);
```

3.4.23. get_ad_gain_type

Proto type
```
bool get_ad_gain_type(UCHAR ch, UCHAR* type);
```

Parameter
ch
채널 번호. Range: 1 ~ 2.
    1: Channel 1
    2: Channel 2

type
AD 게인 타입을 얻어올 포인터. value 별 각각의 의미는 아래와 같다.
    0: x 0.4 (+/- 10.0 Volt)
    1: x 0.8 (+/- 5.0 Volt)
    2: x 2 (+/- 2.0 Volt)
    3: x 4 (+/- 1.0 Volt)

Return
true
성공

false
실패

실패시 get_last_error 를 이용하여 error code, get_last_error_string 을 이용하여 error string 을 확인하여 적절한 조치를 하여야 한다.

Description
지정된 채널에 적용된 게인 타입을 얻는다.

Example
채널 1 의 게인 타입을 얻는 예.

```
bool  bRet;
UCHAR type;

bRet = get_ad_gain_type(1, &type);
```

3.4.24. set_ad_gain_type

Proto type
```
bool set_ad_gain_type(UCHAR ch, UCHAR type);
```

Parameter
ch
채널 번호. Range: 1 ~ 2.
    1: Channel 1
    2: Channel 2

type
AD 게인 타입 설정값. value 별 각각의 의미는 아래와 같다.
    0: x0.4 (+/- 10.0 Volt)
    1: x0.8 (+/- 5.0 Volt)
    2: x2 (+/- 2.0 Volt)
    3: x4 (+/- 1.0 Volt)

Return
true
성공

false
실패

실패시 get_last_error 를 이용하여 error code, get_last_error_string 을 이용하여 error string 을 확인하여 적절한 조치를 하여야 한다.

Description
지정된 채널에 적용된 게인 타입을 설정한다.

Example
채널 1 의 게인 타입을 x2 (+/- 2.0 Volt) 로 설정하는 예.

```
bool  bRet;
UCHAR type=2;

bRet = set_ad_gain_type(1, type);
```

3.4.25. get_ad_value

Proto type
```
bool get_ad_value(UCHAR ch, SHORT* value);
```

Parameter
ch
채널 번호. Range: 1 ~ 2.
   1: Channel 1
   2: Channel 2

value
AD 값을 얻을 포인터. Range: -32768 ~ +32767.

Return
true
성공

false
실패

실패시 get_last_error 를 이용하여 error code, get_last_error_string 을 이용하여 error string 을 확인하여 적절한 조치를 하여야 한다.

Description
지정된 AD 채널의 AD 값을 얻는다.

Example
채널 2 의 현재 변환중인 AD 값을 얻는 예

```
bool  bRet;
SHORT value;

bRet = get_ad_value(2, &value);
```

3.4.26. get_da_value

Proto type
```
bool get_da_value(UCHAR ch, SHORT* value);
```

Parameter
ch
채널 번호. Range: 1 ~ 2.
   1: Channel 1
   2: Channel 2

value
DA 값을 얻을 포인터. Range: -32768 ~ +32767.

Return
true
성공

false
실패

실패시 get_last_error 를 이용하여 error code, get_last_error_string 을 이용하여 error string 을 확인하여 적절한 조치를 하여야 한다.

Description
지정된 DA 채널의 DA 값을 얻는다.

Example
채널 2 의 DA 값을 얻는 예

```
bool  bRet;
SHORT value;

bRet = get_da_value(2, &value);
```

3.4.27. set_da_value

Proto type
```
bool set_da_value(UCHAR ch, SHORT value);
```

Parameter
ch
채널 번호. Range: 1 ~ 2.
   1: Channel 1
   2: Channel 2

value
DA 값 설정. Range: -32768 ~ +32767.

Return
true
성공

false
실패

실패시 get_last_error 를 이용하여 error code, get_last_error_string 을 이용하여 error string 을 확인하여 적절한 조치를 하여야 한다.

Description
지정된 DA 채널의 DA 값을 수동으로 설정하고자 할 때 사용한다.

Example
채널 2 의 DA 값을 1000 으로 설정 예

```
bool  bRet;
SHORT value=1000;

bRet = set_da_value(2, value);
```

3.4.28. get_ad_sampling_ratio

Proto type
```
bool get_ad_sampling_ratio(UCHAR* value);
```

Parameter
value
AD Sampling Clock Ratio 값을 얻을 포인터. Range: 1 ~ 255.

Return
true
성공

false
실패

실패시 get_last_error 를 이용하여 error code, get_last_error_string 을 이용하여 error string 을 확인하여 적절한 조치를 하여야 한다.

Description
AD Sampling Clock Ratio 값을 얻을 때 사용한다. IODAUSB 보드는 하나의 패턴 데이터 DA 출력 후 지정된 샘플링 수 만큼의 DA 입력을 저장할 수 있다. AD 샘플링 횟수가 많아지면, 입력에 대한 오류를 최소화 할 수 있지만, 처리해야 될 데이터 량이 많아지는 단점이 있으므로 시스템의 목적에 맞도록 값을 지정할 필요가 있다.

Example
현재 설정된 AD Sampling Clock Ratio 값을 얻는 예

```
bool  bRet;
UCHAR value;

bRet = get_ad_sampling_ratio(&value);
```

3.4.29. set_ad_sampling_ratio

Proto type
```
bool set_ad_sampling_ratio(UCHAR value);
```

Parameter
value
AD Sampling Clock Ratio 값. Range: 1 ~ 255.

Return
true
성공

false
실패

실패시 get_last_error 를 이용하여 error code, get_last_error_string 을 이용하여 error string 을 확인하여 적절한 조치를 하여야 한다.

Description
AD Sampling Clock Ratio 값을 지정할 때 사용한다. IODAUSB 보드는 하나의 패턴 데이터 DA 출력 후 지정된 샘플링 수 만큼의 DA 입력을 저장할 수 있다. AD 샘플링 횟수가 많아지면, 입력에 대한 오류를 최소화 할 수 있지만, 처리해야 될 데이터 량이 많아지는 단점이 있으므로 시스템의 목적에 맞도록 값을 지정할 필요가 있다.

Example
DA 1 회당 5 회의 AD 샘플링을 지정하는 예

