J-Works, Inc.
www.j-works.com

Contents

Users guide to using modules under Linux or other OS's
Version 1.13

J-Works, Inc.
www.j-works.com

1. Overview

Purpose

This document provides information about how to use the J-Works,Inc USB modules under the linux operating system and other non-windows operating systems. The code generated can either be user-mode, or kernel. The user needs to have basic knowledge in the following areas, USB communications, software coding and building under linux, linux usb file system. Most of our products also have a support disk available as a one-time purchase that contains source code with a make file, that has been built and tested. Each disk contains two (2) sets of source code, one was written using "libusb", the other using "usbdevfs". Both requires that the user build it for the distribution and version of linux they are using. It is not required, as the following information about the module interface is all that is required to generate code. Also this information can be used in other operating systems, as again it is just the basic process.

Usage of this Document

This document explains the basic usb communications required to access and control J-Works usb devices. In samples shown, "libusb" usb functions are shown. "Libusb" is a open source project USB library that can be found at http://libusb.org/. To better understand concepts in this document, the user should have access to the "libusb" documentation, as it will add to the understanding the lower level usb communications that is a part of the operating system. Functions starting with "libusb_" are from the library, and variable types for various usb structures are define in the library includes. Any type of usb support library (like "usbdevfs") can be used. Also any of the communications can also be done in kernel mode. Samples in this code is shown with general "C" syntax, but to help in clarity of the sample, variable types are defined at their actual usage point. Also for clarity no error checking of "libusb" functions are done, and for actual code should be included. Constant values shown as *JW_valuename* are defined in the last sections of the document, as they are unique to each device. All control functions are shown and apply to the modules that require that function, and are not present in all modules. If under section 8 listing of module values, no value is listed for the module model number, it means that function/command is not in that module. If you go the end of the document, all the modules are listed in order and is a good starting point to determine what functions are available.

Basic USB interface concepts

Most of the J-Works modules only use the USB control pipe to control and receive information to each of the modules. Each module has a unique set of vendor command values and are listed in section x. When a module is plugged in, the linux usb file system communicates with the module and obtains some basic information about the device.

Vendor ID, which in the case of all J-Works modules is 0x7c3

Product ID, this is unique to each type of module, and is listed in section x.

By using these two values, a user can determine if the module is available on the system.

Most of the control and data commands for J-Works modules use the vendor control pipe. This takes a data structure that contains a 8 bit control value which is the command value, a 16 bit value, and a 16 bit index. The value and index supply support value for the command. Things like relay number, or settings value.

If the module your are interfacing to number does not exist in the following sections, it is only because the document has not been updated with those values. Please contact us @ support@j-works.com and within a day or two those values will be added.

2. General Interface commands

Finding the device

To find the device you want, it is required to search thru all the usb devices that are attached to the system, and compare their Vendor ID and Product ID.

Global variables used in various code samples

```
libusb_device **devs;
static struct libusb_device_handle* deviceHandle;
```

Sample for finding a device, opening it for later used (deviceHandle)

```
int i = 0;
libusb_init(NULL);                  // init the libusb
libusb_device *dev;
// get a list of all usb devices connected
int deviceCount = libusb_get_device_list(NULL, &devs);
if (deviceCount > 0) {
while ( ((dev = devs[i++]) != NULL) )  { // to thru list checking vendor ID
        struct libusb_device_descriptor desc;
        libusb_get_device_descriptor(dev, &desc); // get the vendor id and product id
        if (desc.idVendor == JW_vendorId && desc.idProduct == JW_productId)   {
                libusb_open(dev,&deviceHandle);
                libusb_claim_interface(deviceHandle, 0); }    }
```

sample for closing device when done with it

```
libusb_close(deviceHandle);
libusb_free_device_list(devs, 1);
libusb_exit(NULL);
```

Serial Number String

Each module has an unique serial number string. If multiple modules of the same type are attached, it can be used to indentify each of the modules. Sample code only shows how to obtain the serial number string, usage depends on user applications. If multiple handles are open, serial number could be stored to know which module was which, and used in generic control functions. Exact string length can vary based on module, but none exceed or will exceed 32 bytes.

sample for obtaining serial number string.

```
unsigned char data[32];
size_t length;
libusb_get_string_descriptor_ascii(deviceHandle,JW_serialNumberIndex,data,length);
```

Flash Led

This command causes the LED on the module to flash a number of times and is used to help identify a module when multiple modules are attached

```
unsigned short value = 0;
unsigned short index = 0;
unsigned char returnData[JW_returnSize];

libusb_control_transfer(deviceHandle, JW_flashLedCmd,value,index, returnData, JW_returnSize, 0);
```

