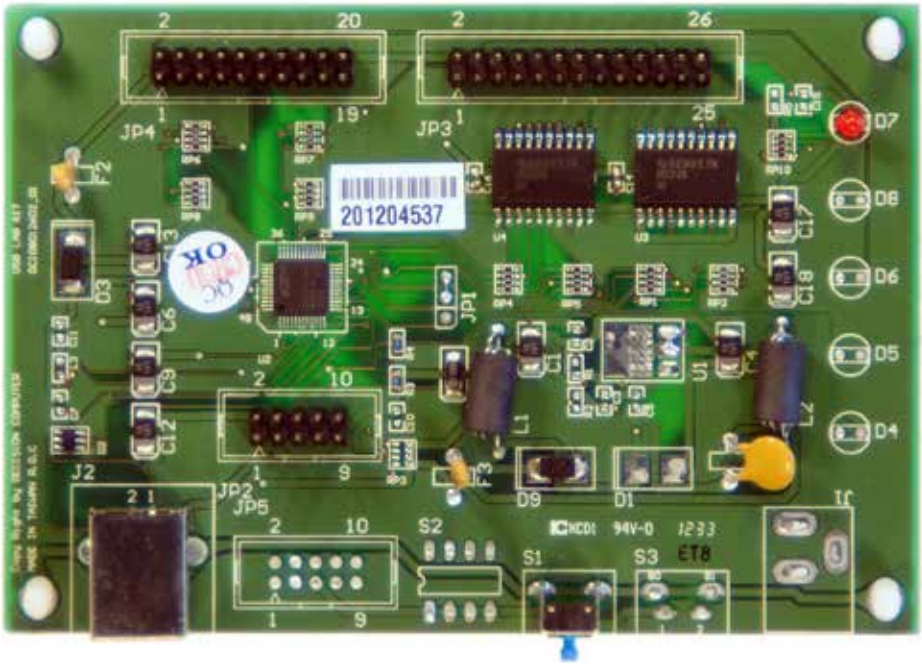
USB-I/O Handbuch



USB-Starter - Platine

8 Digitale Eingänge + 8 Digitale Ausgänge
8 Analog/Digital Eingänge

DECISION-COMPUTER Jürgen Merz e.K.
Lengericher Str. 21
49536 Lienen
Telefon +49 (0)5483-77002
Telefax +49 (0)5483-77003
http://www.decision-computer.de

Daten

Product Code:

AUSBSTARTER USB STARTER

USTART - Platinenversion

Bus: USB 2.0

Beschreibung:

Die USB-Starter ist eine teilbestückte USB-LAB-Platine

8 digitale Eingänge
8 digitale Ausgänge

Each digital I/O provides voltage range from
0V to 3.5V, where 0 to 0.4V is OFF and
2.8V to 3.4V is ON.

Support 8 analog to digital channels

Each analog to digital channels supports
10 bit 0~10V ADC input

Features:

High Speed 8051 µC Core
USB 2.0 Function Controller
Unterstützt USB ID Einstellunggen 0~14

POWER DC+5V 0.5A vom USB-Bus

Abmessungen 115mm(L) x 80mm(B) x 12mm(H)

Betriebstemperatur-Bereich 0 bis 55C.

Relative Feuchtigkeit von 0 bis 90%.

Software/Treiber:

Englisches Handbuch mit Einstellplan, Anschlußplan und Programmierbeispielen auf CD. Für Windows-Vista, Win-7 wird das HID Interface genutzt + Programmierbeispiele. Linux-Treiber + Programmierbeispiele
.
Pro Lieferung erhalten Sie eine frisch gebrannte „Decision-Computer Deutschland Service CD“ mit aktuellen Treibern, Handbüchern, Installationsanleitungen und deutschen Zusatzinformationen.

Der Umfang ist vom Produkt abhängig!

Packungsinhalt:

USB-I/O, USB-Kabel, Software/Handbuch-CD

Sicherheitshinweis

Dieses Produkt ist nicht ausfallsicher und darf daher nicht in Anwendungen verwendet werden, wo Gefah-ren für Gesundheit, Leben, und Sachwerte auftreten können! Anschluß und Reparaturen sind nur vom Fachmann zulässig.

Beim Einbau in eine Maschine oder Anlage, ist sicherzustellen, dass nach dem Einbau weiterhin die maßgeblichen Bestimmungen, Vorschriften und Richtlinien eingehalten werden!

Diese Produkte kommen mit elektrischer Spannung in Berührung, daher müssen die gültigen VDE-Vorschriften beachtet werden, insbesondere VDE 0100, VDE 0550/0551, VDE 0700, VDE 0711 und VDE 0860.

J1 USB Anschluss USB-B

VCC	+5 VDC (USB VBUS POWER)
D-	Data -
D+	Data +
SGND	Signal Ground

B

S1 Reset Taster

Systemreset bei "hängendem" USB-Modul

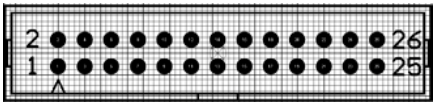
Pin	Signals
1	Reset SW+
2	Reset SW-

Digitale Eingänge /Ausgänge (JP3)

Pin	Signal	Description
1		
2	SGND	Signal Ground
3		
4	SGND	Signal Ground
5	P0D00	IN Port 0/Line 0
6	P0D01	IN Port 0/Line 1
7	P0D02	IN Port 0/Line 2
8	P0D03	IN Port 0/Line 3
9	P0D04	IN Port 0/Line 4
10	P0D05	IN Port 0/Line 5
11	P0D06	IN Port 0/Line 6
12	P0D07	IN Port 0/Line 7
13	P0D08	OUT Port 1/Line 0
14	P0D09	OUT Port 1/Line 1
15	P0D10	OUT Port 1/Line 2
16	P0D11	OUT Port 1/Line 3
17	P0D12	OUT Port 1/Line 4
18	P0D13	OUT Port 1/Line 5
19	P0D14	OUT Port 1/Line 6
20	P0D15	OUT Port 1/Line 7
21	SGND	Signal Ground
22	SGND	Signal Ground
23	+5V	+5V von der Platine (USB)
24	SGND	Signal Ground
25		
26	SGND	Signal Ground

Digital I/O Spannungsbereich von 0V -3.5V
0 bis 0.4V OFF
2,8 bis 3.4V ON

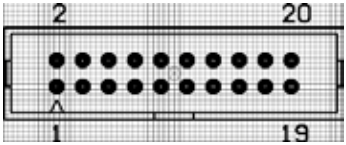
Die Signal-Zuordnungen für die digitale Ein-/Ausgabe werden in der Tabelle gezeigt.



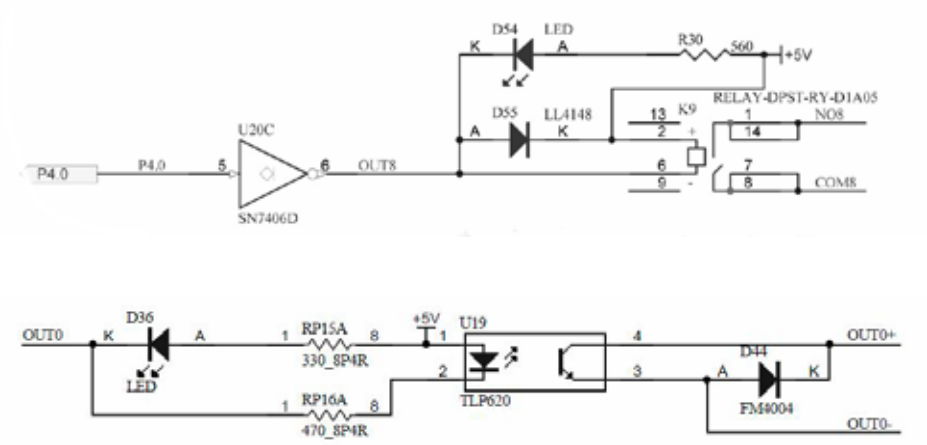
Analog/Digital-Eingänge

Pin	Signal	Description
1		
2	SGND	Signal Ground
3	ADIN0	Analog unipolar input channel 0
4	SGND	Signal Ground
5	ADIN1	Analog unipolar input channel 1
6	SGND	Signal Ground
7	ADIN2	Analog unipolar input channel 2
8	SGND	Signal Ground
9	ADIN3	Analog unipolar input channel 3
10	SGND	Signal Ground
11	ADIN4	Analog unipolar input channel 4
12	SGND	Signal Ground
13	ADIN5	Analog unipolar input channel 5
14	SGND	Signal Ground
15	ADIN6	Analog unipolar input channel 6
16	SGND	Signal Ground
17	ADIN7	Analog unipolar input channel 7
18	SGND	Signal Ground
19	+5V	+5V Power
20	SGND	Signal Ground

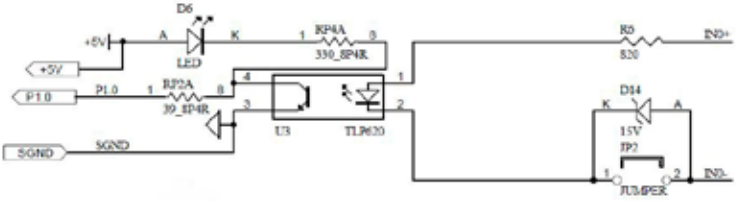
Jeder Analog/Digitale Kanal unterstützt 10 Bits Auflösung und einen Spannungsbereich von 0 ~ 10V



Schaltplan Ausgänge - Beispiel



Schaltplan Eingänge - Beispiel



Starke elektromagnetischen Quellen wie Stromleitungen, großen Elektromotoren, Schaltern oder Schweißmaschinen können starke elektromagnetische Interferenzen verursachen. Auch bei Video-Monitore und -Kabel sind starke Störquellen.

