USB2I2C
USB2SPI
USB2ISP

DATASHEET
V0.1E

USBIO TECH.
A UKOREL COMPANY

Universal USB-Interface-Chip

USB2I2C/USB2SPI/USB2ISP

Update：2012-5-19

Features and General Description：

- ✓ USB to EPP 8bit parallel Port;
- ✓ USB to MEM 8bit parallel Port;
- ✓ USB to SPI(Master) BUS;
- ✓ USB to I2C/IIC/TWI/SMBUS(Master) BUS.

For more information：

Tech：http://www.usb-i2c-spi.com/cn

Sales：http://www.usb-i2c-spi.com/cn/mail.htm

INDEX

1、FEATURES

Features ：

USB to EPP 8bit parallel Port;

USB to MEM 8bit parallel Port;

USB to SPI(Master) BUS;

USB to I2C/IIC/TWI/SMBUS(Master) BUS.

The USB2I2C/USB2SPI/USB2ISPmakes it possible to interface different kind of user applications to the PC over USB, without any need of µC controller programming. The Chip can be used as I²C interface to i²C enabled peripheral IC´s. It also offers an 8 bit bi-directional data bus with EPP/MEM protocol, which could be used to interface HD44789 compatible text LCD displays for example.

Programming is made easy using an API-DLL, suitable for almost any programming language (Delphi, CBC, C, VB, LABVIEW ...).

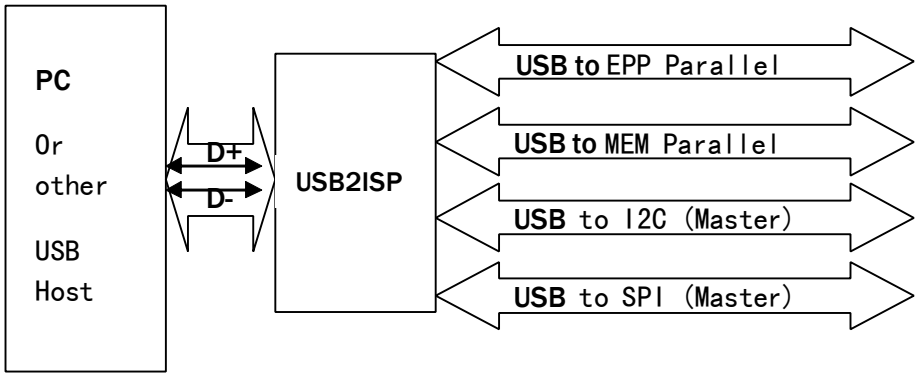


FIG 1- USB2ISP FUNCTION

2、GENERAL DESCRIPTION

2.1.GENERAL DESCRIPTION

Full-speed USB device interface, compatible with USB V2.0 external components only need a 12M crystal and 2 capacitors.

Low-cost, direct conversion of the original parallel port peripherals, the original SPI interface peripheral devices, I2C interface peripherals.

USB2ISP the Collection USB2I2C and USB2SPI design only need USB to I2C function please use USB2I2C (SSOP20 package); please use USB2SPI (SSOP20 package) If you only use USB to SPI function.

In parallel port mode, USB2ISP EPP mode or the MEM mode of monitoring active parallel interface, direct data input and output between the PC host computer and the next bit controller, eliminating the need for microcontroller / DSP / MCU.

Synchronous serial mode, USB2ISP chip supports 4-wire SPI synchronous serial port, three CS lines (CS0, CS1, CS2), SCK line, DIN (MISO) line and the DOUT (MOSI) line, very easy to realized the SPI class interface of the device to read and write the; USB2ISP chip also supports the wire I2C/IIC/TWI/SMBUS synchronous serial port, the SCL line and SDA line PC host computer can easily read and write the I2C/IIC/TWI/SMBUS interface device .

2.2.PARALLEL PORT

Providing two interface mode:EPP and MEM.

EPP mode supplies AS#,DS# and WR# etc signal,similar with EPP V1.7 or EPP V1.9.

MEM mode supplies A0,RD# and WR# etc signal,similar with memory read/write mode.

2.3.SYNCHRONOUS SERIAL

The 2 wire I2C/IIC/TWI/SMBUS interface, supports SCL at：20KHz/100KHz/400KHz/750KHz.

The 4-wire SPI synchronous serial interface, 750KHz fixed SCK clock frequency.

3、 PACKAGE

3.1.PINS

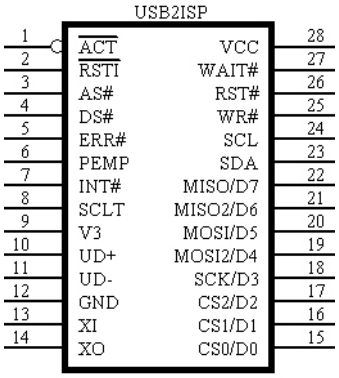


FIG. 2 - USB2ISP

3.2.FOOTPRINT

Package shape	Width of plastic		Pitch of Pin		Thickness		Instruction of package	订货型号
SOP-28	7.62mm	300mil	1.27mm	50mil	2.54mm	100mil	Small outline package of 28-pin	USB2ISP

4、PINS

4.1.GENERAL EXPLANATION

The detail function of USB2I2C/USB2SPI/USB2ISP is decided by function configuration after reset. The same pin may have different define under different function.

4.2.STANDARD PUBLIC PINS

Pin No.	Pin Name	Pin Type	Pin Description
28	VCC	POWER	Positive power input port, requires an 0.1uF power decoupling capacitance
12	GND	POWER	Public ground, ground connection for USB bus
9	V3	POWER	Attachment of VCC input external power while 3.3V;connects of 0.01uF decoupling capacitance outside while 5V
13	XI	IN	Input of crystal oscillator, attachment of crystal and crystal oscillator capacitance external
14	XO	OUT	Opposite output of crystal oscillator, attachment of crystal and crystal oscillator capacitance outside
10	UD+	USB signal	Directly connects to D+ data wire of USB bus
11	UD-	USB signal	Directly connects to D- data wire of USB bus
1	ACT#	IN	By 2K resistor must be pulled down to the ground
2	RSTI	IN	Input of external reset, active with high-level, with pull-down resistor