3. Relay Related Commands

Number Relays

This command returns the number of relays on the module.

```
unsigned short value = 0;
unsigned short index = 0;
unsigned char returnData[JW_returnSize];

libusb_control_transfer(deviceHandle, JW_numRelayCmd,value,index, returnData, JW_returnSize, 0);
returnData[JW_byteForNum] is the number of relays
```

Clear Relay

This command clears (opens) the relay on the module based on bit value. Each bit of the value is a relay. IE bit 0 is relay one, when the bit is set (1) that relay is open. When the bit is (0) the relay for that bit is unchanged.

```
unsigned short value = relayBitValue;
unsigned short index = 0;
unsigned char returnData[JW_returnSize];

libusb_control_transfer(deviceHandle, JW_clearRelayCmd,value,index,returnData, JW_returnSize, 0);
```

Set Relay

This command sets (closes) the relay on the module based on bit value. Each bit of the value is a relay. IE bit 0 is relay one (1), when the bit is set (1) that relay is closed. When the bit is (0) the relay for that bit is unchanged.

```
unsigned short value = relayBitValue;
unsigned short index = 0;
unsigned char returnData[JW_returnSize];

libusb_control_transfer(deviceHandle, JW_setRelayCmd,value,index,returnData, JW_returnSize, 0);
```

Relay State

This command returns a bit value that is the current state of the relays on the module. Each bit of the return value is a relay. IE bit 0 is relay one (1), when the bit is set (1) the relay is closed, when the bit is clear (0) the relay is open.

```
unsigned short value = relayBitValue;
unsigned short index = 0;
unsigned char returnData[JW_returnSize];

libusb_control_transfer(deviceHandle, JW_relayStateCmd,value,index,returnData, JW_returnSize, 0);
returnData[0] is the lsb relay
```

Set all Relays based on value

This commands sets all the relays based on the bit value. When a bit is one (1) the relay is closed, when the bit is zero (0) the relay is open, and the lsb of the value is relay 1. This command changes all the relays based upon value.

```
unsigned short value = relayBitValue; // each relay is changed
unsigned short index = 0;
unsigned char returnData[JW_returnSize];

libusb_control_transfer(deviceHandle, JW_relaySetAllCmd,value,index,returnData, JW_returnSize, 0);
returnData[0] is the lsb relay
```

4. IO related commands

Number IO

This command returns the number of io lines on the module.

```
unsigned short value = 0;
unsigned short index = 0;
unsigned char returnData[JW_returnSize];

libusb_control_transfer(deviceHandle, JW_numIOCmd,value,index, returnData, JW_returnSize, 0);
returnData[JW_byteForNumInput] is the number of inputs
returnData[JW_byteForNumInput] is the number of outputs
returnData[JW_byteIoConfig] is the IO Configuration, a '1' means can be configured
```

Read IO input value

This commands returns the current input values of the IO bits.

```
unsigned short value = 0;
unsigned short index = 0;
unsigned char returnData[JW_returnSize];

libusb_control_transfer(deviceHandle, JW_ReadInCmd,value,index,returnData, JW_returnSize, 0);
returnData[0] is the lsb 8 bits and bit 0 is the lsb bit, or in some cases the only bit.  A value of 1 means the input is on. On some devices on/off state can be reversed based on logic / opto type.
```

Set IO output value

This commands sets the value of the output bit for modules that have configurable IO. Control bits are set by banks of 16 bits each. Bank 1 is the lsb section and each bank is the next 16 bits as required to the maximum number of IO bits. If the direction bit is set to read, command has no effect until the direction bit is set to output.

```
unsigned short value = outputBank;
unsigned short index = outBits;
unsigned char returnData[JW_returnSize];

libusb_control_transfer(deviceHandle, JW_setOutByBankCmd,value,index,returnData,
JW_returnSize, 0);
```

Set IO direction value

This commands sets the value of the direction control bit for modules that have configurable IO. Control bits are set by banks of 16 bits each. Bank 1 is the lsb section and each bank is the next 16 bits as required to the maximum number of IO bits. When the bit is set to one (1) the IO is an output, otherwise when zero (0) it is a input.

```
unsigned short value = inputBank;
unsigned short index = dirBits;
unsigned char returnData[JW_returnSize];

libusb_control_transfer(deviceHandle, JW_setDirByBankCmd,value,index,returnData, JW_returnSize,
0);
```