Wenn das Kabel durch einen Bereich mit beträchtlicher elektromagnetischer Störung geführt werden muss, sollten abgeschirmte Leitungen mit Erdung an der Signalquelle verlegt werden.

Vermeiden Sie es Ihre Signalkabel parallel zu einer Hochspannungsleitung platzieren! Legen Sie das Signalkabel in rechten Winkel zur Stromleitung um unerwünschte Auswirkungen zu minimieren.

Installation

Die Decision-Computer USB Geräte nutzen das HID (Human Interface Device). Da das HID zur Generic Device Class gehört ist der Treiber im Betriebssystem integriert. Wenn ein neues HID-Gerät angeschlossen wird ist keine Treiberinstallation erforderlich. Die Funktionen für Zugriff und Kontrolle des HID befinden sich in der Windows hid.dll im System32 Ordner.

Installationsbeispiel Windows-7

1. USB-Verbindung herstellen. USB-Port mit 500 mA Leistung oder Hub mit Netzteil nutzen.

2. USB-Eingabegerät - Gerätetreiber-software erfolgreich installiert

3. USB-Eingabegerät - Verwendung jetzt möglich

4. In der Systemsteuerung/Hardware erscheint jetzt das Decision-USB-Modul

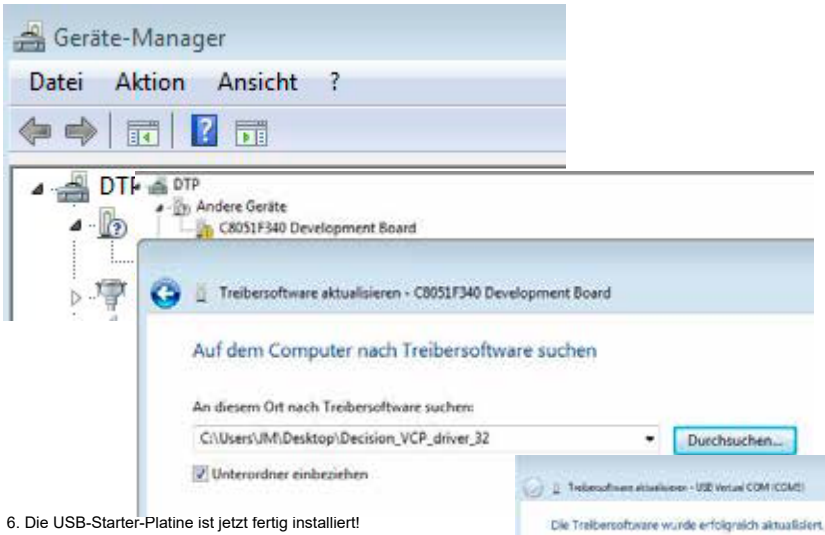
6. Der Geräte-Manager zeigt noch ein „!“. Ursache ist ein fehlender Treiber für die serielle Schnittstelle, die bei der voll bestückten Version USB-Lab vorhanden ist. Die USB-Starter-Platine ist eine teilbestückte USB-LAB-Platine. Daher muss der Treiber ohne vorhandenen Port installiert werden! Dieser Teil der Firmware könnte geänder werden, aber dann entfällt die Möglichkeit der Nachbestückung auf der Platine!

VCP Treiber (Nur für das LABKIT)

Der Virtuelle COM-Anschlusstreiber (VCP) bewirkt, daß das USB-Gerät als zusätzlicher COM-Port auf dem PC verfügbar wird. Anwendungs-Software kann auf die gleiche Weise auf das USB-Gerät zugreifen, als ob es auf einen Standard-COM-Port zugreifen würde. Diese Funktion ist nur im USBLABKIT implementiert.

Beispiel Win-8 USB-Starter wird an den PC angeschlossen
Im Geräte-Manager wird unter „Eingabegeräte (Human Interface Device) ein HID-Konformes, vom Hersteller definiertes Gerät“ installiert.
Damit ist die, für die Funktion der Platine, notwendige Funktion abgeschlossen.

Unter „Andere Geräte“ erscheint mit Ausrufezeichen „C8051F340 Development Board“
Dort Treiber aktualisieren auswählen
Auf dem PC suchen lassen
Auf der gelben CD im Verzeichnis „test_and_tool“ suchen lassen.
Ein „Virtual Communications Port“ wird installiert.
Dieser Port ist nicht herausgeführt und hat aber nur bei der USB-Lab Platine eine Funktion



6. Die USB-Starter-Platine ist jetzt fertig installiert!

Beispielinstallation Win-8:

USB-Starter wird an den PC angeschlossen
Im Geräte-Manager wird unter „Eingabegeräte (Human Interface Device) ein HID-Konformes, vom Hersteller definiertes Gerät“ installiert.
Damit ist die, für die Funktion der Platine, notwendige Funktion abgeschlossen.

Unter „Andere Geräte“ erscheint mit Ausrufezeichen „C8051F340 Development Board“
Dort Treiber aktualisieren auswählen
Auf dem PC suchen lassen
Auf der gelben CD im Verzeichnis „test_and_tool“ suchen lassen.
Ein „Virtual Communications Port“ wird installiert.
Dieser Port ist nicht herausgeführt und hat aber nur bei der USB-Lab Platine eine Funktion

SOFTWARE-PROGRAMMIERUNG UNTER WINDOWS UND LINUX

Unter Windows bieten wir als Programmierhilfe eine Funktions-Bibliothek und Dll-Datei. Das Handbuch „USBDll_Manual.pdf“ und Demo-Code in VB/VC/Delphi finden Sie auf der Decision-Studio-CD.

Linux-Anwendern bieten wir eine C-Source für den direkten Zugriff auf die USB-Geräte. Handbuch und Beispiel finden Sie unter „Dcihid-0.5.1.tgz“.

DIAGNOSE UNTER WINDOWS

USB Test Program.exe ist ein Diagnoseprogramm zum Testen USB-Geräten unter Windows
Die USB-Test Software ist auf der Decision-Studio-CD zu finden.

Die Beispiele und Treiber werden fortlaufend weiterentwickelt. Die aktuelle Version finden Sie auch auf der Decision-Computer-Merz „Service-CD“.

Eine wichtige Informationsmationsquelle ist immer das Internet **http://www.usb-industrial.com**

Software-Support auf dem kurzen Weg: **http://www.usb-industrial.com/support.html**

USB-Industrial.com Übersicht:

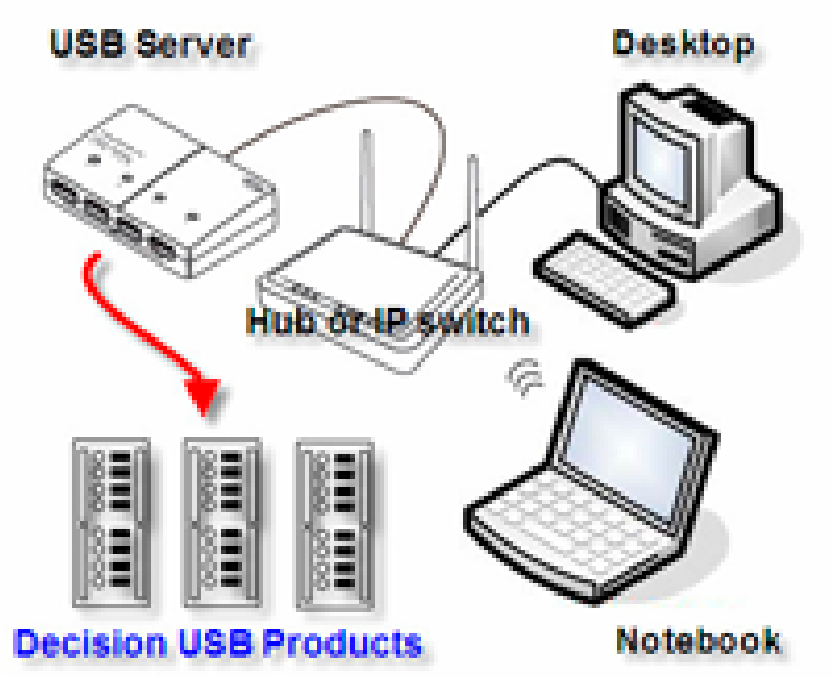
Windows Support	2010/04 USBDll.dll 2.0.0.4	This package includes Dynamic-link library which is developed by Decision Computer to communicate with the USB Series Device. It can be included in multiple computer language (VB6, VC6, VB.NET, C# Delphi) under Windows.
Watchdog Timer		This watchdog timer is a kind of software timer that triggers a system reset or other corrective action if the main program, due to some fault condition. The intention is to bring the system back from the unresponsive state into normal operation. This function is new released and please contact us to get further information.
VCP driver	(For LABKIT Only)	Virtual COM port (VCP) drivers cause the USB device to appear as an additional COM port available to the PC. Application software can access the USB device in the same way as it would access a standard COM port. This function is only implemented in USBLABKIT
Linux Support	dcihid - 0.5.1 Basic function library and demo program 2009.05.01	This package includes a c library and a demo program which is developed by Decision Computer to communicate with the USB Series Device under Linux. It also includes a ReadMe file to demonstrate how to use it and package‘s format is .tgz.
Firmware Update	Firmware Hex file Download	This Package includes a driver and a software which is developed by Decision Computer to update the newest firmware into the USB Series Device. When new version of firmware is released, user can follow the instructions to update the firmware.
LabVIEW Support	LabVIEW 8 LabVIEW 2009	This package includes manual and examples which demonstrate how to connect and develop USB Series Device under LabVIEW,which is a well-known platform and development environment for a visual programming language from NI.
ProfiLAB Support		This package includes manual and examples which demonstrate how to connect and develop USB Series Device under ProfiLAB, which is a well-known platform and development environment for a visual programming language from Abacom.
Init Value Setting Tool	(For Output Channel)	The Init Value Setting Tool is a software tool to set init value for output channel. User can use this tool to plan output channel as default high or default low when power on.
Data Acquisition and Remote Monitoring Tool		The Data Acquisition and Remote Monitoring Tool (DARMT) is a software tool to record high/low state reports at local computer, and transmit them to FTP site to achieve data acquisition and remote monitoring