```
bool  bRet;
UCHAR value = 5;

bRet = set_ad_sampling_ratio(value);
```

3.4.30. capture_start

Proto type
```
bool capture_start();
```

Parameter
```
None.
```

Return
```
true
```
성공

```
false
```
실패

실패시 `get_last_error` 를 이용하여 `error code`, `get_last_error_string` 을 이용하여 `error string` 을 확인하여 적절한 조치를 하여야 한다.

Description
Pattern X, Pattern Y 데이터를 참조하여 Capture 작업(DA 출력 및 DA 입력)을 개시한다. Capture 작업은 한번 시작되면 정지 함수 호출전까지 무한 반복된다.

Example
Capture 작업을 시작하는 예.

```
bool bRet;

bRet = capture_start();
```

3.4.31. capture_stop

Proto type
```
bool capture_stop();
```

Parameter
None.

Return
`true`
성공

`false`
실패

실패시 `get_last_error` 를 이용하여 `error code`, `get_last_error_string` 을 이용하여 `error string` 을 확인하여 적절한 조치를 하여야 한다.

Description
Capture 작업(DA 출력 및 DA 입력)을 정지한다.

Example
Capture 작업을 정지하는 예.

```
bool bRet;

bRet = capture_stop();
```

3.4.32. capture_adda_status

Proto type
```
bool capture_adda_status(UCHAR *status);
```

Parameter
status
Capture 작업 상태를 얻어올 포인터. value 별 각각의 의미는 아래와 같다.
    0: Capture 정지 상태. (;IODAUSB_ADDA_STATUS_ADDA_STOP)
    1: Capture 실행 상태. (;IODAUSB_ADDA_STATUS_ADDA_START)

Return
true
성공

false
실패

실패시 get_last_error 를 이용하여 error code, get_last_error_string 을 이용하여 error string 을 확인하여 적절한 조치를 하여야 한다.

Description
Capture 작업 상태를 얻는다.

Example
Capture 작업 상태를 얻는 예.

```
bool bRet;

bRet = capture_adda_status(&status);
```

3.4.33. capture_bulkin_status

Proto type
```
bool capture_bulkin_status(UCHAR *status);
```

Parameter
status
Capture 작업 후 얻은 AD 값의 전송 상태를 얻어올 포인터.
value 별 각각의 의미는 아래와 같다.
    0: Capture 데이터 전송이 완료된 상태.(; IODAUSB_ADDA_STATUS_BULK_STOP)
    1: Capture 데이터 전송이 진행중인 상태.(; IODAUSB_ADDA_STATUS_BULK_BUSY)

Return
true
성공

false
실패

실패시 get_last_error 를 이용하여 error code, get_last_error_string 을 이용하여 error string 을 확인하여 적절한 조치를 하여야 한다.

Description
Capture 작업 후 얻은 AD 값의 전송 상태를 얻는다. Capture 작업이 시작되어 AD 값이 발생하면 Device 는 즉시 그 값을 전송 시작하며 IODAUSB_ADDA_STATUS_BULK_BUSY 상태가 된다. 전송이 완료되면 IODAUSB_ADDA_STATUS_BULK_STOP 상태가 되고 전송버퍼는 비워진다.

Example
Capture 작업 후 얻은 AD 값의 전송 상태를 얻는 예.

```
bool bRet;

bRet = capture_bulkin_status (&status);
```

3.4.34. capture_buffer_status

Proto type
```
bool capture_buffer_status(UCHAR *status);
```

Parameter
status
Capture 작업 버퍼 상태를 얻어올 포인터.
value 별 각각의 의미는 아래와 같다.
    0: Capture 데이터 버퍼가 여유있는 상태.(; IODAUSB_ADDA_STATUS_BUF_EMPTY)
    1: Capture 데이터 버퍼가 모두 점유된 상태.(; IODAUSB_ADDA_STATUS_BUF_FULL)

Return
true
성공

false
실패

실패시 get_last_error 를 이용하여 error code, get_last_error_string 을 이용하여 error string 을 확인하여 적절한 조치를 하여야 한다.

Description
IODA 는 Capture 작업 후 얻은 AD 값을 내부 버퍼에 기록하며 외부에서 이 버퍼의 데이터를 읽어가면 버퍼는 비워진다. Capture 작업이 계속되는 도중, 버퍼의 데이터를 읽어가지 않으면 일정시간 후 버퍼는 모두 채워지게 되는데, IODAUSB_ADDA_STATUS_BUF_FULL 상태가 된다. Buffer 가 Full 된 후에도 Capture 작업은 계속되며 가장 오래된 데이터부터 데이터의 중복이 시작된다. Buffer 가 Full 된 이후의 데이터는 실제 AD 값과 상당한 차이가 있으므로 이런 상태가 발생하지 않도록 상위 프로그램의 데이터 처리 구조는 최대한 빠르게 버퍼의 값을 읽어가도록 설계되어야 한다.

Example
Capture 작업 버퍼 상태를 확인하여 버퍼가 모두 점유된 경우 Capture 작업을 중지하는 예.