5. Watchdog Timer related commands

Set Watchdog timer values

This commands sets all the values used in the watchdog timer functions. There are two times sent, the first is the time period that the reset command must be sent or the timer will timeout. The second time is how low the relay will stay closed if the timeout occurs. A value of 0 for reset means the relay will stay closed. Both times are 100msec per count.

```
unsigned short sValue = usTime; // number of 100msec periods that wdt must be reset within
unsigned short sIndex = usReset;// number of 100msec periods that after time, relay will stay closed
unsigned char returnData[JW_returnSize];

libusb_control_transfer(deviceHandle,JW_setWdt, sValue, sIndex, returnData,JW_returnSize,0);
```

Enable Watchdog timer

This command enables the watchdog timer based on values previous sent using the set command. If a reset command is not received within the time period set, the timeout will occur, and the relay will close.

```
unsigned char returnData[JW_returnSize];
libusb_control_transfer(deviceHandle,JW_enableWdt,0,0, returnData,JW_returnSize,0);
```

Turn off watchdog timer

This command turns off the watchdog timer.

```
libusb_control_transfer(deviceHandle,JW_clrWdt,0,0, returnData,JW_returnSize,0);
```

Reset watchdog timer

This command resets the wdt, and must be called within the period set. Some modules have a hardware input that must be toggled within that time.

```
libusb_control_transfer(deviceHandle,JW_resetWdt,0,0, returnData,JW_returnSize,0);
```

6. Counter related commands

Read event count and trigger value

This command returns the current count value. Byte 0 of the returnData is the lsb, and the length of the count depends on the counter size of the module, and the counter value and trigger value is the same length JW_numCountBytes. So if the value is 3, count is bytes 0,1,2 and trigger value is 3,4,5. Trigger value on modules that support it, is the count that causes hardware lines to change.

```
unsigned char returnData[JW_returnSize];
libusb_control_transfer(deviceHandle,JW_readCount,0,0, returnData,JW_returnSize,0);
```

Reset event count

This command resets the current count value to zero (0).

```
unsigned char returnData[JW_returnSize];
libusb_control_transfer(deviceHandle,JW_resetCount,0,0, returnData,JW_returnSize,0);
```

Set Edge and Reset mode

This command sets the edge used to trigger event count, and whether external hardware reset is used. When the lsb bit 0 of the value data is set, the falling edge is used. When bit 1 is set external reset is used.

```
unsigned short value = modeValue;
unsigned char returnData[JW_returnSize];
libusb_control_transfer(deviceHandle,JW_resetCount,value,0, returnData,JW_returnSize,0);
```

Read Edge and Reset mode

This command returns the edge / reset mode value in returnData[0].

```
unsigned char returnData[JW_returnSize];
libusb_control_transfer(deviceHandle,JW_readEdgeMode,0,0, returnData,JW_returnSize,0);
```

Set Trigger

This command sets the event trigger value. The value is the lsb and the index is the msb to form up to a 32 bit value depending on size of counter.

```
unsigned short value = lsbTriggerValue;
unsigned short index = msbTriggerValue;
unsigned char returnData[JW_returnSize];
libusb_control_transfer(deviceHandle,JW_setEventTrigger,value,index, returnData,JW_returnSize,0);
```

7. Analog related commands

DAC configuration

This command returns the configuration of the DAC Module. returnData Byte 0 of the return is resolution of the DAC in bits. Byte 1 is the number of DAC channels. Byte 2 is the range of the DAC as volts.

```
unsigned char returnData[JW_returnSize];
libusb_control_transfer(deviceHandle,JW_dacConfig,0,0, returnData,JW_returnSize,0);
```

Set DAC

This command sets the DAC value. "index" is the channel number and is zero based (ie 0 = channel 1)

```
unsigned char returnData[JW_returnSize];
libusb_control_transfer(deviceHandle,JW_setDac,0,index, returnData,JW_returnSize,0);
```

Read DAC

This command reads the current DAC value. "index" is the channel number and is zero based (ie 0 = channel 1). "returnData" each pair of values (0,1 etc) is the current DAC setting, lower byte is lsb and upper byte is msb. Bank 0 is channels 0-3.

```
unsigned char returnData[JW_returnSize];
libusb_control_transfer(deviceHandle,JW_readDac,0,index, returnData,JW_returnSize,0);
```