USB per LAN oder Wireless

Die Fernbedienung von Decision-USB-Produkten per LAN oder Wireless mit einem Steuer-PC ist sehr einfach mit einem Multi-Port-USB-Server oder auch einer Fritzbox möglich.

Da kein Treiber installiert werden muss, ist die Installation und Programmierung sehr einfach.

Unter Windows sind die externen USB-I/O direkt im Geräte-Manager zu sehen und lassen sich verbinden oder steuern wie im ursprünglichen Host-PC.

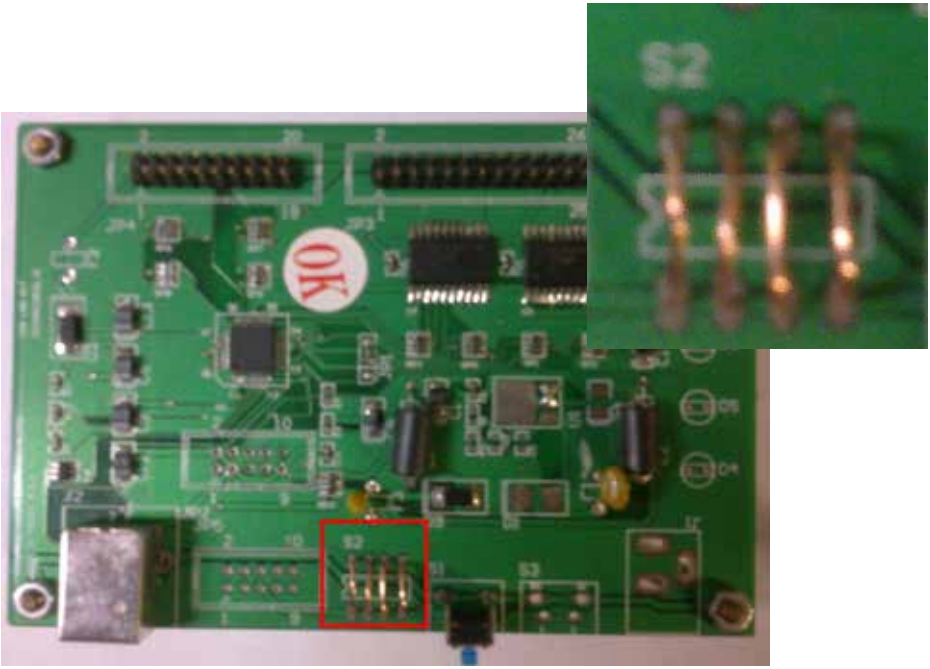


Firmware-Update

1. Remove the external input signal Voltage and only support device power.

2. Use 4 little wires to connect each of two points on S2, just like the demonstration below.

3. Connect PC to the Board by USB



4. When connecting the wires, computer will treat the board as a different device, it needs to install driver. If this is the first to use this function, please indicate the driver install path to the Driver Folder to install the driver.

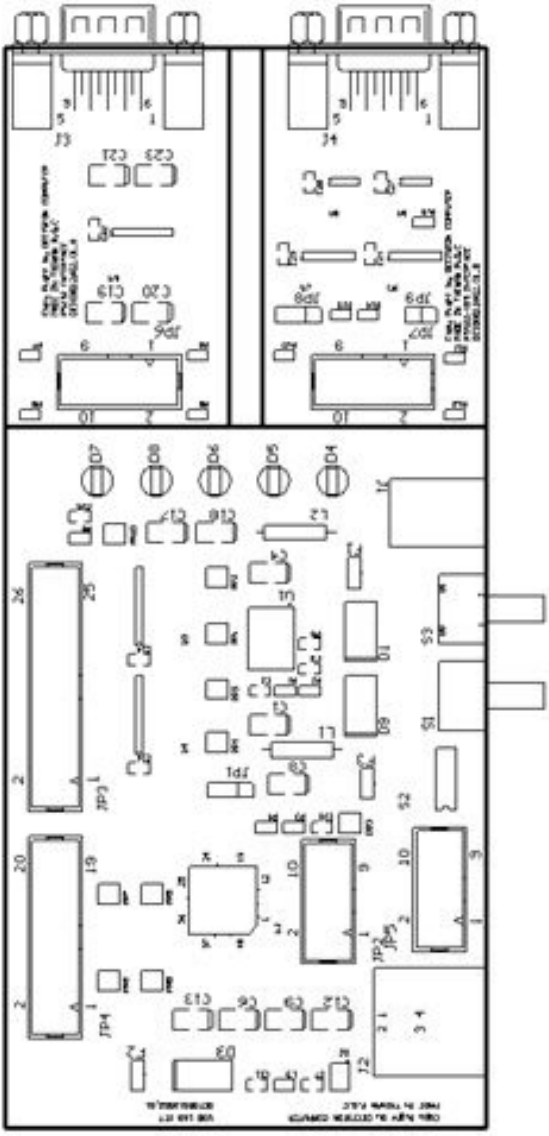
5. Open the Software USBBootloader.exe and press the Open button and indicate the hex file and then press the Download button to update firmware.