4.3.PARALLEL MODE PINS

Pin No.	Pin Name	Pin Type	Pin Description
22～15	D7～D0	Tri-state Output	8-bit bi-directional data bus, with pull-up resistor
25	WR#	OUT	EPP mode: indicate write operation, write with low-level, read with high-level
			MEM mode: write strobe output WR#, low-level active
4	DS#	OUT	EPP mode: data operation select, low-level active
			MEM mode: read strobe output RD#, low-level active
26	RST#	OUT	Reset output, low-level active
3	AS#	Tri-state Output	EPP mode: address operation strobe, low-level active
			MEM mode: address wire output ADDR or A0
27	WAIT#	IN	USB2I2C/USB2PSI/USB2ISP request to wait, low-level active, with pull-up resistor
7	INT#	IN	Interrupt request input, active with rising edge, with pull-up resistor
5	ERR#	IN	Self-define common input, with pull-up resistor
8	SLCT	IN	Self-define common input, with pull-up resistor
6	PEMP	IN	Self-define common input, with pull-up resistor

4.4.SPI/I2C SERIAL INTERFACE PINS

Pin No.	Pin Name	Pin Type	Pin Description
22	DIN（MISO）	IN	4-wire serial data input, with pull-up resistor
21	DIN2（MISO2）	IN	5-wire serial data input 2,with pull-up resistor
20	DOUT（MOSI）	Tri-state Output	4-wire serial data output, other name is MOSI or SDO
19	DOUT2（MOSI2）	Tri-state Output	5-wire serial data output 2, other name is MOSI2 or SDO2
18	SCK	Tri-state Output	4-wire/5-wire serial interface clock output, other name is SCK
17	CS2	Tri-state Output	4-wire serial interface chip select output 2#
16	CS1	Tri-state Output	4-wire serial interface chip select output 1#
15	CS0	Tri-state Output	4-wire serial interface chip select output 0#
24	SCL	Drain open Output	2-wire serial interface clock output, with pull-up resistor
23	SDA	Drain open Output/IN	2-wire serial interface data input/output, with pull-up resistor
26	RST#	OUT	Reset output, low-level active
7	INT#	IN	Interrupt request input, active with rising edge, with pull-up resistor

5、HARDWARE APPLICATION

5.1. BASIC CONNECTION

P3 is USB endpoint, USB bus contains a double 5V source wires and a double data signal wires. Usually, the +5V source wire is red while connects to ground wire is black. D+ signal wire is green, D- signal wire is white. The source current is up to 500mA supplied by USB bus. Generally, USB2I2C/USB2SPI/USB2ISP and other low-cost USB productions can use 5V source directly. If USB productions supply common power by other mode, USB2I2C/USB2SPI/USB2ISPneed to use the common power. If these productions use other common power and USB bus power at the same time, connects 5V power wire of USB bus to 5V common power of USB productions via 1Ω resistance. And join ground wire of the two power devices.

C13 and C14 are monolithic or high frequency ceramic capacitances. The capacity of C13 varies from 4700pF to 0.02uF, eliminates the coupling of inner power of USB2I2C/USB2SPI/USB2ISP. The capacity of C14 is 0.1uF, eliminates the coupling of external power. The crystal X3、capacitance C11 and C12 are composed of clock oscillating circuit of USB2I2C/USB2SPI/USB2ISP. Frequency of X3 is 12MHz. C11 and C12 are monolithic or high frequency with capacity of 15pF~30pF capacitances.

When designing the PCB, pay much attention to some notes: decoupling capacitance C13 and C14 must keep near to connection pin of USB2I2C/USB2SPI/USB2ISP; makes sure D+ and D- are parallel and supply ground or covering copper besides to decrease the disturb from outside signal; the relevant signal leads between XI and XO must be kept as short as possible. In order to lessen the high frequency clock disturb outside, setting ground wire on the circle or covering copper to the relative equipments.

R1 is configured resistance, ACT pin must be pulled down to the ground by 2K ohm resistor.

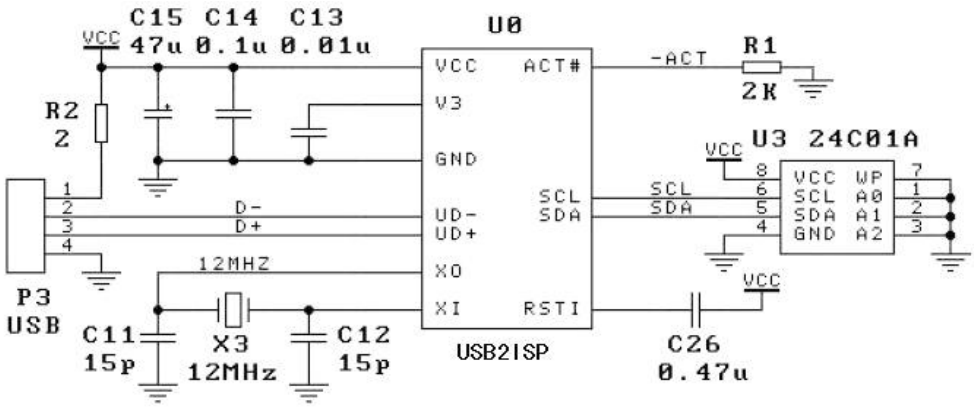
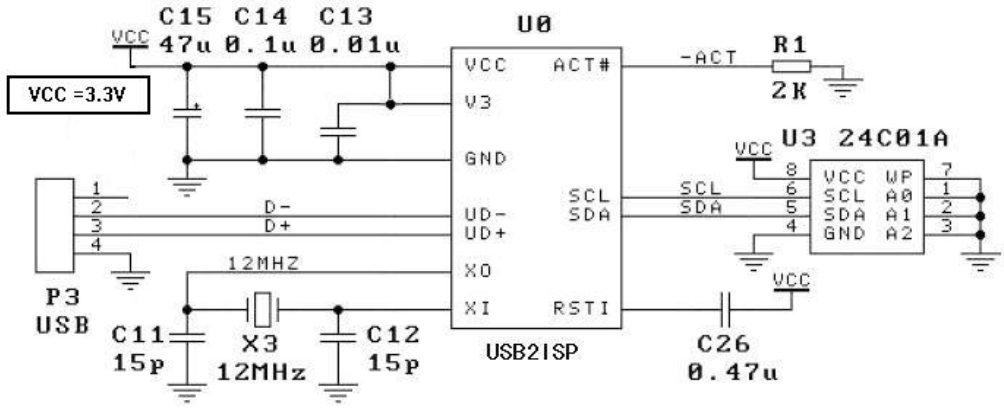


FIG. 5 - USB2ISP BASIC CONNECTION(5V AND 3.3V)



5.2 PID/VID

USB2I2C/USB2SPI/USB2ISP：VID = 4348，PID = 5512。

5.3 I2C/IIC/TWI/SMBUS BUS

The USB2I2C/USB2ISP generates bus clock rates of 20KHz/100KHz/400KHz/750KHz and operates as bus master in any case. Slave operation is not supported.