```
bool bRet;

bRet = capture_buffer_status (&status);
if(bRet)
{
      if( status == IODAUSB_ADDA_STATUS_BUF_FULL ) capture_stop();
}
```

3.4.35. get_digital_input

Proto type
```
bool get_digital_input(UCHAR *indata, UCHAR nBytesToRead);
```

Parameter
indata
Digital 입력 데이터를 읽어올 포인터.

nBytesToRead
읽을 바이트 수 지정. IODA 는 입력 접점이 1 바이트(8 비트) 이하로 표현되므로 nBytesToRead 값은 1 로 고정된다.

Return
true
성공

false
실패

실패시 get_last_error 를 이용하여 error code, get_last_error_string 을 이용하여 error string 을 확인하여 적절한 조치를 하여야 한다.

Description
Digital Input 값을 바이트 단위로 읽어 오는 예.

Example
example

```
bool  bRet;
UCHAR indata;

bRet = get_digital_input(&indata, 1);
```

3.4.36. get_digital_output

Proto type
```
bool get_digital_output(UCHAR *outdata, UCHAR nBytesToRead);
```

Parameter
outdata
Digital 출력 데이터를 읽어올 포인터.

nBytesToRead
읽을 바이트 수 지정. IODA 는 출력 접점이 1 바이트(8 비트) 이하로 표현되므로 nBytesToRead 값은 1 로 고정된다.

Return
true
성공

false
실패

실패시 get_last_error 를 이용하여 error code, get_last_error_string 을 이용하여 error string 을 확인하여 적절한 조치를 하여야 한다.

Description
Digital Output 값을 바이트 단위로 읽어 오는 예.

Example
example

```
bool  bRet;
UCHAR outdata;

bRet = get_digital_output(&outdata, 1);
```

3.4.37. get_digital_input_bit

Proto type
```
bool get_digital_input_bit(UCHAR *indata, UCHAR bitnum);
```

Parameter
indata
Digital 입력 데이터를 읽어올 포인터. value 별 각각의 의미는 아래와 같다.
    0: OFF
    1: ON

bitnum
비트 번호 지정.
Range: 0 ~ 3

Return
true
성공

false
실패

실패시 get_last_error 를 이용하여 error code, get_last_error_string 을 이용하여 error string 을 확인하여 적절한 조치를 하여야 한다.

Description
지정된 입력 비트 번호에 해당하는 접점의 ON/OFF 상태를 얻는다.

Example
Digital Input Bit 2 의 값을 얻는 예.

```
bool  bRet;
UCHAR indata;

bRet = get_digital_input_bit(&indata, 2);
```

3.4.38. get_digital_output_bit

Proto type
```
bool get_digital_output_bit(UCHAR *outdata, UCHAR bitnum);
```

Parameter
outdata
Digital 출력 데이터를 읽어올 포인터. value 별 각각의 의미는 아래와 같다.
    0: OFF
    1: ON

bitnum
비트 번호 지정.
Range: 0 ~ 7

Return
true
성공

false
실패

실패시 get_last_error 를 이용하여 error code, get_last_error_string 을 이용하여 error string 을 확인하여 적절한 조치를 하여야 한다.

Description
지정된 출력 비트 번호에 해당하는 접점의 ON/OFF 상태를 얻는다.

Example
Digital Output Bit 5 의 값을 얻는 예.

```
bool  bRet;
UCHAR outdata;

bRet = get_digital_output_bit(&outdata, 5);
```

3.4.39. set_digital_output_bit

Proto type
```
bool set_digital_output_bit(UCHAR outdata, UCHAR bitnum);
```

Parameter
outdata
Digital 출력 데이터를 설정. value 별 각각의 의미는 아래와 같다.
    0: OFF
    1: ON

bitnum
비트 번호 지정.
Range: 0 ~ 7

Return
true
성공

false
실패

실패시 get_last_error 를 이용하여 error code, get_last_error_string 을 이용하여 error string 을 확인하여 적절한 조치를 하여야 한다.

Description
지정된 출력 비트 번호에 해당하는 접점을 ON/OFF 한다.

Example
Digital Output Bit 5 접점을 ON 시키는 예.

```
bool  bRet;
UCHAR outdata=1;

bRet = set_digital_output_bit(outdata, 5);
```

3.4.40. get_last_error

Proto type
```
int get_last_error();
```

Parameter
None.

Return
마지막에 발생된 에러코드를 반환한다. 에러코드에 대한 자세한 내용은 Error List 항목을 참조한다.

Description
None.

Example
장치를 Open 후 에러코드, 에러내용을 확인 하는 예.

```
bool  bRet;
int   nErrCode;
char  cbuf[256];

bRet = ioda_open();
if(!bRet)
{
      nErrCode = get_last_error();
      if(nErrCode!= IODAUSB_NOERR) get_last_error_string(cbuf);
}
```

3.4.41. get_last_error_string

Proto type
```
void get_last_error_string(CHAR* cErr);
```

Parameter
cErr
에러내용이 복사될 char 형 포인터.

Return
None.

Description
마지막에 발생된 에러내용을 반환한다. 에러내용에 대한 자세한 내용은 Error List 항목을 참조한다. 에러 문자열 최대 크기: 256 바이트.

Example
장치를 Open 후 에러코드, 에러내용을 확인 하는 예.

```
bool  bRet;
int   nErrCode;
char  cbuf[256];

bRet = ioda_open();
if(!bRet)
{
      nErrCode = get_last_error();
      if(nErrCode!= IODAUSB_NOERR) get_last_error_string(cbuf);
}
```

3.4.42. memory_clear

Proto type
```
bool memory_clear();
```

Parameter
```
None
```

Return
```
true
```
성공

```
false
```
실패

실패시 `get_last_error` 를 이용하여 `error code`, `get_last_error_string` 을 이용하여 `error string` 을 확인하여 적절한 조치를 하여야 한다.

Description
DLL 내부 버퍼 저장된 AD 값을 모두 Clear 한다.

Example
Capture 수행직전 까지 쌓인 값을 모두 지우고 Capture 를 시작하는 예.