6. Disconnect from PC and remove the wires.

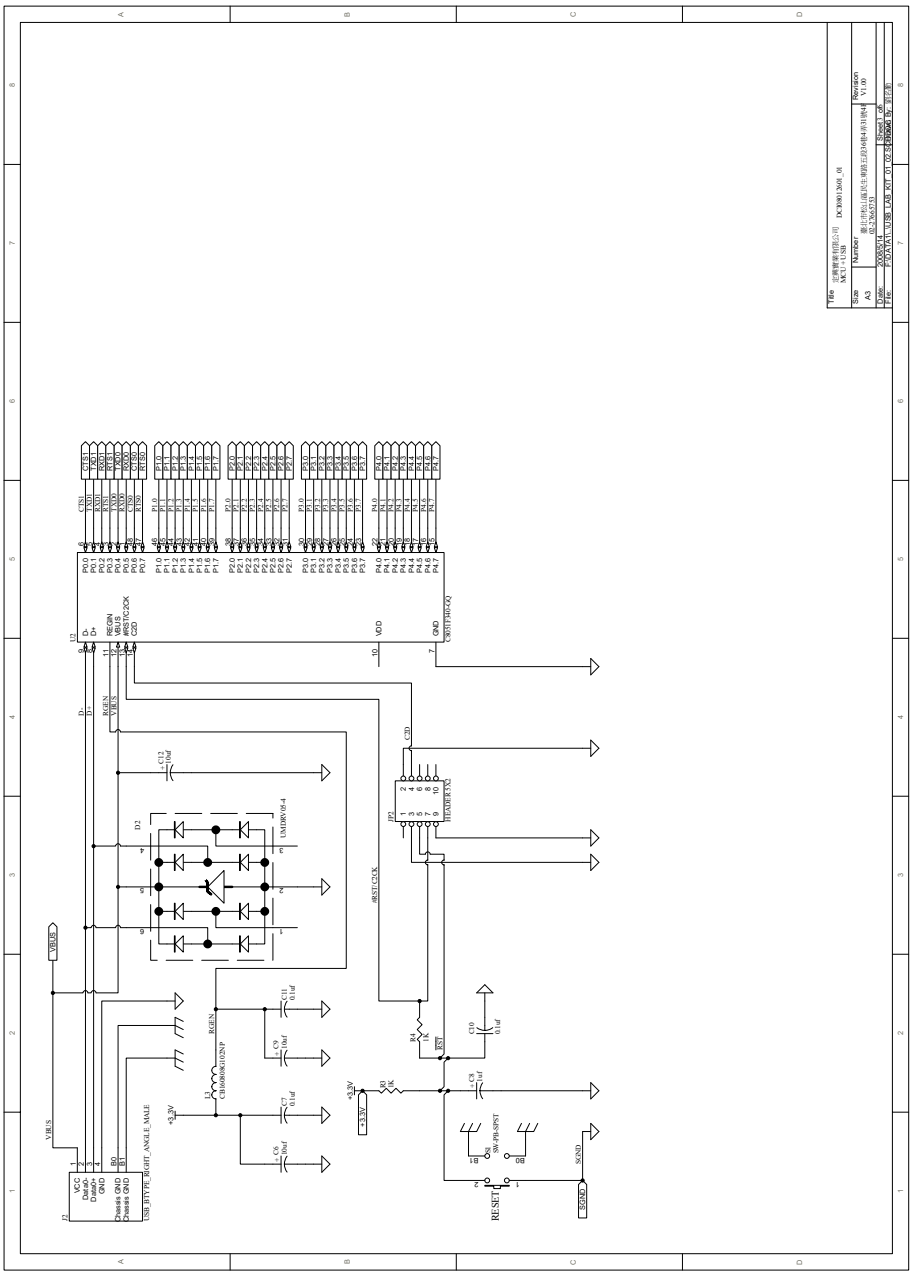
Schaltplan - USB-LAB - Bestückung

Die Schaltpläne sind als A4-PDF im Download zu finden:

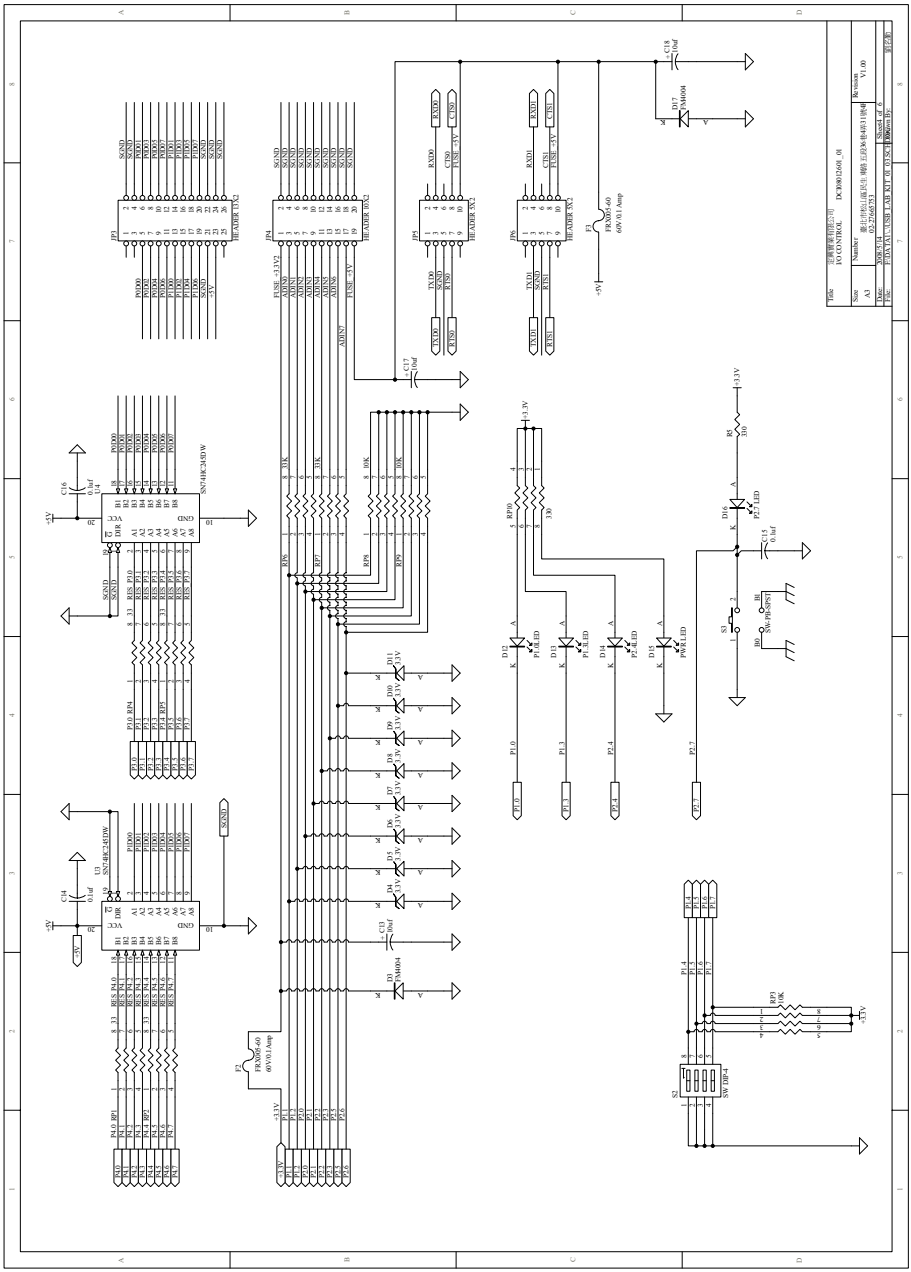
http://www.decision-computer.de/Download/USB/download-lab.html