API functions

Function USBIO_StreamI2C(...) Read/Write I²C

Function USBIO_ReadI2C(...) reads from chips with internal memory or register address.

Function USBIO_WriteI2C(...) writes to chips with internal memory or register address.

Function USBIO_ReadEEPROM(...) reads I²C-EEPROM data

Function USBIO_WriteEEPROM(...) writes I²-EEPROM data

5.4 SPI BUS

The chip operates as SPI master. Slave operation is not possible. API functions exchange data over a buffer, that needs to be filled with send data before function is called. After API function returns, the buffer contains the bytes read. This means that number of bytes read and bytes written is equal.

Pin	Name	Function	Remark
22	DIN	Input	Data IN
21	DIN2	Input	Data IN2
20	DOUT	Output	Data OUT
19	DOUT2	Output	Data OUT2
18	DCK	Output	Data Clock
17	CS2	Output	CS2
16	CS1	Output	CS1
15	CS0	Output	CS0
24	SCL	Output	SCL
23	SDA	Output	SDA
26	RST#	Output	Reset
7	INT#	Input	Interrupt request (rising edge)
5,8,6		Input	Reserved

5.4.1 STANDARD-SPI (MODE 0)

AP function: USBIO_StreamSPI4

Standard SPI (Mode 0) with two uni-directional data lines, clock and chip select. (Atmel, etc.) This is the most commonly used SPI mode. Data is read from input pin 22 (DIN; DATA IN ; MISO) and written to output Pin 20 (DOUT; DATA OUT; MOSI). Pin 18 (DCK; DATA CLOCK; SCKL) offers the clock signal. Outputs CSx (Chip Select; Slave Select) activate the selected chip. Serial transfer of data may start with (MSB or LSB) first.

5.4.2 3-WIRE SPI

API function: USBIO_StreamSPI3

Data transfer is done with a single, bi-directional data line. A rising clock edge writes data, a falling clock edge reads data. This kind of protocol can be found at MAXIM chip DS1626 for example.

5.4.3 SPI WITH DUAL DATA LINES

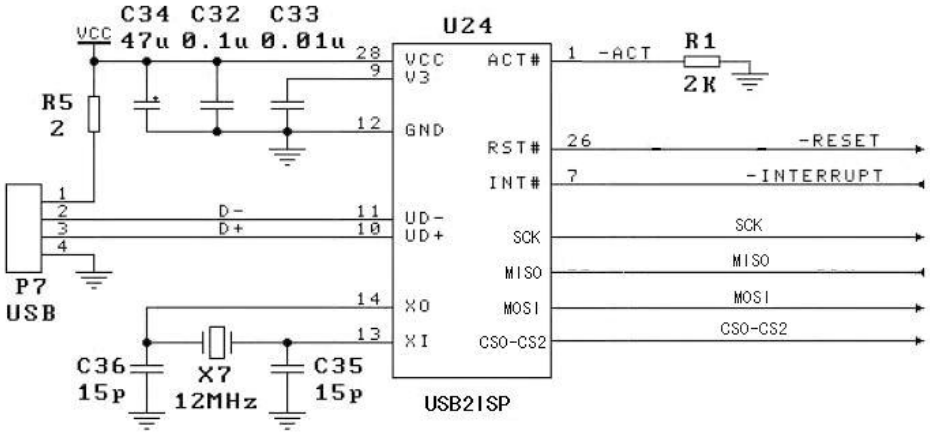
API function: USBIO_StreamSPI5

This function uses two pairs of uni-directional data lines. A data byte is split into upper and lower half. Upper nibble is transferred over DIN/DOUT, while lower nibble is transferred over DIN2/DOUT2. A concrete example for this mode could not be found.

5.4.4 BIT MODE SPI

API function: USBIO_BitStreamSPI

This function can generate an individual serial signal pattern. A LTC1257 D/A converter for example uses a 12 bit serial transfer with final LOAD signal. For such cases the function offers 6 serial output channels and 2 serial input channels. These eight channels are assigned to the bits 0..7 of the in coming and out going data bytes. Bit 6 and bit 7 contain read data from data inputs D6 and D7. Output D3 offers the clock signal. Output pattern for D0...D2, D4 and D5 is generated from the bits 0...2, 4 and 5 of the data bytes shipped to the API call. As the function does not set the data direction of the data lines, D0..D5 must be programmed as output before, using the corresponding function USBIO_SetOutput().



5.5 GPIO: PROGRAMMING I/O INDIVIDUALLY

Each I/O pin of the USB2ISP is assigned to a certain bit in the parameters Enable, SetDirOut and SetDataOut of the API functions listed below.

15 bi-directional pins (with programmable data direction; input or output):

Bit 0-7	D0-D7	pin 15-22
Bit 8	ERR#	pin 5
Bit 9	PEMP	pin 6
Bit 10	ACK	pin 7
Bit 11	SLCT	pin 8
Bit 12	-	-
Bit 13	BUSY/WAIT#	pin 27
Bit 14	AUTOFD#/DATAS#	pin 4
Bit 15	SLCTIN#/ADDRS#	pin 3

One pseudo-bi-directional pin (Combination of input and open-collector-output) :

Bit 23	SDA	pin 23 (functions GetInput/GetStatus)
Bit 16	SDA	pin 23 (function SetOutput)

Three uni-directional pins (output only):

Bit 16	RESET#	pin 26
Bit 17	WRITE#	pin 25
Bit 18	SCL pin	pin 24

API functions

USBIO_SetOutput

- Sets data direction (input/output) of bi-directional pins,

- Sets output state HIGH/LOW

USBIO_Set_D5_D0

- Sets data direction of pins D0...D5

- Sets output state of pins D0...D5

USBIO_GetInput und USBIO_GetStatus

- Reads input states

Parameter Enable

- Prohibits or allows changes of data direction

- Bit set = changes allowed

- Bit clear = changes are ignored

Parameter SetDirOut:

- Sets data direction: Input/Output

- Bit set = Pin operates as output (Caution!)