```
bool bRet;

bRet = capture_stop();
bRet = memory_clear();
bRet = capture_start();
```

3.4.43. memory_read

Proto type
```
bool memory_read(UCHAR *pDst, ULONG nBytesToRead, ULONG *nBytesRead);
```

Parameter
pDst
데이터가 저장될 버퍼의 포인터.

nBytesToRead
읽고자 하는 데이터 크기.
Range: 0 ~ 67108864(;64MB)

nBytesRead
읽어온 데이터 크기.

Return
true
성공

false
실패

실패시 get_last_error 를 이용하여 error code, get_last_error_string 을 이용하여 error string 을 확인하여 적절한 조치를 하여야 한다.

Description
DLL 의 버퍼에 저장된 데이터를 읽는다. 버퍼는 FIFO 구조로 운영되므로 실시간 데이터 처리가 필요하다면 상위 프로그램의 데이터 처리구조를 최대한 빠르고 빈번하게 버퍼의 데이터를 읽어 버퍼에 데이터가 최소한으로 쌓이도록 설계하여야 한다.
데이터 버퍼링은 Capture 기능이 시작됨과 동시에 시작되고, Capture 기능이 정지됨과 동시에 중지된다. memory_stacked_size 함수를 이용하여 현재 쌓여 있는 데이터 량을 확인할 수 있고, memory_clear 함수를 이용하여 버퍼에 쌓인 데이터를 모두 비울 수 있다. capture_start 함수를 호출하면 이전의 데이터는 모두 삭제되고 새로운 버퍼링 작업을 개시한다.

Example
Capture 기능을 100ms 수행후 그 값을 모두 읽어오는 예.

```
bool bRet;
UCHAR *pDst;
ULONG nBytesToRead, nBytesRead;
bRet = capture_stop();
bRet = capture_start();
Sleep(100);
bRet = capture_stop();
pDst = new UCHAR[1000];
nBytesToRead = 1000;

// memory_stacked_size 함수를 이용하여 버퍼에 쌓인 데이터 개수만큼 읽을 수 있음.
```

```
do
{
      nBytesRead = 0;
      memory_read(pDst, nBytesToRead, &nBytesRead);
} while ( 0 < nBytesRead );

delete [] pDst;
```

3.4.44. memory_stacked_size

Proto type
```
bool memory_stacked_size(ULONG *nSize);
```

Parameter
`nSize`
DLL 내부 버퍼에 쌓인 AD 데이터 량.

Return
`true`
성공

`false`
실패

실패시 `get_last_error` 를 이용하여 `error code`, `get_last_error_string` 을 이용하여 `error string` 을 확인하여 적절한 조치를 하여야 한다.

Description
DLL 내부 버퍼에 쌓인 AD 데이터 량을 알고자 할 때 사용한다. 상위 프로그램의 데이터 처리구조는 최대한 빠르고 빈번하게 버퍼의 데이터를 읽어 (;memory_read 함수 사용) 버퍼에 데이터가 최소한으로 쌓이도록 설계하여야 한다.

Example
DLL 내부 버퍼에 쌓인 처리되지 않은 AD 데이터 량을 확인하는 예.

```
bool  bRet;
ULONG nSize;

bRet =  memory_stacked_size(&nSize);
```

3.4.45. add_1b

Proto type
```
bool add_1b(CHAR a, CHAR* b, LONG* sum);
```

Parameter
a
가산될 데이터.

b
가산될 데이터가 저장된 포인터.

sum
결과값을 받을 포인터. sum = a + *b 연산 결과를 반환한다.

Return
true
성공

false
실패

실패시 get_last_error 를 이용하여 error code, get_last_error_string 을 이용하여 error string 을 확인하여 적절한 조치를 하여야 한다.

Description
라이브러리 테스트용 함수이다.

Example
11 + 22 = 33 테스트 예.

```
bool bRet;
CHAR A=11;
CHAR B=22;
LONG sum=0;

bRet = add_1b(A, &B, &sum);
```

3.4.46. add_2b

Proto type
```
bool add_2b(SHORT a, SHORT* b, LONG* sum);
```

Parameter
a
가산될 데이터.

b
가산될 데이터가 저장된 포인터.

sum
결과값을 받을 포인터. sum = a + *b 연산 결과를 반환한다.

Return
true
성공

false
실패

실패시 get_last_error 를 이용하여 error code, get_last_error_string 을 이용하여 error string 을 확인하여 적절한 조치를 하여야 한다.

Description
라이브러리 테스트용 함수이다.

Example
1111 + 2222 = 3333 테스트 예.

```
bool bRet;
SHORT A=11;
SHORT B=22;
LONG sum=0;

bRet = add_1b(A, &B, &sum);
```

3.4.47. add_4b

Proto type
```
bool add_4b(LONG a, LONG* b, LONG* sum);
```

Parameter
a
가산될 데이터.

b
가산될 데이터가 저장된 포인터.

sum
결과값을 받을 포인터. sum = a + *b 연산 결과를 반환한다.

Return
true
성공

false
실패

실패시 get_last_error 를 이용하여 error code, get_last_error_string 을 이용하여 error string 을 확인하여 적절한 조치를 하여야 한다.

Description
라이브러리 테스트용 함수이다.

Example
1111111 + 2222222= 3333333 테스트 예.

```
bool bRet;
LONG A=1111111;
LONG B=2222222;
LONG sum=0;

bRet = add_1b(A, &B, &sum);
```

3.4.48. ioda_init

Proto type
```
void ioda_init();
```

Parameter
None.

Return
None.

Description

제조사 개발자 외 일반사용자는 본 API 사용 금지함!

IODAUSB 라이브러리를 초기화 한다. 본 함수 호출후 라이브러리는 USB 통신에 대한 Device, Control EndPoint, Bulk-In EndPoint, Bulk-Out EndPoint 설정 및 내부 변수에 대한 초기화를 완료한다. IODAUSB 라이브러리 사용시 반드시 초기에 한번 호출되어야 하나, IODAUSB.DLL 을 최초 사용하는 Process 가 실행될 때 DLL Main 루틴에서 자동으로 호출되도록 설계되었으므로 사용자는 호출할 필요가 없다.

Example
None.

3.4.49. ioda_usb_close

Proto type
```
void ioda_usb_close();
```

Parameter
```
None.
```

Return
```
None.
```

Description
USB 통신을 종료 할 때 사용한다. USB 통신중 비정상 상황이 발생한 경우에도 사용된다. 본 함수 호출 후 USB 관련 Device 와 각 EndPoint 객체 및 Pointer 는 초기화 되므로, 종료 후 다시 통신을 open 할 필요가 발생할 때는 반드시 ioda_usb_open 함수를 사용하여 통신을 open 시켜야 한다. 본 함수는 ioda_usb_open 함수 호출시 자동으로 호출되므로, 특별한 경우를 제외하면 일반적인 경우 본 함수를 호출할 필요는 거의 없다.

Example
IODA 장치와 USB 통신을 종료하는 예.