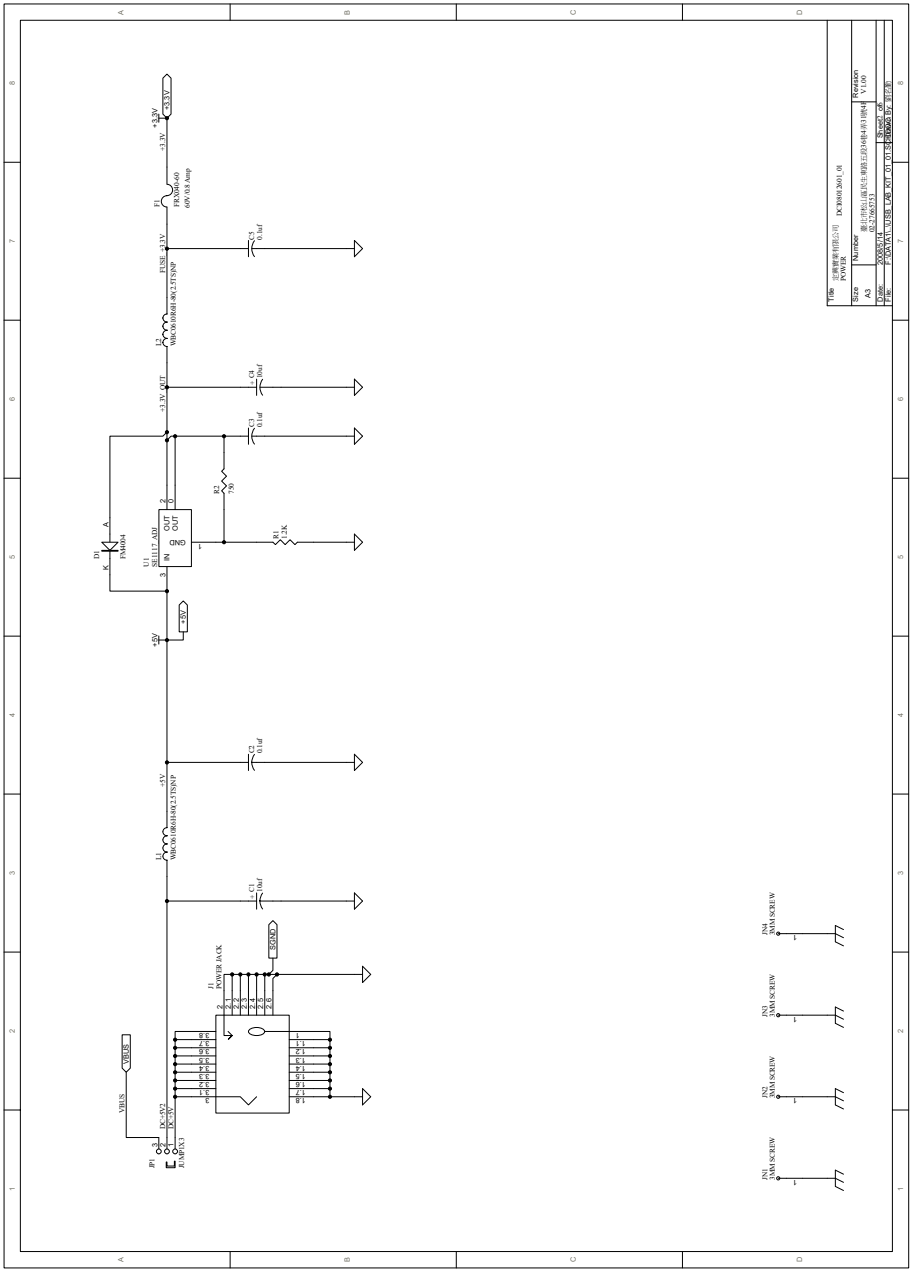
Schaltplan - USB-LAB - Reset-USB



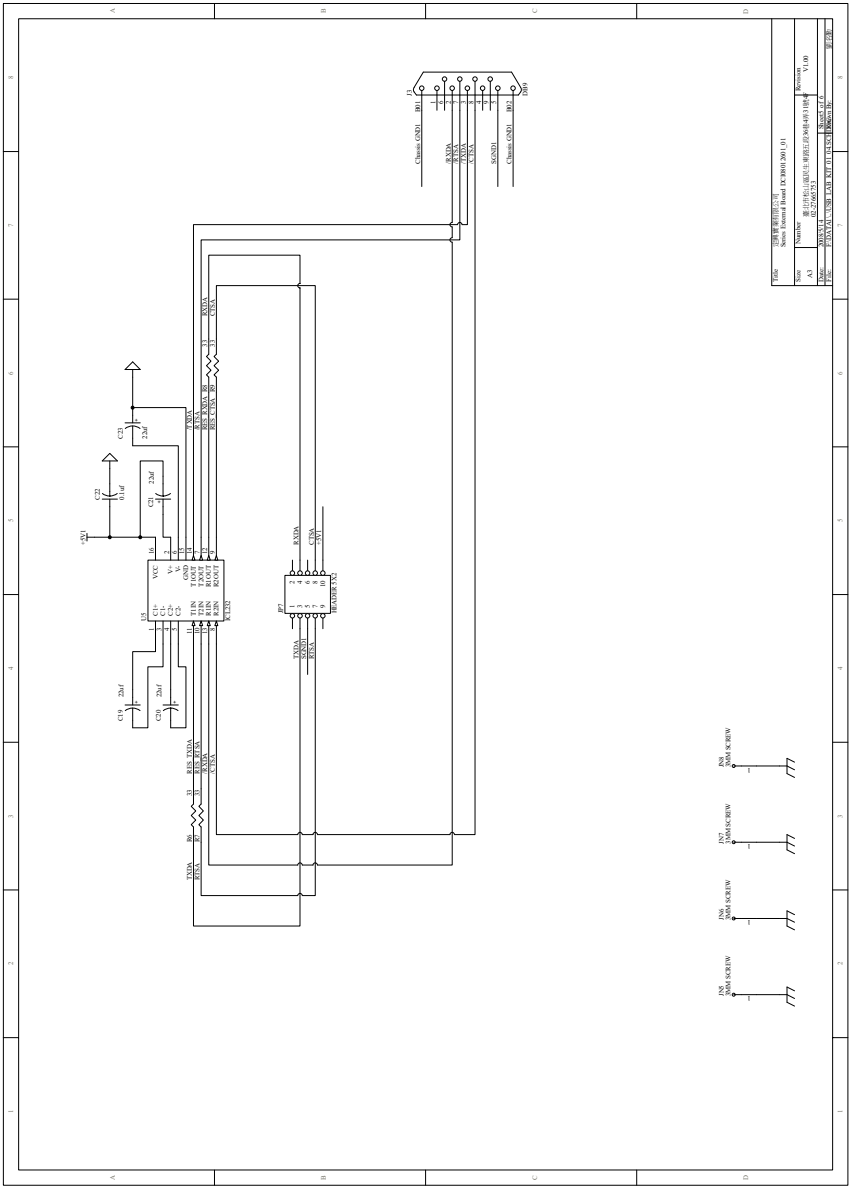
Schaltplan - USB-LAB - ID-DIO-AD

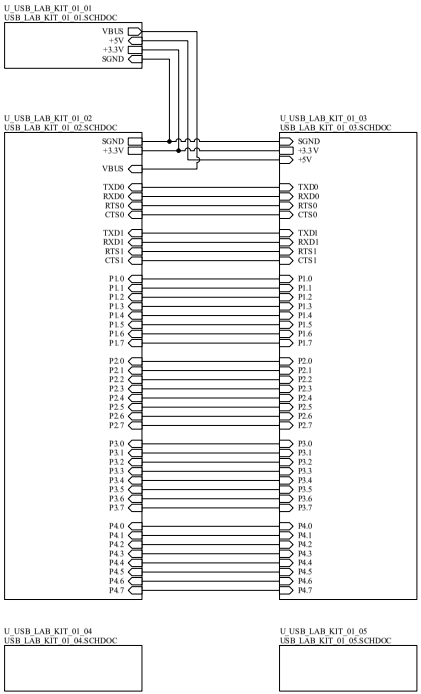


Schaltplan - USB-LAB - Stromversorgung



Schaltplan - USB-LAB - RS232





Title	定興實業有限公司 USB LAB KIT	DCI08012601_01	
Size: B	Number		Revision: 1.00
	臺北市松山區民生東路五段36巷4弄31號4F 02-27665753		
Date:	2008/5/14	Sheet 1of 6	
File:	F:\DATA1\...\USB_LAB_KIT_01.prj	Drawn By:	劉盈勳

CERTIFICATE Certificate

CE

VERIFICATION OF COMPLIANCE

APPLICANT	DESICION GROUP INC.
ADDRESS	4th Floor, No. 31, Alley 4, Lane 36, Sec. 5, Ming-Shen East Road, Taipei Postal code: 10576, Taiwan, R.O.C.
EQUIPMENT	USB Automation I/O board
MODEL NAME	AUSB series
TRADE NAME	
REPORT NO.	WSCE1608014
STANDARD(S)	EMI --- EN 55032 CLASS B: 2012 EN 61000-3-2: 2014 EN 61000-3-3: 2013 EMS --- EN 55024: 2010 IEC 61000-4-2 : 2008 IEC 61000-4-3 : 2006+A1: 2007+A2:2010 IEC 61000-4-4 : 2012 IEC 61000-4-5 : 2014 IEC 61000-4-6 : 2013 IEC 61000-4-8 : 2010 IEC 61000-4-11 : 2004

The above equipment was tested by WEISHANG Certification Co., Ltd. for compliance with the requirements set forth in the EUROPEAN COUNCIL Directive 2014/30/EU and the technical standards mentioned above. The results of testing in this report apply only to the product/system, which was tested. Other similar equipment will not necessarily produce the same results due to production tolerance.

Approved By: Brian

Brian Yu / Manager

Issued Date: SEP. 06, 2016

WEISHANG Certification Corp.
12F.-3, No.27-1, Ln. 169, Kangning St., Xizhi Dist., New Taipei City 221, Taiwan (R.O.C.)

DECLARATION OF CONFORMITY

For the following equipment :

Equipment : USB Automation I/O board

Model Name: AUSB series

Applicant: DESICION GROUP INC.

Address: 4th Floor, No. 31, Alley 4, Lane 36, Sec. 5, Ming-Shen East Road, Taipei Postal code: 10576, Taiwan, R.O.C.

Is herewith confirmed to comply with the requirements set out in the Council Directive on the Approximation of the Laws of the Member States relating to Electromagnetic Compatibility (2014/30/EU). For the evaluation regarding the electromagnetic compatibility, the following standards were applied :

EN 55032 CLASS B: 2012
EN 61000-3-2: 2014
EN 61000-3-3: 2013

EN55024: 2010
IEC 61000-4-2: 2008
IEC 61000-4-3: 2006+A1: 2007+A2:2010
IEC 61000-4-4: 2012
IEC 61000-4-5: 2014
IEC 61000-4-6: 2013
IEC 61000-4-8: 2010
IEC 61000-4-11: 2004

The following manufacturer/importer is responsible for this declaration :

Person responsible for marking this declaration :

Cagen Kan Chang 201609 6

(Place) (Date) (Signature)

DECISION GROUP INC. TAIPEI TAIWAN

A.1 Copyright

Each piece of SmartLab package permits user to use SmartLab only on a single computer, a registered user may use the program on a different computer, but may not use the program on more than one computer at the same time.

Corporate licensing agreements allow duplication and distribution of specific number of copies within the licensed institution. Duplication of multiple copies is not allowed except through execution of a licensing agreement. Welcome call for details.

A.2 Warranty Information

SmartLab warrants that for a period of one year from the date of purchase (unless otherwise specified in the warranty card) that the goods supplied will perform according to the specifications defined in the user manual. Furthermore that the SmartLab product will be supplied free from defects in materials and workmanship and be fully functional under normal usage.

In the event of the failure of a SmartLab product within the specified warranty period, SmartLab will, at its option, replace or repair the item at no additional charge. This limited warranty does not cover damage resulting from incorrect use, electrical interference, accident, or modification of the product.

All goods returned for warranty repair must have the serial number intact. Goods without serial numbers attached will not be covered by the warranty.

The purchaser must pay transportation costs for goods returned. Repaired goods will be dispatched at the expense of SmartLab.

To ensure that your SmartLab product is covered by the warranty provisions, it is necessary that you return the Warranty card.

Under this Limited Warranty, SmartLab's obligations will be limited to repair or replacement only, of goods found to be defective a specified above during the warranty period. SmartLab is not liable to the purchaser for any damages or losses of any kind, through the use of, or inability to use, the SmartLab product.

SmartLab reserves the right to determine what constitutes warranty repair or replacement.