- Bit clear = Pin operates as input

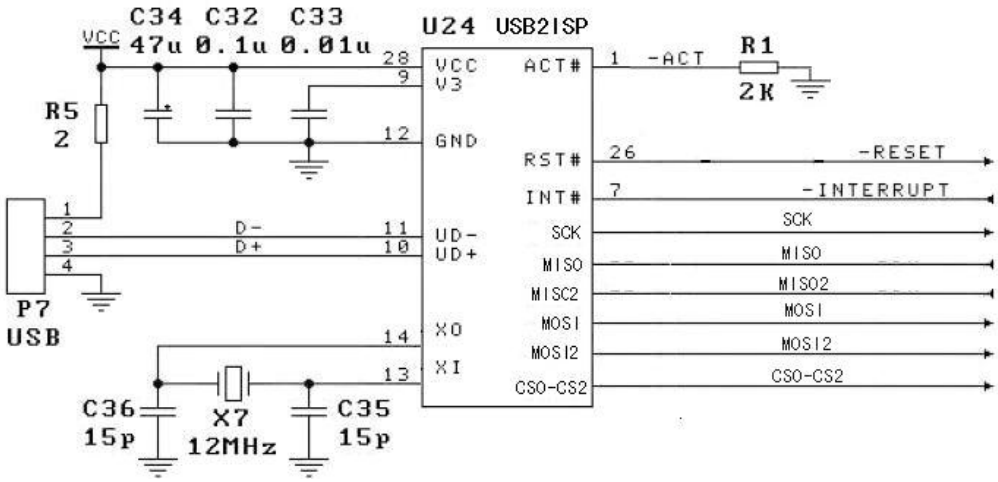
Parameter SetDataOut

- Switch outputs

- Bit set = switch output HIGH

- Bit clear = switch output LOW

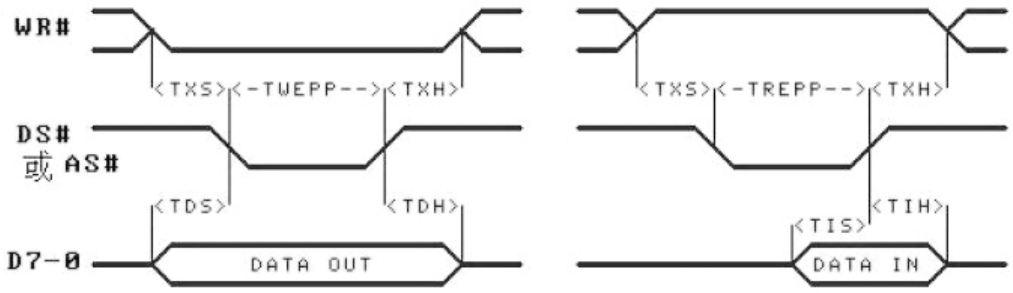
Example: USBIO_SetOutput(0, $FF, $FF, $F0) sets data direction of D0...D7 to output. Other lines stay untouched. D0...3 are programmed LOW. D4...D7 are programmed HIGH.



USB2ISP TO 5 WIRE SPI BUS

5.6 EPP PARALLEL PORT

The USB2ISP reads (/WR=HIGH) and writes (/WR=HIGH) from/to an 8 bit data bus. A data register (/DS=LOW) or address register (/AS=LOW) can be selected. The diagram below shows the signal pattern:



EPP MODEL

EPP MODE TIME

Pin	Name	Function	Remarks
22...15	D7...D0	bi-directional	8 bit data bus
25	WR#	Output	Read /Write
4	DS#	Output	/Data select
26	RST#	Output	/Reset
3	AS#	Output	/Address select
27	WAIT#	Input	/Wait
7	INT#	Input	Interrupt request (rising edge)
5	ERR#	Input	IN0
8	SLCT	Input	IN3
6	PEMP	Input	IN1

API functions:

Function USB2I2C/USB2SPI/USB2ISPEppReadData(...) Read data register /WR=1; /DS=0;

Function USB2I2C/USB2SPI/USB2ISPEppReadAddr(...) Read address register /WR=1; /AS=0;

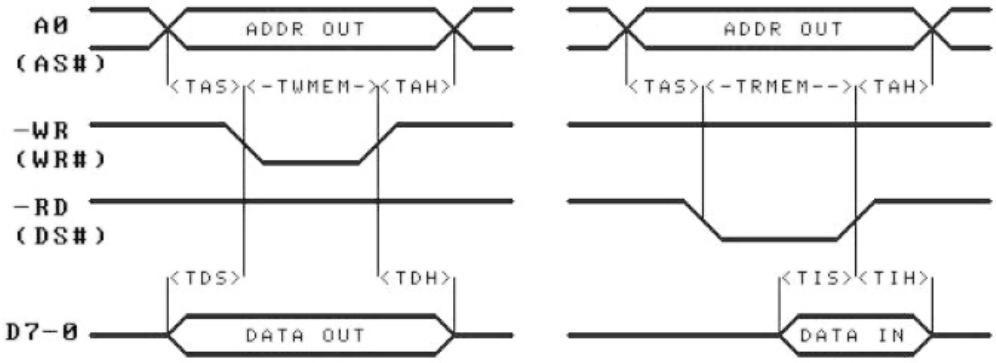
Function USB2I2C/USB2SPI/USB2ISPEppWriteData(...) Write data register /WR=0; /DS=0

Function USB2I2C/USB2SPI/USB2ISPEppWriteAddr(...) Write data register /WR=0; /AS=0

Function USB2I2C/USB2SPI/USB2ISPEppSetAddr(...) Write address register /WR=0; /AS=0 (1 Byte)

5.7 MEM PARALLEL PORT

The USB2ISP reads (/RD=LOW) and writes (/WR=LOW) from/to an 8-bit data bus. Address line A0 allows selecting between two registers (for example reading DAC A and DAC B of a LTC7528 Dual 8-Bit DA converter or registers of HD44780 compatible LCD displays.



MEM MODEL

MEM MODE TIME

Pin	Name	Function	Remark
22...15	D7...D0	bi-directional	8-bit data bus
25	WR#	Output	/Write
4	RD#	Output	/Read
26	RST#	Output	/Reset
3	A0	Output	Address select
27	WAIT#	Input	/Wait
7	INT#	Input	Interrupt request (rising edge)
5	ERR#	Input	IN0
8	SLCT	Input	IN3
6	PEMP	Input	IN1

API functions:

Function USB2I2C/USB2SPI/USB2ISPMemReadAddr0(...) Read; A0=LOW; /RD=0

Function USB2I2C/USB2SPI/USB2ISPMemReadAddr1(...) Read; A0=HIGH /RD=0

Function USB2I2C/USB2SPI/USB2ISPMemWriteAddr0(...) Write; A0=LOW; /WR=0

Function USB2I2C/USB2SPI/USB2ISPMemWriteAddr1(...) Write; A0=HIGH; /WR=0

9、USB2ISP INSTALL DRIVER

9.1 FOR WINDOWS

DOWNLOAD DRIVER

WEB: www.usb-i2c-spi.com/cn

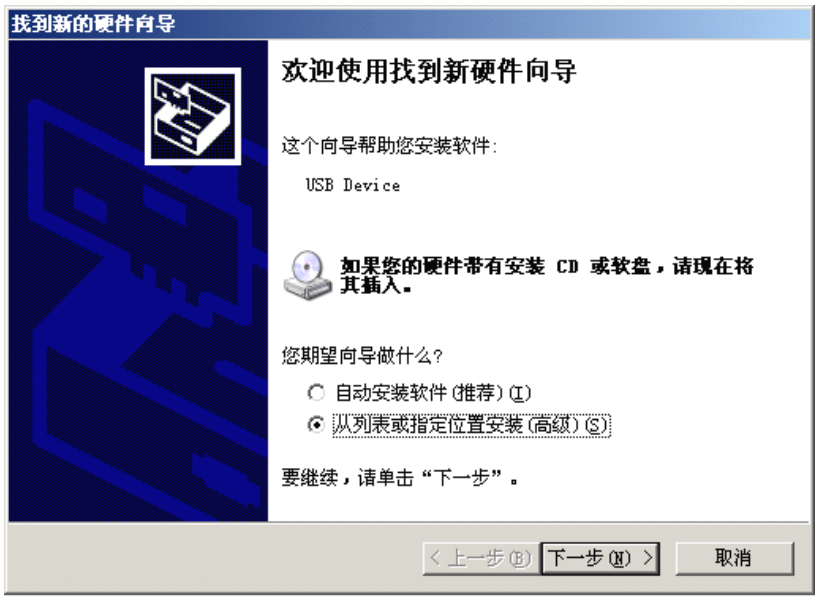
http://www.usb-i2c-spi.com/cn/down.htm。

```
├─DRIVER
|   |   USBIOX.DLL
|   |   USBIOX.INF
|   |   USBIOX.SYS
|   └─DRIVER_NEW0711      //DRIVER FILE
|           USBIOX.DLL     //FOR VC , VB , CBC , Delphi...
|           USBIOX.INF     //
|           USBIOX.SYS     //
└─LIB_C
        USBIOX.H          //All driver API instructions for use, which is an important documentation
USBIOX.LIB                //Library for VC , VB , CBC , Delphi ....
```

PLUS USB2ISP/USB2I2C/USB2SPI and Find new Hardware

\Windows发现了新USB硬件设备

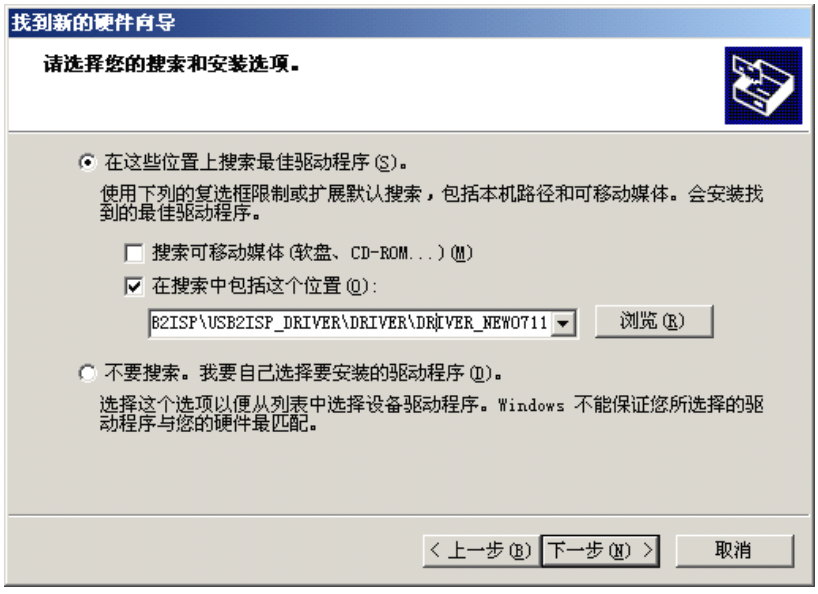
Prompts to install the driver



提示安装驱动

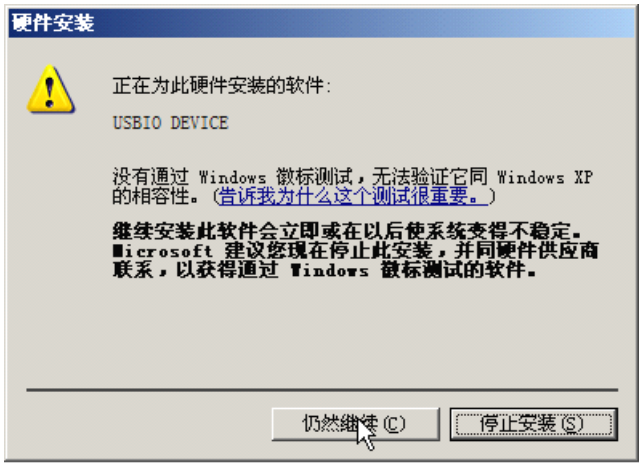
Select [from a list or specific location (Advanced)] option, and then click the Next button.

Specifies the path of the driver file



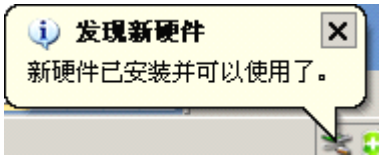
Copy files

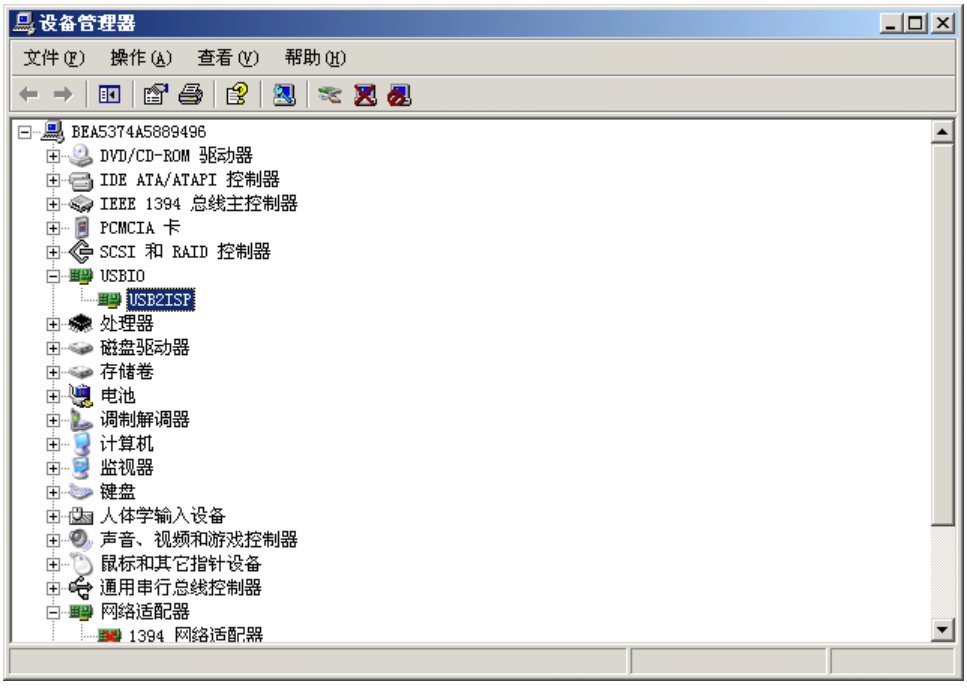
Next is system copied driven process. The first installation may also be prompted to "not passed Windows Logo testing", select [continue] button.