Read ADC

This command reads the current ADC value. "index" is the adc channel number and is zero based (ie 0 = channel 1) . "returnData" is an array of data which also includes device information.

```
unsigned char returnData[JW_returnSize];
libusb_control_transfer(deviceHandle,JW_readADC,0,index, returnData,JW_returnSize,0);
```

returnData Array

byte 0 lsb of adc reading

byte 1 msb of adc reading

byte 2 channel number that was sampled

byte 3 reserved for future use

byte 4 number of adc bits

byte 5 adc voltage, see module values for actual range

byte 6 gain of channel

byte 7 reserved for future use

8. Module Values

ASB67412 Values

JW_vendorId	= 0x07c3
JW_productId	= 0x0671
JW_serialNumberIndex	= 5
JW_flashLedCmd	= 0xb1
JW_returnSize	= 8
JW_dacConfig	= 0xc1
JW_setDac	= 0xb2
JW_readDac	= 0xb4

CSB147x Values

JW_vendorId	= 0x07c3
JW_productId	= 0x8470
JW_serialNumberIndex	= 6
JW_flashLedCmd	= 0xd1
JW_returnSize	= 8
JW_numIOCmd	= 0xc1
JW_byteForNumInput	= 0
JW_byteForNumOutput	= 1
JW_byteIoConfig	= 2
JW_setDirByBankCmd	= 0xce
JW_setOutByBankCmd	= 0xc5
JW_readInCmd	= 0xc9

DSB1616

JW_vendorId	= 0x07c3
JW_productId	= 0x8470
JW_serialNumberIndex	= 6
JW_flashLedCmd	= 0xd1
JW_returnSize	= 8
JW_numIOCmd	= 0xc1
JW_byteForNumInput	= 0
JW_byteForNumOutput	= 1
JW_byteIoConfig	= 2
JW_setDirByBankCmd	= 0xce
JW_setOutByBankCmd	= 0xc5
JW_readInCmd	= 0xc9

JSB210A Values

JW_vendorId	= 0x07c3
JW_productId	= 0x0002
JW_serialNumberIndex	= 5
JW_flashLedCmd	= 0x04
JW_returnSize	= 5
JW_numRelayCmd	= 0xb2
JW_byteForNum	= 0

JW_clearRelayCmd	= 0x0c
JW_setRelayCmd	= 0x0b
JW_relayStateCmd	= 0x0a

JSB212 Values

JW_vendorId	= 0x07c3
JW_productId	= 0x0002
JW_serialNumberIndex	= 5
JW_flashLedCmd	= 0x04
JW_returnSize	= 5
JW_numRelayCmd	= 0xb2
JW_byteForNum	= 0
JW_clearRelayCmd	= 0x0c
JW_setRelayCmd	= 0x0b
JW_relayStateCmd	= 0x0a

JSB215A Values

JW_vendorId	= 0x07c3
JW_productId	= 0x0002
JW_serialNumberIndex	= 5
JW_flashLedCmd	= 0x04
JW_returnSize	= 5
JW_numRelayCmd	= 0xb2
JW_byteForNum	= 0
JW_clearRelayCmd	= 0x0c
JW_setRelayCmd	= 0x0b
JW_relayStateCmd	= 0x0a

JSB218 Values

JW_vendorId	= 0x07c3
JW_productId	= 0x1200
JW_serialNumberIndex	= 3
JW_flashLedCmd	= 0x01
JW_returnSize	= 5
JW_numRelayCmd	= 0x03
JW_byteForNum	= 0
JW_clearRelayCmd	= 0x07
JW_setRelayCmd	= 0x06
JW_relayStateCmd	= 0x04

JSB219 Values

JW_vendorId	= 0x07c3
JW_productId	= 0x1200
JW_serialNumberIndex	= 3
JW_flashLedCmd	= 0x01
JW_returnSize	= 8
JW_numRelayCmd	= 0x03
JW_byteForNum	= 0
JW_clearRelayCmd	= 0x07
JW_setRelayCmd	= 0x06

JW_relayStateCmd	= 0x04

JSB229 Values

JW_vendorId	= 0x07c3
JW_productId	= 0x8229
JW_serialNumberIndex	= 5
JW_flashLedCmd	= 0x01
JW_returnSize	= 8
JW_numRelayCmd	= 0x03
JW_byteForNum	= 1
JW_clearRelayCmd	= 0x07
JW_setRelayCmd	= 0x06
JW_relayStateCmd	= 0x04

JSB231 Values

JW_vendorId	= 0x07c3
JW_productId	= 0x0002
JW_serialNumberIndex	= 5
JW_flashLedCmd	= 0x04
JW_returnSize	= 5
JW_numRelayCmd	= 0xb2
JW_byteForNum	= 0
JW_clearRelayCmd	= 0x0c
JW_setRelayCmd	= 0x0b
JW_relayStateCmd	= 0x0a