Return Authorization: It is necessary that any returned goods are clearly marked with an RA number that has been issued by SmartLab. Goods returned without this authorization will not be attended to.

USB
Dynamic Industrial Interface
V 2.0.1.9

A Universal
Application Programming Interface
To Data Acquisition Products

Users Manual

Design & Implementation by
Decision Computer International Company

2010/04/20

Contents

1. Introduction

This document provides the USB Dynamic Industrial Interface Specifications, including all function calls, and operating procedures.

Disclaimer:

Decision Computer International Company (DECISION) cannot take responsibility for consequential damages caused by using this software. In no event shall DECISION be liable for any damages whatsoever (including, without limitation, damages for loss of business profits, business interruption, loss of business information, or any other pecuniary loss) arising out of the use of or inability to use this product, even if we have been advised of the possibility of such damages.

Trademark Acknowledgments:
Windows 98, Windows ME, Windows 2000, Windows XP, Windows 7, Visual Basic, Visual C++ are registered trademarks of Microsoft Corporation.

2. Features

The USB Dynamic Industrial Interface (USBDII) was created to provide a standard way to access the functionality provided by all USB data acquisition products. Specifically, the USBDII provides the following features:

Platform-independent

The library is compatible under Windows 98, Windows ME, Windows 2000, windows XP, Vista, and Win7. The compatibility under these operation systems guarantees that programs written for either operating system will work unchanged on the other, even without recompilation.

Abstracts Card Functionality from Card Design

The interface concentrates on a card's functionality and hides the user from having to know specifics about the card design, for example, which port needs to be accessed in order to access specific functionality. All details of the card implementation are hidden from the user.

Multiple Device Support

You could access device by its name or by its information (device type, id index).

Programming Language Independent

The library provides a language independent way to access the USB industrial I/O cards, by using a Dynamic-Link-Library architecture.

3. Device Type Definition

Below are names for device types and its' corresponding defined value:

USB_16PIO	0x01	// USB 16 Channel Photo Input / 16 Channel Photo Output Board
USB_LABKIT	0x02	// USB LABKIT
USB_16PR	0x03	// USB 16 Channel Photo Input / 16 Channel Relay Output Board
USB_STARTER	0x04	// USB STARTER
USB_8PR	0x06	// USB 8 Channel Photo Input / 8 Channel Relay Output Board
USB_4PR	0x07	// USB 4 Channel Photo Input / 4 Channel Relay Output Board
USB_8PI	0x08	// USB 8 Channel Photo Input Board
USB_8RO	0x09	// USB 8 Channel Relay Output Board
USB_16PI	0x0A	// USB 16 Channel Photo Input Board
USB_16RO	0x0B	// USB 16 Channel Relay Output Board
USB_32PI	0x0C	// USB 32 Channel Photo Input Board
USB_32RO	0x0D	// USB 32 Channel Relay Output Board
USB_IND	0x0E	// USB Industry Board
USB_M_4IO	0x10	// USB Mini 4 I/O

Notice : Please use this function to open USB_14ADDA or USB_16ADDA.

4. Data Types of Function calls

Since the USBDII was developed in the C++ language, some data types used may not be present in the programming language you want to use. Please find the following data type conversion table for your convenience:

HANDLE	An opaque 32-bit integer
BYTE	A 8-bit unsigned integer
BOOL	A 32-bit integer, either 0 (FALSE) or 1 (TRUE)
DWORD	A 32-bit unsigned integer
HWND	A 32-bit integer representing a valid handle to a Window
LPTSTR	A 32-bit flat pointer to a zero terminated string
LPBOOL	A 32-bit flat pointer to a variable of type BOOL
LPBYTE	A 32-bit flat pointer to a variable of type BYTE
LPDWORD	A 32-bit flat pointer to a variable of type DWORD

Also note that the DLL employs the Standard Call (Pascal) calling mechanism, which is used for all system. USBDII as well and is compatible with VB, VC, Delphi, .NET, and notice the variable with same type name may have different define in different program language. For example, in Visual Basic 6, the width of Integer is 16 bits and the width of Long is 32 bits, but in Visual Basic. Net, the width of Integer becomes 32 bits and the width of Long becomes 64 bits. If you declare variable with different width from our define, it may cause some run-time error.

5. Functions to open and close Devices

hid_OpenDevice

This function opens a device for further access by USB. Please do not use this function to open USB_14ADDA or USB_16ADDA.

Declaration
HANDLE hid_OpenDevice (DWORD device_type,
DWORD device_id);

Parameters
device_type The type of the device to open.
device_id Device's id on the Board.
For more information, please see “Device Type Table & ID Table” following below.

Return value
A valid handle representing the device, or INVALID_HANDLE_VALUE (-1) if an error occurred. For USB_STARTER, there is no ID selection and device_id = 0

Example
HANDLE hDevice = hid_OpenDevice(Device Type, Device Index); if (hDevice == INVALID_ HANDLE_VALUE)
{
MessageBox (NULL,“Open Failed!“,“Error“,MB_OK);
}

---

hid_CloseDevice

This function closes a device by USB.

Declaration
BOOL hid_CloseDevice (HANDLE hDevice)

Parameters
hDevice A valid device handle.

Return value
TRUE if successful, FALSE otherwise.

Example
hid_CloseDevice(hDevice);

com_OpenDevice

This function opens a device for further access by Serial Port. Please use this function to open USB_14ADDA or USB_16ADDA.

Declaration

```
HANDLE  com_OpenDevice ( DWORD device_type,
                         DWORD device_id,
                         DWORD port_num   );
```

Parameters

device_type	The type of the device to open.
device_id	Device's id on the board. For more information, please see “Device Type Table & ID Table” following below.
port_num	Com Port Num to open.

Return value

A valid handle representing the device, or INVALID_HANDLE_VALUE (-1) if an error occurred.

Example

```
HANDLE hDevice = com_OpenDevice(Device Type, Device Index, 1); if (hDevice == INVALID_
HANDLE_VALUE)
        MessageBox (NULL,“Open Failed!“,“Error“,MB_OK);
```

com_CloseDevice

This function closes a device by Serial Port.

Declaration

BOOL com_CloseDevice(HANDLE hDevice)

Parameters

hDevice A valid device handle.

Return value

TRUE if successful, FALSE otherwise.

Example

com_CloseDevice(hDevice);

Remarks

Please see “Serial_Communication.pdf” to set hardware for serial communication, and USB_LAB-KIT, USB_STARTER, USB_8PR are not supported by serial communication.

Device Type Table

Product	device_type
USB_16PIO	0x01
USB_LABKIT	0x02
USB_16PR	0x03
USB_STARTER	0x04
USB_8PR	0x06
USB_4PR	0x07
USB_8PI	0x08
USB_8RO	0x09
USB_16PI	0x0A
USB_16RO	0x0B
USB_32PI	0x0C
USB_32RO	0x0D
USB_IND	0x0E
USB_M_4IO	0x10

Device ID Table

(Switch Setting on the Device Board)

Switch Setting	device_id
1, 2, 3, 4 OFF	0
2, 3, 4 OFF, 1 ON	1
1, 3, 4 OFF, 2 ON	2
3, 4 OFF, 1, 2 ON	3
1, 2, 4 OFF, 3 ON	4
2, 4 OFF, 1, 3 ON	5
1, 4 OFF, 2, 3 ON	6
4 OFF, 2, 3, 4 ON	7
1, 2, 3 OFF, 4 ON	8
2, 3 OFF, 1, 4 ON	9
1, 3 OFF, 2, 4 ON	10
3 OFF, 1, 2, 4 ON	11
1, 2 OFF, 3, 4 ON	12
2 OFF, 1, 3, 4 ON	13
1 OFF, 2, 3, 4 ON	14
1, 2, 3, 4 ON	Firmware update

6. Functions for digital input/output

hid_SetDigitalByte

This function sets or clears a byte on a digital output line by USB.

Declaration

```
BOOL    hid_SetDigitalByte ( HANDLE hDevice,
                             DWORD dwPort,
                             BYTE byPortState
                             );
```

Parameters

hDevice	A valid device handle, previously obtained from hid_OpenDeviceDevice
dwPort	The index of the port on the card to manipulate. The first port has index 0. For more information, please see “Write Address Table” following below.
byPortState	The new state of the port

Return value

TRUE if successful, FALSE otherwise.
If an error occurred, GetLastError() may return the following values:
ERROR_INVALID_PARAMETER - The handle passed was invalid, or the port number was out of range for the device selected.

Example

```
HANDLE hDevice = hid_OpenDevice(0x01,0);
if (hDevice != INVALID_HANDLE_VALUE)
{
hid_SetDigitalByte( hDevice, 0, 0xFF); // set’s all bits on the first port
hid_CloseDevice(hDevice);
}
```

com_SetDigitalByte

This function sets or clears a byte on a digital output line by Serial Port.