微软徽标认证

Successful installation





Through [My computer] → [Properties] → [Hardware Device Manager] to see the newly installed equipment. Can also open USB2ISP_DEMO_VBCN.exe, state display.

9.2 FOR LINUX

USB2I2C/USB2SPI/USB2ISP supplies Linux 2.6 kernel driver, need Libusb support open source.

Download :http://www.usb-i2c-spi.com/cn/down.htm, File name: 【USB2XXXLinuxDriver】

10、APPLICATION SOFTWARE DEVELOPMENT

In USB2ISP/USB2I2C/USB2SPI mode EPP / MEM / I²C / Digital I/O the chip can be accessed with API function calls.

Therefore USBIOX.DLL needs to be imported. Examples for Delphi, C, Visual Basic and CBC are available.

All API functions need to have the device number (iIndex) as first parameter.

The first chip on USB is device number 0 (iIndex=0), the 2nd chip on USB is device 1, and so on.

General functions

Function USBIO_OpenDevice:

Function connects to USB2ISP/USB2I2C/USB2SPI and needs to be called before other calls.

Function USBIO_CloseDevice:

Function terminates connection with USB2ISP/USB2I2C/USB2SPI .

Function USBIO_ResetDevice:

Function resets chip to power-on-state.

Function USBIO_SetExclusive:

iExclusive:

0=shared ;

1=exclusive;