```
ioda_usb_close();
```

3.4.50. ioda_exit

Proto type
```
void ioda_exit();
```

Parameter
```
None.
```

Return
```
None.
```

Description

제조사 개발자 외 일반사용자는 본 API 사용 금지함!

IODAUSB 라이브러리를 사용중 이용했던 모든 Resource 를 반납하고 라이브러리를 종료 한다.. IODAUSB 라이브러리 사용후 반드시 호출되어야 하나, 최후로 사용중인 Process 가 종료되어 IODAUSB.DLL 이 프로세스로부터 분리되는 시점에서 자동으로 호출되도록 설계되었으므로 사용자는 호출할 필요가 없다.

Example
```
None.
```

3.4.51. pattern_download_start

Proto type
```
bool pattern_download_start();
```

Parameter
None.

Return
`true`
성공

`false`
실패

실패시 `get_last_error` 를 이용하여 `error code`, `get_last_error_string` 을 이용하여 `error string` 을 확인하여 적절한 조치를 하여야 한다.

Description

제조사 개발자 외 일반사용자는 본 API 사용 금지함!

패턴 데이터 Download 시작을 Device 에게 알려준다.

Example
None.

3.4.52. pattern_download_stop

Proto type
```
bool pattern_download_stop();
```

Parameter
None.

Return
`true`
성공

`false`
실패

실패시 `get_last_error` 를 이용하여 `error code`, `get_last_error_string` 을 이용하여 `error string` 을 확인하여 적절한 조치를 하여야 한다.

Description

제조사 개발자 외 일반사용자는 본 API 사용 금지함!

패턴 데이터 Download 가 완료되었음을 Device 에게 알려준다.

Example
None.

3.4.53. pattern_download_status

Proto type
```
bool pattern_download_status(UCHAR *status);
```

Parameter
status
Pattern 데이터 Bulk-Out 통신 상태를 얻을 포인터. value 별 각각의 의미는 아래와 같다.
    0: Bulk-Out Stopped
    1: Bulk-Out Busy

Return
true
성공

false
실패

실패시 get_last_error 를 이용하여 error code, get_last_error_string 을 이용하여 error string 을 확인하여 적절한 조치를 하여야 한다.

Description

제조사 개발자 외 일반사용자는 본 API 사용 금지함!

패턴 데이터 다운로드에 관련된 Bulk-Out 통신 상태를 얻을 때 사용된다.

Example
None.

3.4.54. get_upload_data_type

Proto type
```
bool get_upload_data_type(UCHAR* value);
```

Parameter
value
Bulk-In 으로 받는 데이터 종류를 반환. value 별 각각의 의미는 아래와 같다.
    0: AD Input (;Normal Mode)
    1: Gray (전송데이터가 Increment 되는 Mode)
    2: Pattern X 데이터
    3: Pattern Y 데이터

Return
true
성공

false
실패

실패시 get_last_error 를 이용하여 error code, get_last_error_string 을 이용하여 error string 을 확인하여 적절한 조치를 하여야 한다.

Description

제조사 개발자 외 일반사용자는 본 API 사용 금지함!

Bulk-In 통신으로 어떤 종류의 데이터가 전송되는지를 알려준다.

Example
None.

3.4.55. set_upload_data_type

Proto type
```
bool set_upload_data_type(UCHAR value);
```

Parameter
value
Bulk-In 으로 받을 데이터의 종류를 설정. value 별 각각의 의미는 아래와 같다.
    0: AD Input (;Normal Mode)
    1: Gray (전송데이터가 Increment 되는 Mode)
    2: Pattern X 데이터
    3: Pattern Y 데이터

Return
true
성공

false
실패

실패시 get_last_error 를 이용하여 error code, get_last_error_string 을 이용하여 error string 을 확인하여 적절한 조치를 하여야 한다.

Description

제조사 개발자 외 일반사용자는 본 API 사용 금지함!

Bulk-In 통신으로 어떤 종류의 데이터를 받을지 설정한다. 본 API 호출은 시스템에 심각한 문제를 초래할 수 있으므로 사용법을 충분히 숙지하여야 한다.

Example
None.

3.4.56. MutexLock

Proto type
```
void MutexLock();
```

Parameter
```
None.
```

Return
```
None.
```

Description

제조사 개발자 외 일반사용자는 본 API 사용 금지함!

DLL 내부의 데이터 Protection 을 위해 Resource 를 Protection 한다.

Example
```
None.
```

3.4.57. MutexUnlock

Proto type
```
void MutexUnlock();
```

Parameter
```
None.
```

Return
```
None.
```

Description

제조사 개발자 외 일반사용자는 본 API 사용 금지함!

DLL 내부의 데이터 Protection 을 위해 Protection 된 Resource 를 Release 한다.

Example
```
None.
```

3.4.58. USBInformationPrint

Proto type
```
void USBInformationPrint(char * sTitle);
```

Parameter
`sTitle`
디버그 출력 제목 스트링.

Return
`None.`

Description

제조사 개발자 외 일반사용자는 본 API 사용 금지함!

연결된 `USB Object` 에 대한 정보를 `Debug Window` 에 출력한다.

Example
`None.`

3.4.59. get_vsync_low_width

Proto type
```
bool get_vsync_low_width(USHORT* clockcount);
```

Parameter
clockcount
프레임간 작업 지연 시간을 얻어올 포인터. 단위: ADC Sampling Count.

Return
parameter

Return
true
성공

false
실패

실패시 get_last_error 를 이용하여 error code, get_last_error_string 을 이용하여 error string 을 확인하여 적절한 조치를 하여야 한다.

Description

제조사 개발자 외 일반사용자는 본 API 사용 금지함!

프레임간 작업 지연 시간을 얻어온다.

Example
None.

3.4.60. set_vsync_low_width

Proto type
```
bool set_vsync_low_width(USHORT clockcount);
```

Parameter
`clockcount`
프레임간 작업 지연 시간 설정. 단위: ADC Sampling Count.

Return
`parameter`

Return
`true`
성공

`false`
실패

실패시 `get_last_error` 를 이용하여 `error code`, `get_last_error_string` 을 이용하여 `error string` 을 확인하여 적절한 조치를 하여야 한다.

Description

제조사 개발자 외 일반사용자는 본 API 사용 금지함!

프레임간 작업 지연 시간을 설정한다.

Example
`None.`

3.4.61. ioda_usb_change_bulk_mode

Proto type
```
bool ioda_usb_change_bulk_mode(UCHAR mode);
```

Parameter
mode
USB 통신 모드 변경. value 별 각각의 의미는 아래와 같다.
    0: IODAUSB_INTERFACE_MODE_BULKIN
    1: IODAUSB_INTERFACE_MODE_BULKOUT

Return
true
성공

false
실패

실패시 get_last_error 를 이용하여 error code, get_last_error_string 을 이용하여 error string 을 확인하여 적절한 조치를 하여야 한다.

Description

제조사 개발자 외 일반사용자는 본 API 사용 금지함!

Bulk-In/Bulk-Out 모드 전환 직전에 반드시 호출되어야 한다.

Example
None.

3.4.62. get_upload_data_gray

Proto type
```
bool get_upload_data_gray(SHORT* graylevel);
```

Parameter
`graylevel`
`Gray` 초기 값 반환. `Bulk-In` 데이터 모드가 `Gray Mode` 인 경우에만 유효하다.

Return
`true`
성공

`false`
실패

실패시 `get_last_error` 를 이용하여 `error code`, `get_last_error_string` 을 이용하여 `error string` 을 확인하여 적절한 조치를 하여야 한다.

Description

제조사 개발자 외 일반사용자는 본 API 사용 금지함!

`Bulk-In` 데이터 모드가 `Gray Mode` 인 경우 초기 값 설정 확인시 사용된다.

Example
`None.`

3.4.63. set_upload_data_gray

Proto type
```
bool set_upload_data_gray(SHORT graylevel);
```

Parameter
graylevel
Gray 초기 값 설정. Bulk-In 데이터 모드가 Gray Mode 인 경우에만 유효하다.

Return
true
성공

false
실패

실패시 get_last_error 를 이용하여 error code, get_last_error_string 을 이용하여 error string 을 확인하여 적절한 조치를 하여야 한다.

Description

제조사 개발자 외 일반사용자는 본 API 사용 금지함!

Bulk-In 데이터 모드가 Gray Mode 인 경우 초기 값 설정에 사용된다.

Example
None.

3.4.64. READ_1BYTE

Proto type
```
bool READ_1BYTE(UCHAR cmd, UCHAR addr, UCHAR *pDst);
```

Parameter
```
cmd
Control Command