Declaration

```
BOOL    com_SetDigitalByte ( HANDLE hDevice,
                             DWORD dwPort,
                             BYTE byPortState
                             );
```

Parameters

hDevice	A valid device handle, previously obtained from com_OpenDevice
dwPort	The index of the port on the card to manipulate. The first port has index 0. For more information, please see “Write Address Table” following below.
byPortState	The new state of the port

Return value

TRUE if successful, FALSE otherwise.
If an error occurred, GetLastError() may return the following values:
ERROR_INVALID_PARAMETER - The handle passed was invalid, or the port number was out of range for the device selected.

Example

```
HANDLE hDevice = com_OpenDevice(0x01,0);
if (hDevice != INVALID_HANDLE_VALUE)
{
com_SetDigitalByte( hDevice, 0, 0xFF); // set’s all bits on the first port
com_CloseDevice(hDevice);
}
```

Remarks

Please see “Serial_Communication.pdf” to set hardware for serial communication, and USB_LAB-KIT, USB_STARTER, USB_8PR are not supported by serial communication.

Write Address Table

Product	dwPort	Content
USB_16PIO	0x02	OUT07 to OUT00
	0x03	OUT15 to OUT08
USB_LABKIT	0x03	P1D07 to P1D00
USB_STARTER	0x03	P1D07 to P1D00
USB_16PR	0x02	OUT07 to OUT00
	0x03	OUT15 to OUT08
USB_8PR	0x01	OUT07 to OUT00
	0x02	DIO7 to DIO0
	0x03	DIO15 to DIO8
USB_4PR	0x02	OUT03 to OUT00
USB_8RO	0x02	OUT07 to OUT00
USB_16RO	0x02	OUT07 to OUT00
	0x03	OUT15 to OUT08
USB_32RO	0x00	OUT07 to OUT00
	0x01	OUT15 to OUT08
	0x02	OUT23 to OUT16
	0x03	OUT31 to OUT24
USB_IND	0x00	Port 0
	0x01	Port 1
	0x02	Port 2
	0x03	Port 3
	0x04	Port 4
	0x05	Port 5
	0x06	Port 6
	0x07	Port 7
	0x08	DIO
	0x0D	IOCONFIG
USB_M_4IO	0x02	OUT03 to OUT00

hid_GetDigitalByte

This function reads a complete byte from a digital input port of a device by USB.

Declaration

```
BOOL    hid_GetDigitalByte ( HANDLE hDevice,
                             DWORD dwPort,
                             LPBYTE lpbyPortState
                             );
```

Parameters

hDevice	A valid device handle, previously obtained from hid_OpenDeviceDevice
dwPort T	he index of the port on the card to manipulate. The first port has index 0. For more information, please see “Read Address Table” following below.
lpbyPortState	A pointer to a variable of type BYTE receiving the new state of the port

Return value

TRUE if successful, FALSE otherwise.
If an error occurred, GetLastError() may return the following values:
ERROR_INVALID_PARAMETER – The handle passed was invalid, or the port number was out of range for the device selected.

Example

```
HANDLE hDevice = hid_OpenDevice(0x01,0); if (hDevice != INVALID_HANDLE_VALUE)
{
hid_GetDigitalByte( hDevice, 0, &byState); // reads the state of the first input port hid_
CloseDevice(hDevice);

}
```

com_GetDigitalByte

This function reads a complete byte from a digital input port of a device by Serial Port.

Declaration

```
BOOL    com_GetDigitalByte ( HANDLE hDevice,
                             DWORD dwPort,
                             LPBYTE lpbyPortState
                             );
```

Parameters

hDevice	A valid device handle, previously obtained from com_OpenDevice
dwPort	The index of the port on the card to manipulate. The first port has index 0. For more information, please see “Read Address Table” following below.
lpbyPortState	A pointer to a variable of type BYTE receiving the new state of the port

Return value

TRUE if successful, FALSE otherwise.
If an error occurred, GetLastError() may return the following values:
ERROR_INVALID_PARAMETER – The handle passed was invalid, or the port number was out of range for the device selected.

Example

```
HANDLE hDevice = com_OpenDevice(0x01,0);
if (hDevice != INVALID_HANDLE_VALUE)
{
com_GetDigitalByte( hDevice, 0, &byState); // reads the state of the first input port
com_CloseDevice(hDevice);
}
```

Remarks

Please see “Serial_Communication.pdf” to set hardware for serial communication, and USB_LAB-KIT, USB_STARTER, USB_8PR are not supported by serial communication.

Read Address Table

Product	dwPort	Content
USB_16PIO	0x00	IN07 to IN00
	0x01	IN15 to IN08
USB_LABKIT	0x02	P0D07 to P0D00
USB_STARTER	0x02	P0D07 to P0D00
USB_16PR	0x00	IN07 to IN00
	0x01	IN15 to IN08
USB_8PR	0x00	IN07 to IN00
	0x02	DIO7 to DIO0
	0x03	DIO15 to DIO8
	0x10	JP9/JP10 Settings
USB_4PR	0x00	IN03 to IN00
USB_8PI	0x00	IN07 to IN00
USB_16PI	0x00	IN07 to IN00
	0x01	IN15 to IN08
USB_32PI	0x00	IN07 to IN00
	0x01	IN15 to IN08
	0x02	IN23 to IN16
	0x03	IN31 to IN24
USB_IND	0x00	Port 0
	0x01	Port 1
	0x02	Port 2
	0x03	Port 3
	0x04	Port 4
	0x05	Port 5
	0x06	Port 6
	0x07	Port 7
	0x08	DIO
	0x0D	IOCONFIG

	0x10	Port 0 default value
	0x11	Port 1 default value
	0x12	Port 2 default value
	0x13	Port 3 default value
	0x14	Port 4 default value
	0x15	Port 5 default value
	0x16	Port 6 default value
	0x17	Port 7 default value
	0x18	Port DIO default value
	0x19	Input/output default setting
USB_M_4IO	0x00	IN03 to IN00

Remarks

In USB_8PR, we provide 2 digital ports for user to define either as input or output. It can be defined by Jumper 10 and Jumper 11 on the board. And we can use hid_GetDigitalByte / com_GetDigitalByte function to read Jumper State to determine witch port is either input or output.

hid_GetDigitalByte(hDevice, 0x10, &byState); // or use com_GetDigitalByte for serial communication

When JP9 is closed, DIO7 - DIO0 is for Input. The fifth bit of byState is 0
When JP9 is opened, DIO7 - DIO0 is for Output. The fifth bit of byState is 1
When JP10 is closed, DIO15 – DIO8 is for Input. The sixth bit of byState is 0
When JP10 is opened, DIO15 – DIO8 is for Output. The sixth bit of byState is 1

7. Functions for reset hardware device

hid_ResetHW

This function directly resets the hardware device by USB. And all channels on the board will load default value. If you need to control the device again, please use hid_open to get the handle again.

Declaration
BOOL hid_ResetHW(HANDLE hDevice)

Parameters
hDevice A valid device handle.

Return value
TRUE if successful, FALSE otherwise.

Example

```
hid_ResetHW (hDevice);
```

---

com_ResetHW

This function directly resets the hardware device by Serial Port. And all channels on the board will load default value.

Declaration
BOOL com_ResetHW(HANDLE hDevice)

Parameters
hDevice A valid device handle.

Return value
TRUE if successful, FALSE otherwise.

Example

```
com_ResetHW(hDevice);
```

8. Functions for analog input/output

hid_GetAnalogChannel

This function reads a complete word from an analog input port of a device by USB.

Declaration

```
BOOL    hid_GetAnalogChannel ( HANDLE hDevice,
                               DWORD dwPort,
                               LPDWORD lpdwPortState
                               );
```

Parameters

hDevice	A valid device handle, previously obtained from hid_OpenDeviceDevice dw-
Port	The index of the port on the card to manipulate. The first port has index 0.
lpdwPortState	A pointer to a variable of type DWORD receiving the new state of the port

Return value

TRUE if successful, FALSE otherwise.
If an error occurred, GetLastError() may return the following values:
ERROR_INVALID_PARAMETER - The handle passed was invalid, or the port number was out of range for the device selected.

Example

```
HANDLE hDevice = hid_OpenDevice(0x02,0); // USB_LABKIT
if (hDevice != INVALID_HANDLE_VALUE)
{
hid_GetAnalogChannel ( hDevice, 0, &dwState); // reads the state of the first analog input port
hid_CloseDevice (hDevice);
}
```

Remarks

This function now only enable in USB_LABKIT and USB_STARTER device. The range of dwPort is from 0~7.

com_GetADHex

This function reads a complete word in hex from an analog input port of a device by USB.

Declaration

```
BOOL com_GetADHex(HANDLE hDevice,
                  UINT dwPort,
                  UINT *lpdwValue
                  );
```

Parameters

hDevice	A valid device handle, previously obtained from com_OpenDevice
dwPort	The index of the port on the card to manipulate. The first port has index 0.
lpdwValue	A pointer to a variable of type UINT receiving the new state of the port

Return value

TRUE if successful, FALSE otherwise.
If an error occurred, GetLastError() may return the following values:
ERROR_INVALID_PARAMETER - The handle passed was invalid, or the port number was out of range for the device selected.