For Information on other API functions please refer to the examples

```
// USB functions
// **************
type mPUSBIO_NOTIFY_ROUTINE =ProceDure (iEventStatus:cardinal); stdcall;
const USBIO_DEVICE_ARRIVAL = 3 ;
USBIO_DEVICE_REMOVE_PEND = 1 ;
USBIO_DEVICE_REMOVE = 0 ;
Function USBIO_SetDeviceNotify(iIndex:cardinal; iDeviceID: PCHAR;
iNotifyRoutine:mPUSBIO_NOTIFY_ROUTINE):boolean;Stdcall; external 'USBIOX.DLL';

// ********************
// General routines
// ********************
// USB2I2C/USB2SPI/USB2ISP is addressed with 'iIndex':
// iIndex=0 => 1st USB2I2C/USB2SPI/USB2ISP @ USB
// iIndex=1 => 2nd USB2I2C/USB2SPI/USB2ISP @ USB
// iIndex=2 => 3rd ...
// USBIO_OpenDevice: Connect to USB2I2C/USB2SPI/USB2ISP
Function USBIO_OpenDevice(iIndex: cardinal): integer; Stdcall; external 'USBIOX.DLL' ;
// USBIO_CloseDevice: Terminate connection with USB2I2C/USB2SPI/USB2ISP
procedure USBIO_CloseDevice(iIndex: cardinal); Stdcall; external 'USBIOX.DLL';
// USBIO_ResetDevice: USB2I2C/USB2SPI/USB2ISP reset to power-on defaults
Function USBIO_ResetDevice(iIndex: cardinal): boolean; Stdcall; external 'USBIOX.DLL';
// USBIO_SetExclusive: //Allow/prevent re-open
// iExclusive: 0=shared ;
// 1=exclusive;
Function USBIO_SetExclusive(iIndex:cardinal; iExclusive:cardinal ):boolean;Stdcall; external
'USBIOX.DLL';

// *********************
// Information requests
// *********************
Function USBIO_GetVersion: cardinal; Stdcall; external 'USBIOX.DLL';
Function USBIO_DriverCommand(iIndex: cardinal; ioCommand: mPWIN32_COMMAND):cardinal; Stdcall;
external 'USBIOX.DLL';
Function USBIO_GetDrvVersion: cardinal; Stdcall; external 'USBIOX.DLL';
Function USBIO_GetDeviceDescr(iIndex: cardinal; oBuffer: PVOID; ioLength: PULONG): boolean;
Stdcall; external 'USBIOX.DLL';
Function USBIO_GetConfigDescr(iIndex: cardinal; oBuffer: PVOID; ioLength: PULONG): boolean;
Stdcall; external 'USBIOX.DLL';
Function USBIO_GetDeviceName(iIndex:cardinal):PVOID;Stdcall; external 'USBIOX.DLL';
Function USBIO_GetVerIC(iIndex:cardinal):cardinal;Stdcall; external 'USBIOX.DLL';

// **********
// Interrupts
// **********
Function USBIO_SetIntRoutine(iIndex: cardinal; iIntRoutine: mPUSBIO_INT_ROUTINE): boolean;Stdcall;
external 'USBIOX.DLL';
Function USBIO_ReadInter(iIndex: cardinal; iStatus:PULONG): boolean; Stdcall; external
'USBIOX.DLL';
Function USB2I2C/USB2SPI/USB2ISPbortInter(iIndex: cardinal): boolean; Stdcall; external 'USBIOX.DLL';
```

```
//Delay; iDelay: Unit 1ms
Function USBIO_SetDelaymS(iIndex:cardinal; iDelay:cardinal ):boolean;Stdcall; external
'USBIOX.DLL';
//Timeout; iWriteTimeout: Unit 1ms
Function USBIO_SetTimeout(iIndex:cardinal; iWriteTimeout:cardinal; iReadTimeout:cardinal
):boolean;Stdcall; external 'USBIOX.DLL';
Function USBIO_ReadData(iIndex:cardinal; oBuffer:PVOID; ioLength:PULONG ):boolean;Stdcall;
external 'USBIOX.DLL';
Function USBIO_WriteData(iIndex:cardinal; iBuffer:PVOID; ioLength:PULONG ):boolean;Stdcall;
external 'USBIOX.DLL';
Function USBIO_FlushBuffer(iIndex:cardinal):boolean;Stdcall; external 'USBIOX.DLL';
Function USBIO_WriteRead(iIndex:cardinal; iWriteLength:cardinal; iWriteBuffer:PVOID;
iReadStep:cardinal; iReadTimes:cardinal; oReadLength:PULONG; oReadBuffer:PVOID
):boolean;Stdcall; external 'USBIOX.DLL' ;

// ***********************************
// Parallel data transfer (EPP/MEM)
// ***********************************
// iMode: 0 = EPP/EPP V1.7 ; 1 = EPP V1.9, 2 = MEM; 256 = keep current
Function USBIO_InitParallel(iIndex: cardinal; iMode :cardinal): boolean; Stdcall; external
'USBIOX.DLL';
// iMode: 0 = EPP/EPP V1.7 ; 1 = EPP V1.9, 2 = MEM
Function USBIO_SetParaMode(iIndex: cardinal; iMode: cardinal): boolean; Stdcall; external
'USBIOX.DLL';
// Read & Write (MEM und EPP)
Function USBIO_ReadData0(iIndex: cardinal; oBuffer: PVOID; ioLength: PULONG ): boolean; Stdcall;
external 'USBIOX.DLL';
Function USBIO_ReadData1(iIndex: cardinal; oBuffer: PVOID; ioLength: PULONG ): boolean; Stdcall;
external 'USBIOX.DLL';
Function USB2I2C/USB2SPI/USB2ISPbortRead(iIndex: cardinal): boolean; Stdcall; external 'USBIOX.DLL';
Function USBIO_WriteData0(iIndex:cardinal; iBuffer: PVOID; ioLength: PULONG ): boolean; Stdcall;
external 'USBIOX.DLL';
Function USBIO_WriteData1(iIndex:cardinal; iBuffer: PVOID; ioLength: PULONG ): boolean; Stdcall;
external 'USBIOX.DLL';
Function USB2I2C/USB2SPI/USB2ISPbortWrite(iIndex:cardinal): boolean; Stdcall; external 'USBIOX.DLL';
// EPP Read/Write
Function USBIO_EppReadData(iIndex:cardinal; oBuffer:PVOID; ioLength:PULONG ):boolean; Stdcall;
external 'USBIOX.DLL';
Function USBIO_EppReadAddr(iIndex:cardinal; oBuffer:pvoid; ioLength:PULONG ):boolean;Stdcall;
external 'USBIOX.DLL';
Function USBIO_EppWriteData(iIndex:cardinal; iBuffer:pvoid; ioLength:PULONG ):boolean;Stdcall;
external 'USBIOX.DLL';
Function USBIO_EppWriteAddr(iIndex:cardinal; iBuffer:PVOID; ioLength:PULONG ):boolean;Stdcall;
external 'USBIOX.DLL';
Function USBIO_EppSetAddr(iIndex:cardinal; iAddr:byte ):boolean;Stdcall; external 'USBIOX.DLL';
// MEM Read/Write
Function USBIO_MemReadAddr0(iIndex:cardinal; oBuffer:pvoid; ioLength:PULONG ):boolean;Stdcall;
external 'USBIOX.DLL';
Function USBIO_MemReadAddr1(iIndex:cardinal; oBuffer:pvoid; ioLength:PULONG ):boolean;Stdcall;
external 'USBIOX.DLL';
function USBIO_MemWriteAddr0(iIndex:cardinal; iBuffer:pvoid; ioLength:PULONG ):boolean;Stdcall;
external 'USBIOX.DLL';
Function USBIO_MemWriteAddr1(iIndex:cardinal; iBuffer:pvoid; ioLength:PULONG ):boolean;Stdcall;
external 'USBIOX.DLL';
```


```
external 'USBIOX.DLL';
// ****************************************
// Serial data transfer (I2C, SPI)
// ****************************************
// USBIO_SetStream() konfiguriert I2C und SPI
// Bit 1-0: I2C speed 00= low speed /20KHz
// 01= standard /100KHz
// 10= fast /400KHz
// 11= high speed /750KHz
// Bit 2: SPI - Modus
// 0= Standard SPI (D3=clock out, D5=serial out, D7 serial in
// 1= SPI with two data lines (D3=clock out, D5,D4=serialout, D7,D6 serial in)
// Bit 7: SPI 0= LSB first
// 1= MSB first
// other bits 0 (reserved)
Function USBIO_SetStream(iIndex:cardinal; iMode:cardinal):boolean;Stdcall; external
'USBIOX.DLL';

Function USBIO_SetStream(iIndex:cardinal; iMode:cardinal):boolean;Stdcall; external
'USBIOX.DLL';
// ********************************
// SPI - Serial Peripheral Interface
// ********************************
//
// USB2I2C/USB2SPI/USB2ISP is MASTER in any case
//
// USBIO_BitStreamSPI()
// Serial daten transfer with (non-SPI conform) IC's (e.g. LTC1257)
// Simultaneously READ and WRITE through ioBuffer byte array:
// Bit7: D7 <- Input (read data)
// Bit6: D6 <- Input (read data)
// Bit5: D5 -> Output (write data)
// Bit4: D4 -> Output (write data)
// Bit3: D3 -> Output (SCLK; serial clock generated by USB2I2C/USB2SPI/USB2ISP);
// Bit2: D2 -> Output (write data)
// Bit1: D1 -> Output (write data)
// Bit0: D0 -> Output (write data)
// Output pattern is defined by bits D0,D1,D2,D4,D5 of bytes in ioBuffer (before function call).
// Bits D6 and D7 of bytes in ioBuffer contain read data (after function call).
// D3 is the auto-generated clock signal. (positive pulse for each byte in ioBuffer)
// Call Set_D5_D0(iIndex, $3F) before to set D0...D5 direction to output!
// iLength is the number of bytes to be tranfered through ioBuffer.
```


```
Function USBIO_BitStreamSPI(iIndex:cardinal; iLength:cardinal; ioBuffer:PVOID):boolean;Stdcall;
external 'USBIOX.DLL';
// USBIO_StreamSPI4()
// Standard SPI (Mode 0) with two uni-directional data lines, clock and chip select.
//
// D7 <- Input (Read data; MISO)
// D5 -> Output (Write data; MOSI)
// D3 -> Output (SCLK; serial clock generated by USB2I2C/USB2SPI/USB2ISP);
// D2 -> Output (Chip Select 2)
// D1 -> Output (Chip Select 1)
// D0 -> Output (Chip Select 0)
Function USBIO_StreamSPI4(iIndex:cardinal; iChipSelect:cardinal; iLength:cardinal; ioBuffer:PVOID
):boolean;Stdcall; external 'USBIOX.DLL';
// USBIO_StreamSPI3()
// SPI with only ONE bi-directional(!) data line, clock and chip select
// e.g. Maxim (DS1626); (Write with rising clock; read with falling clock)
Function USBIO_StreamSPI3(iIndex:cardinal; iChipSelect:cardinal; iLength:cardinal;
ioBuffer:PVOID):boolean;Stdcall; external 'USBIOX.DLL';
// USBIO_StreamSPI5()
// SPI with 4 data lines (2 pairs uni-directional);
Function USBIO_StreamSPI5(iIndex:cardinal; iChipSelect:cardinal; iLength:cardinal; ioBuffer:PVOID;
ioBuffer2:PVOID ):boolean;Stdcall; external 'USBIOX.DLL';

// ***********
// *** I2C ***
// ***********
//
// USB2I2C/USB2SPI/USB2ISP is MASTER in any case
//
// USBIO_StreamI2C(): Universal READ and/or WRITE over I2C
// READ only: iWriteLength = 0;
// WRITE only: iReadLength = 0;
Function USBIO_StreamI2C(iIndex:cardinal; iWriteLength:cardinal; iWriteBuffer:pvoid;
iReadLength:cardinal; oReadBuffer:pvoid ):boolean;stdcall; external 'USBIOX.DLL';
// READ and WRITE from/to IC with internal register-/memory address
// iDevice: 0..127 = I2C device address
// iAddr: register adress; This is not the I2C device addresse!
Function USBIO_ReadI2C(iIndex:cardinal; iDevice: byte; iAddr: byte; oByte: pbyte ):boolean;
Stdcall; external 'USBIOX.DLL';
Function USBIO_WriteI2C(iIndex:cardinal; iDevice :byte; iAddr: byte; iByte: byte ):boolean;
Stdcall; external 'USBIOX.DLL';
// I2C-EEPROM: READ and WRITE
type EEPROM_TYPE
=(ID_24C01,ID_24C02,ID_24C04,ID_24C08,ID_24C16,ID_24C32,ID_24C64,ID_24C128,ID_24C256,ID_24C512,ID
_24C1024,ID_24C2048,ID_24C4096);
Function USBIO_ReadEEPROM(iIndex:cardinal; iEepromID:EEPROM_TYPE; iAddr:cardinal;
iLength:cardinal; oBuffer:Pbytearray ):boolean;stdcall; external 'USBIOX.DLL';
Function USBIO_WriteEEPROM(iIndex:cardinal; iEepromID:EEPROM_TYPE; iAddr:cardinal;
iLength:cardinal; iBuffer:pbytearray ):boolean;stdcall; external 'USBIOX.DLL';
```


```
// Direct I/O pins programming
// ************************************
// iEnable: Enable data direction changes
// Bit set = allow changes
// Bit clear = prevent changes
// iSetDirOut: Data direction setting: Input or Output
// Bit set = Pin programmed as output (Caution!)
// Bit clear = Pin is input
// iSetDataOut: Output status
// Bit set = Output HIGH
// Bit clear = Output LOW
// Pin assignment USB2I2C/USB2SPI/USB2ISP:
// Bi-directional pins (input or output):
// Bit 0-7 = D0-D7 pin 15-22
// Bit 8 = ERR# pin 5
// Bit 9 = PEMP pin 6
// Bit 10 = ACK pin 7
// Bit 11 = SLCT pin 8
// Bit 12 = unused -
// Bit 13 = BUSY/WAIT# pin 27
// Bit 14 = AUTOFD#/DATAS# pin 4
// Bit 15 = SLCTIN#/ADDRS# pin 3
// pseudo-bi-directional pins (Combination of input and open-collector output) :
// Bit 23 = SDA pin 23 (for functions GetInput/GetStatus)
// Bit 16 = SDA pin 23 (for funktion SetOutput)
// Uni-directional pins (output only):
// Bit 16 = RESET# pin 26
// Bit 17 = WRITE# pin 25
// Bit 18 = SCL pin pin 24
// Note: Other API functions (e.g. SPI, ... may change pins!
// Connections between two outputs can cause damage to the chip.
// Set outputs
Function USBIO_SetOutput(iIndex:cardinal; iEnable:cardinal;
iSetDirOut:cardinal;iSetDataOut:cardinal ):boolean;Stdcall; external 'USBIOX.DLL';
Function USBIO_Set_D5_D0(iIndex:cardinal; iSetDirOut:cardinal; iSetDataOut:cardinal
):boolean;Stdcall; external 'USBIOX.DLL';
// read inputs
Function USBIO_GetInput(iIndex:cardinal; iStatus:PULONG ):boolean;Stdcall; external
'USBIOX.DLL';
Function USBIO_GetStatus(iIndex:cardinal; iStatus: PULONG ): boolean; Stdcall; external
'USBIOX.DLL';
```