addr
Address

pDst
Value buffer pointer
```

Return
```
true
```
성공

```
false
```
실패

실패시 get_last_error 를 이용하여 error code, get_last_error_string 을 이용하여 error string 을 확인하여 적절한 조치를 하여야 한다.

Description

제조사 개발자 외 일반사용자는 본 API 사용 금지함!

USB 의 ControlEndPoint 를 이용하여 1 Byte 데이터를 읽는다.

Example
```
None.
```

3.4.65. READ_2BYTE

Proto type
```
bool READ_2BYTE(UCHAR cmd, UCHAR addr, USHORT *pDst);
```

Parameter
```
cmd
Control Command

addr
Address

pDst
Value buffer pointer
```

Return
```
true
```
성공

```
false
```
실패

실패시 get_last_error 를 이용하여 error code, get_last_error_string 을 이용하여 error string 을 확인하여 적절한 조치를 하여야 한다.

Description

제조사 개발자 외 일반사용자는 본 API 사용 금지함!

USB 의 ControlEndPoint 를 이용하여 2 Byte 데이터를 읽는다.

Example
```
None.
```

3.4.66. READ_4BYTE

Proto type
```
bool READ_4BYTE(UCHAR cmd, UCHAR addr, ULONG *pDst);
```

Parameter
```
cmd
Control Command

addr
Address

pDst
Value buffer pointer
```

Return
```
true
```
성공

```
false
```
실패

실패시 get_last_error 를 이용하여 error code, get_last_error_string 을 이용하여 error string 을 확인하여 적절한 조치를 하여야 한다.

Description

제조사 개발자 외 일반사용자는 본 API 사용 금지함!

USB 의 ControlEndPoint 를 이용하여 2 Byte 데이터를 읽는다.

Example
```
None.
```

3.4.67. READ_NBYTE

Proto type