Example

```
HANDLE hDevice = com_OpenDevice(card_id,card_number,10);
if (hDevice != INVALID_HANDLE_VALUE)
{
com_GetAnalogChannel ( hDevice, 0, &dwState); // reads the state of the first analog input port
com_CloseDevice (hDevice);
}
```

Remarks

This function now only enable in USB_14ADDA and USB_16ADDA device. The range of dwPort

com_GetADMilli

This function reads the result in decimal millivolt from an analog input port of a device by USB.

Declaration

```
BOOL com_GetADMilli (HANDLE hDevice,
                     UINT dwPort,
                     LONG *lpdwValue
                     );
```

Parameters

hDevice — A valid device handle, previously obtained from com_OpenDevice
dwPort — The index of the port on the card to manipulate. The first port has index 0.
lpdwValue — A pointer to a variable of type singed 32-bit integer receiving the new state of the port

Return value

TRUE if successful, FALSE otherwise.
If an error occurred, GetLastError() may return the following values:
ERROR_INVALID_PARAMETER - The handle passed was invalid, or the port number was out of range for the device selected.

Example

```
HANDLE hDevice = com_OpenDevice(card_id,card_number,10);
if (hDevice != INVALID_HANDLE_VALUE)
{
com_GetADMilli ( hDevice, 0, &dwState); // reads the state of the first analog input port
com_CloseDevice (hDevice);
}
```

Remarks

This function now only enable in USB_14ADDA and USB_16ADDA device. The range of dwPort is from 0~15.

com_GetADMicro

This function reads the result in decimal microvolt from an analog input port of a device by USB.

Declaration

```
BOOL com_GetADMicro (HANDLE hDevice,
                     UINT dwPort,
                     Long *lpValue
                     );
```

Parameters

hDevice A valid device handle, previously obtained from com_OpenDevice
dwPort The index of the port on the card to manipulate. The first port has index 0.
lpValue A pointer to a variable of type singed 32-bit integer receiving the new state of the port

Return value

TRUE if successful, FALSE otherwise.
If an error occurred, GetLastError() may return the following values:
ERROR_INVALID_PARAMETER - The handle passed was invalid, or the port number was out of range for the device selected.

Example

```
HANDLE hDevice = com_OpenDevice(card_id,card_number,10);
 if (hDevice != INVALID_HANDLE_VALUE)
{
com_GetADMicro ( hDevice, 0, &dwState); // reads the state of the first analog input port
com_CloseDevice (hDevice);
}
```

Remarks

This function now only enable in USB_14ADDA and USB_16ADDA device. The range of dwPort is from 0~15

com_SetDAHex

This function writes a complete word in hex to an analog output port of a device by USB.

Declaration

```
BOOL com_SetDAHex(HANDLE hDevice,
                  UINT dwPort,
                  UINT dwValue
                  );
```

Parameters

hDevice	A valid device handle, previously obtained from hid_OpenDeviceDevice
dwPort	The index of the port on the card to manipulate. The first port has index 0.
dwValue	An unsigned hexical value to assign new value to DA channel

Return value

TRUE if successful, FALSE otherwise.
If an error occurred, GetLastError() may return the following values:
ERROR_INVALID_PARAMETER - The handle passed was invalid, or the port number was out of range for the device selected.

Example

```
HANDLE hDevice = com_OpenDevice(card_id,card_number,10);
if (hDevice != INVALID_HANDLE_VALUE)
{
com_SetDAHEX ( hDevice, 0, dwState); // writes the state to the first analog output port
com_CloseDevice (hDevice);
}
```

Remarks

This function now only enable in USB_14ADDA and USB_16ADDA device. The range of dwPort is from 0~15.

com_SetDAMilli

This function writes a singed decimal value in millivolt to an analog output port of a device by USB.

Declaration

```
BOOL com_SetDAMilli(HANDLE hDevice,
                    UINT dwPort,
                    LONG lnValue
                    );
```

Parameters

hDevice	A valid device handle, previously obtained from com_OpenDevice
dwPort	The index of the port on the card to manipulate. The first port has index 0.
lnValue	An signed decimal value to assign new value to DA channel

Return value

TRUE if successful, FALSE otherwise.
If an error occurred, GetLastError() may return the following values:
ERROR_INVALID_PARAMETER - The handle passed was invalid, or the port number was out of range for the device selected.

Example

```
HANDLE hDevice = com_OpenDevice(card_id,card_number,10);
if (hDevice != INVALID_HANDLE_VALUE)
{
com_SetDAMilli ( hDevice, 0, dwState); // writes the state to the first analog output port
com_CloseDevice (hDevice);
}
```

Remarks

This function now only enable in USB_14ADDA and USB_16ADDA device. The range of dwPort is from 0~15.

com_SetDAMicro

This function writes a singed decimal value in microvolt to an analog output port of a device by USB.

Declaration

```
BOOL com_GetADHex(HANDLE hDevice,
                  UINT dwPort,
                  LONG lnValue
                  );
```

Parameters

hDevice	A valid device handle, previously obtained from hid_OpenDeviceDevice
dwPort	The index of the port on the card to manipulate. The first port has index 0.
lnValue	An signed decimal value to assign new value to DA channel

Return value

TRUE if successful, FALSE otherwise.
If an error occurred, GetLastError() may return the following values:
ERROR_INVALID_PARAMETER - The handle passed was invalid, or the port number was out of range for the device selected.

Example

```
HANDLE hDevice = com_OpenDevice(card_id,card_number,10);
if (hDevice != INVALID_HANDLE_VALUE)
{
com_SetDAMicro ( hDevice, 0, dwState); // writes the state to the first analog output port
com_CloseDevice (hDevice);
}
```

Remarks

This function now only enable in USB_14ADDA and USB_16ADDA device. The range of dwPort is from 0~15.

9. Functions for Watch dog

hid_SetWD

This function sets time interval for Watch Dog.

Declaration

```
BOOL    hid_SetWD( HANDLE hDevice,
                   BYTE byMode      );
```

Parameters

hDevice A valid device handle, previously obtained from hid_OpenDeviceDevice
byMode Time interval for Watch Dog (Value 1~5 as 1/5/10/30/60 seconds, default as 10s)

Return value

TRUE if successful, FALSE otherwise.
If an error occurred, GetLastError() may return the following values:
ERROR_INVALID_PARAMETER - The handle passed was invalid, or the port number was out of range for the device selected.

---

hid_EnableWD

This function enables/disables Watch Dog.

Declaration

```
BOOL    hid_EnableWD( HANDLE hDevice,
                      BOOL bEnabled );
```

Parameters

hDevice	A valid device handle, previously obtained from hid_OpenDeviceDevice
bEnabled	Enable/disable watch dog.

Return value

TRUE if successful, FALSE otherwise.
If an error occurred, GetLastError() may return the following values:
ERROR_INVALID_PARAMETER - The handle passed was invalid, or the port number was out of range for the device selected.

10. Using the Dynamic Industrial Interface with different programming languages

This chapter provides an overview about how to best utilize the Dynamic Industrial Interface in various programming languages.

If you experience difficulties calling the Dynamic Industrial Interface functions from your programming language, or are using a programming language not covered in this documentation, please feel free to visit our web-site, to which we will post updated information regarding DII programming issues. You may also contact our technical support through our website: www.decision.com.tw

10.1. C++

Since the DII DLL was developed using C++, you may easily use it to access Industrial I/O devices. For this purpose, a C++ header file ("USBDII.h") as well as an import library ("USBDII.lib") are being shipped with the interface library. Make sure that you have installed the development release, not the retail release, which does not include support programming files. In your C/C++ source code files, just include the "USBDII.h" include file, then you can use any of the functions provided by the USBDII DLL. Be sure to include the import library "USBDII.lib" during the linking step of your application. So your applications successfully references the actual interface DLL.

10.2. Visual Basic

Since the Dynamic Industrial Interface is fully 32-bit compliant, only 32-bit versions of Visual Basic are supported. Specifically, Version 6.0 are tested and supported. If you are using Visual Basic to access any I/O Devices supported by the USB Dynamic Industrial Interface (USBDII), you can call the USBDII DLL directly. But before that, you should import them. You may also consult the Visual Basic sample application for more information about using Visual Basic to access the USB Dynamic Industrial Interface (USBDII).

11. Technical Support and Feedback

We believe that customer input is the most valuable source for creating successful products. We continuously update and extend the Dynamic Industrial Interface with new functionality, for specific devices, for specific applications, to meet your specific needs, and provide supportive products around the USBDII.

You may also contact our technical support through our website: www.decision.com.tw

12. Release notes

2015/02/17
Version 2.0.1.9
Fix multiple cards open for USB_M_4IO Version 2.0.1.8
Fix slow open speed for USB_M_4IO Version 2.0.1.7
Add support for USB_M_4IO

2012/11/09
Version 2.0.1.6
x64 version released

2011/11/17
Version 2.0.1.3
Release analog input/output functions for virtual com port.

2011/11/16
Version 2.0.1.2
Remove address checking
Fix the problem of hid_GetDigitalByte can not read some address of USBIND.
Provide default value read back function for USBIND.

2011/11/3
Version 2.0.1.1
Fix address limitations for USB Industry.

2010/04/20
Version 2.0.1.0
Update for supporting USB Industry.