11、FOOTPRINT DIMENSIONS

SOP28 footprint (mil/mm)

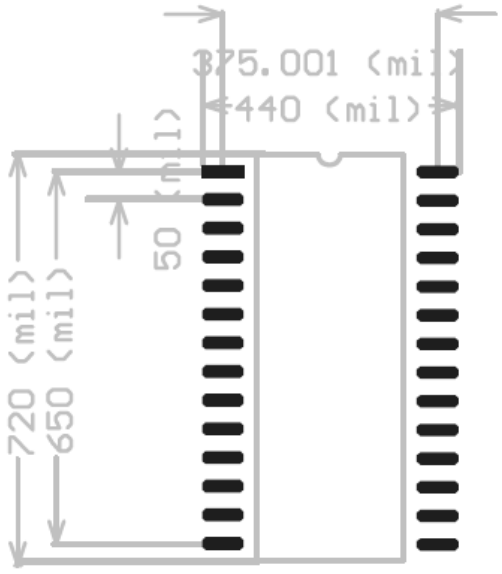


FIG. 11 - USB2ISP (MIL)

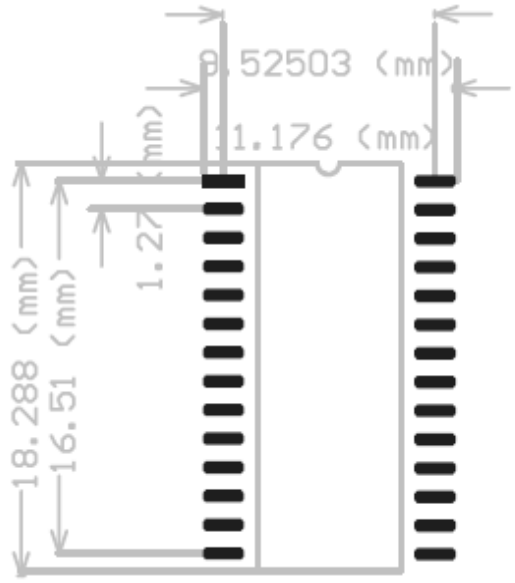


FIG. 12 - USB2ISP (MM)

12. COPYRIGHT