```
bool READ_NBYTE(  UCHAR cmd, UCHAR addr, UCHAR *pDst,
                  LONG nBytesToRead, LONG *nBytesRead );
```

Parameter

cmd
Control Command

addr
Address

pDst
Value buffer pointer

nBytesToRead
Number of bytes for read

nBytesRead
Pointer of Number bytes read

Return

true
성공

false
실패

실패시 get_last_error 를 이용하여 error code, get_last_error_string 을 이용하여 error string 을 확인하여 적절한 조치를 하여야 한다.

Description

제조사 개발자 외 일반사용자는 본 API 사용 금지함!

USB 의 ControlEndPoint 를 이용하여 n Byte 데이터를 읽는다.

Example

None.

3.4.68. WRITE_1BYTE

Proto type
```
bool WRITE_1BYTE(UCHAR cmd, UCHAR addr, UCHAR data);
```

Parameter
```
cmd
Control Command

addr
Address

data
1 byte value for writing
```

Return
```
true
```
성공

```
false
```
실패

실패시 get_last_error 를 이용하여 error code, get_last_error_string 을 이용하여 error string 을 확인하여 적절한 조치를 하여야 한다.

Description

제조사 개발자 외 일반사용자는 본 API 사용 금지함!

USB 의 ControlEndPoint 를 이용하여 1 Byte 데이터를 기록한다.

Example
```
None.
```

3.4.69. WRITE_2BYTE

Proto type
```
bool WRITE_2BYTE(UCHAR cmd, UCHAR addr, USHORT data);
```

Parameter
```
cmd
Control Command

addr
Address

data
2 byte value for writing
```

Return
```
true
```
성공

```
false
```
실패

실패시 get_last_error 를 이용하여 error code, get_last_error_string 을 이용하여 error string 을 확인하여 적절한 조치를 하여야 한다.

Description

제조사 개발자 외 일반사용자는 본 API 사용 금지함!

USB 의 ControlEndPoint 를 이용하여 2 Byte 데이터를 기록한다.

Example
```
None.
```

3.4.70. READ_BULK

Proto type
```
bool READ_BULK(UCHAR *pDst, ULONG nBytesToRead, ULONG *nBytesRead);
```

Parameter
```
pDst
Buffer pointer for data receiving

nBytesToRead
Number of bytes to read

nBytesRead
Pointer of number received byte
```

Return
```
true
```
성공

```
false
```
실패

실패시 get_last_error 를 이용하여 error code, get_last_error_string 을 이용하여 error string 을 확인하여 적절한 조치를 하여야 한다.

Description

제조사 개발자 외 일반사용자는 본 API 사용 금지함!

USB 의 BulkIn EndPoint 를 이용하여 지정된 바이트 크기의 데이터를 읽는다.

Example
```
None.
```

3.4.71. WRITE_BULK

Proto type
```
bool WRITE_BULK(UCHAR *pSrc, ULONG nByteToWrite, int EventID);
```

Parameter
```
pSrc
Buffer pointer for data writing

nByteToWrite
Number of bytes to write

EventID
Async object index for data writing
```

Return
```
true
```
성공

```
false
```
실패

실패시 get_last_error 를 이용하여 error code, get_last_error_string 을 이용하여 error string 을 확인하여 적절한 조치를 하여야 한다.

Description

제조사 개발자 외 일반사용자는 본 API 사용 금지함!

USB 의 BulkIn EndPoint 를 이용하여 지정된 바이트 크기의 데이터를 기록한다.

Example
```
None.
```

A. APPENDIX

A.1. Pattern Data Read Structure

Pattern_X Read Process

- Total Data Size to Read : (PatternX_Width) × (PatternX_Height) × 2 ; Bytes
- Field Structure

Data Pointer	Item	Size (Byte)	Remark
ptr0	Pattern_X Data	2	(0,0)
ptr0 + 2	Pattern_X Data	2	(1,0)
ptr0 + 4	Pattern_X Data	2	(2,0)
…	…	2	…
ptr0 + (N-1)*2	Pattern_X Data	2	(PatternX_Width-1, PatternX_Height-1)

Pattern_Y Read Process

- Total Data Size to Read : (PatternY_Width) × (PatternY_Height) × 2 ; Bytes
- Field Structure

Bulk Packet No.	Item	Size (Byte)	Remark
ptr0	Pattern_Y Data	2	(0,0)
ptr0 + 2	Pattern_Y Data	2	(1,0)
ptr0 + 4	Pattern_Y Data	2	(2,0)
…	…	2	…
ptr0 + (N-1)*2	Pattern_Y Data	2	(PatternY_Width-1, PatternY_Height-1)

A.2. Pattern Data Write Structure

아래는 Pattern X/Y Raw data 를 Device 로 전송시 사용되는 데이터 형식이다. DA 출력을 위한 패턴 데이터로서 임의의 한점에 대한 패턴데이터는 16 비트 크기의 Pattern-X 와 16 비트 크기의 Pattern-Y 로 조합된 32 비트 데이터로 구성되어야 하며, Pattern-X, Pattern-Y 의 폭과 높이는 반드시 동일하여야 한다.

Pattern_X/Y Write Process

- Total Data Size to Write : (PatternX_Width) × (PatternX_Height) × 4 ; Bytes
- Field Structure

Byte Offset	Items	Byte Order
0	Pattern_X (0,0)	LSB
1	Pattern_X (0,0)	MSB
2	Pattern_Y (0,0)	LSB
3	Pattern_Y (0,0)	MSB
4	Pattern_X (1,0)	LSB
5	Pattern_X (1,0)	MSB
6	Pattern_Y (1,0)	LSB
7	Pattern_Y (1,0)	MSB
…	…	…
4× (Width-1) × (Height-1) + 0	Pattern_X (Width-1, Height-1)	LSB
4× (Width-1) × (Height-1) + 1	Pattern_X (Width-1, Height-1)	MSB
4× (Width-1) × (Height-1) + 2	Pattern_Y (Width-1, Height-1)	LSB
4× (Width-1) × (Height-1) + 3	Pattern_Y (Width-1, Height-1)	MSB

A.3. Sample Program

```
#include <stdio.h>
#include <time.h>
#include <process.h>

#include "IODAUSB.h"

#pragma comment (lib, "IODAUSB.lib")

// Pauses for a specified number of mili-seconds
void sleep(double ms)
{
	clock_t goal;
	clock_t wait = (clock_t)(ms * CLOCKS_PER_SEC/1000.0);
	goal = wait + clock();
	while( goal > clock() );
}

#define WIDTH  128
#define HEIGHT 80

void main()
{
	char	cmsg[][16]={ "Failed", "Success" };
	UCHAR	data8u=0;
	ULONG	data32u=0;
	LONG	testno=0;
	LONG	i=0,x=0,y=0;
	bool	bret=0;

	////////////////////////////////////////////////////////////////////////////////
	//-----test: USB open
	bret = ioda_usb_open(0);
	printf( "[%08d] USB Open: %s\n", testno++, cmsg[bret] );
	if(!bret) return;

	////////////////////////////////////////////////////////////////////////////////
	// test: digital outputs
	for(i=0; i<8; i++)
	{
		bret = set_digital_output_bit(1, (UCHAR)i);
		printf( "[%08d] Turn ON Digital output bit#%d: %s\n", testno++, i, cmsg[bret] );
		sleep(100);
		if(!bret) return;

		bret = set_digital_output_bit(0, (UCHAR)i);
		printf( "[%08d] Turn OFF Digital output bit#%d: %s\n", testno++, i, cmsg[bret] );
		sleep(100);
		if(!bret) return;
	}

	////////////////////////////////////////////////////////////////////////////////
	//-----test: pattern download
	SHORT	pattern_x[WIDTH*HEIGHT];
	SHORT	pattern_y[WIDTH*HEIGHT];
	UCHAR	pattern_xy[WIDTH*HEIGHT*4];

	// create 16bit patternX/Y data
	for(i=0; i<HEIGHT*WIDTH; i++)
	{
		pattern_x[i] = (SHORT)i;
		pattern_y[i] = (SHORT)0xaabb;
	}
	// create pattern raw data which are to be downloaded to device
	for(i=0; i<HEIGHT*WIDTH; i++)
	{
		pattern_xy[i*4+0] = (pattern_x[i]>>0)&0xFF; //pattern_x lsb
		pattern_xy[i*4+1] = (pattern_x[i]>>8)&0xFF; //pattern_x msb
		pattern_xy[i*4+2] = (pattern_y[i]>>0)&0xFF; //pattern_y lsb
		pattern_xy[i*4+3] = (pattern_y[i]>>8)&0xFF; //pattern_y msb
	}
	printf( "[%08d] Ready patterns data\n", testno++ );

	// set pattern information
	data8u  = 0; // set current pattern id to #0
	data32u = 0; // set pattern start addrress
	bret = set_pattern_memory_info(data8u, data32u, WIDTH, HEIGHT);
```

```
        printf( "[%08d] Set pattern information: %s\n", testno++, cmsg[bret] );
        if(!bret) return;

        // select pattern
        bret = select_pattern(data8u);
        printf( "[%08d] Select pattern #0: %s\n", testno++, cmsg[bret] );
        if(!bret) return;

        // download pattern_x and pattern_y
        bret = write_frame_data(1024, 1, pattern_xy); // dummy download
        bret = write_frame_data(WIDTH*2, HEIGHT, pattern_xy); //2: patterns means 2 patterns are
mixed
        printf( "[%08d] PatternX download to device: %s\n", testno++, cmsg[bret] );
        if(!bret) return;

        ////////////////////////////////////////////////////////////////////
        //-----test: Pattern X Read which was downloaded to the device
        // remove all the previous captured data
        capture_stop();
        bret = memory_clear();
        printf( "[%08d] Clear data: %s\n", testno++, cmsg[bret] );
        if(!bret) return;

        set_upload_data_type(2); // <--2: upload type assign to pattern x

        // ready to upload
        capture_start();

        // get data
        ULONG nBytesToRead,nBytesRead,nByteDone=0;
        ULONG nReadTotal = WIDTH*HEIGHT*2;
        UCHAR * pBuffer = new UCHAR[nReadTotal];
        while(nByteDone<nReadTotal)
        {
                if( IODAUSB_BULK_IN_BUF_SIZE <= (nReadTotal-nByteDone) )
                        nBytesToRead = IODAUSB_BULK_IN_BUF_SIZE;
                else
                        nBytesToRead = nReadTotal - nByteDone;

                bret = memory_read(pBuffer+nByteDone, nBytesToRead, &nBytesRead);
                printf( "[%08d] Number of bytes PatternX Received=%d: %s\n", testno++, nBytesRead,
cmsg[bret] );
                if(!bret) return;

                nByteDone +=nBytesRead;
        }

        // show received pattern data
        i=0;
        while(i<(LONG)nByteDone)
        {
                if((i%32)==0) printf("\n           %08X: ", i);
                printf("%02X ", pBuffer[i++]);
        }
        delete [] pBuffer;
        printf("\n");

        // ADDA stop
        bret = capture_stop();
        printf( "[%08d] Capture stop: %s\n", testno++, cmsg[bret] );

        ////////////////////////////////////////////////////////////////////////////////
        //-----test: Read ADC Input
        // remove all the previous captured data

        capture_stop();
        printf( "[%08d] Clear data: %s\n", testno++, cmsg[bret] );
        if(!bret) return;

        set_upload_data_type(0); // <--0: Bulk-In read type assign to ADC Input

        // ready to upload
        capture_start();

        // get captured data
        if( IODAUSB_BULK_IN_BUF_SIZE <= (WIDTH*2) )
                nBytesToRead = IODAUSB_BULK_IN_BUF_SIZE;
        else
                nBytesToRead = WIDTH*2;
        UCHAR * pAdcBuffer = new UCHAR[nBytesToRead];
        nBytesRead = 0;
```

```
        bret = memory_read(pAdcBuffer, nBytesToRead, &nBytesRead);
        printf( "[%08d] Number of bytes ADC Data Received=%d: %s\n", testno++, nBytesRead,
cmsg[bret] );
        if(!bret) return;
        // show received pattern data
        i=0;
        while(i<(LONG)nBytesRead)
        {
                if((i%32)==0) printf("\n           %08X: ", i);
                printf("%02X ", pAdcBuffer[i++]);
        }

        delete [] pAdcBuffer;
        printf("\n");


        ////////////////////////////////////////////////////////////////////////////////
        printf( "[%08d] Test End.\n", testno++ );

        system("pause");
}
```