JSB2431 Values

JW_vendorId	= 0x07c3
JW_productId	= 0x2341
JW_serialNumberIndex	= 5
JW_flashLedCmd	= 0xb1
JW_returnSize	= 8
JW_numRelayCmd	= 0xb2
JW_byteForNum	= 0
JW_clearRelayCmd	= 0xb9
JW_setRelayCmd	= 0xb8
JW_relayStateCmd	= 0xb7

JSB252 Values

JW_vendorId	= 0x07c3
JW_productId	= 0x0252
JW_serialNumberIndex	= 5
JW_flashLedCmd	= 0xb1
JW_returnSize	= 8
JW_numRelayCmd	= 0xb2
JW_byteForNum	= 0
JW_clearRelayCmd	= 0xb9
JW_setRelayCmd	= 0xb8
JW_relayStateCmd	= 0xb7
JW_relaySetAllCmd	= 0xb5

JSB255 Values

JW_vendorId	= 0x07c3
JW_productId	= 0x8229
JW_serialNumberIndex	= 5
JW_flashLedCmd	= 0x01
JW_returnSize	= 8
JW_numRelayCmd	= 0x03
JW_byteForNum	= 1
JW_clearRelayCmd	= 0x07
JW_setRelayCmd	= 0x06
JW_relayStateCmd	= 0x04

JSB265 Values

JW_vendorId	= 0x07c3
JW_productId	= 0x1200
JW_serialNumberIndex	= 3
JW_flashLedCmd	= 0x01
JW_returnSize	= 5
JW_numRelayCmd	= 0x03
JW_byteForNum	= 0
JW_clearRelayCmd	= 0x07
JW_setRelayCmd	= 0x06
JW_relayStateCmd	= 0x04

JSB272 Values

JW_vendorId	= 0x07c3
JW_productId	= 0x1200
JW_serialNumberIndex	= 3
JW_flashLedCmd	= 0x01
JW_returnSize	= 5
JW_numRelayCmd	= 0x03
JW_byteForNum	= 0
JW_clearRelayCmd	= 0x07
JW_setRelayCmd	= 0x06
JW_relayStateCmd	= 0x04

JSB273 Values

JW_vendorId	= 0x07c3
JW_productId	= 0x1200
JW_serialNumberIndex	= 3
JW_flashLedCmd	= 0x01
JW_returnSize	= 5
JW_numRelayCmd	= 0x03
JW_byteForNum	= 0
JW_clearRelayCmd	= 0x07
JW_setRelayCmd	= 0x06
JW_relayStateCmd	= 0x04

JSB274 Values

JW_vendorId	= 0x07c3
JW_productId	= 0x1200
JW_serialNumberIndex	= 3
JW_flashLedCmd	= 0x01
JW_returnSize	= 5
JW_numRelayCmd	= 0x03
JW_byteForNum	= 0
JW_clearRelayCmd	= 0x07
JW_setRelayCmd	= 0x06
JW_relayStateCmd	= 0x04

JSB280 Values

JW_vendorId	= 0x07c3
JW_productId	= 0x1280
JW_serialNumberIndex	= 3
JW_flashLedCmd	= 0x01
JW_returnSize	= 5
JW_numRelayCmd	= 0x03
JW_byteForNum	= 0
JW_clearRelayCmd	= 0x07
JW_setRelayCmd	= 0x06
JW_relayStateCmd	= 0x04
JW_relaySetAllCmd	= 0x02

JSB281 Values

JW_vendorId	= 0x07c3
JW_productId	= 0x1281
JW_serialNumberIndex	= 5
JW_flashLedCmd	= 0xb1
JW_returnSize	= 8
JW_numRelayCmd	= 0xb2
JW_byteForNum	= 0
JW_clearRelayCmd	= 0xb4
JW_setRelayCmd	= 0xb3
JW_relayStateCmd	= 0x04
JW_relaySetAllCmd	= 0x02

JSB283Values

JW_vendorId	= 0x07c3
JW_productId	= 0x1283
JW_serialNumberIndex	= 5
JW_flashLedCmd	= 0xb1
JW_returnSize	= 8
JW_numRelayCmd	= 0xc1
JW_byteForNum	= 1
JW_clearRelayCmd	= 0xb7
JW_setRelayCmd	= 0xb6
JW_relayStateCmd	= 0xb8
JW_relaySetAllCmd	= 0xc4

JSB2851 Values

JW_vendorId	= 0x07c3
JW_productId	= 0x1285
JW_serialNumberIndex	= 6
JW_flashLedCmd	= 0xb1
JW_returnSize	= 8
JW_clearRelayCmd	= 0xb9
JW_setRelayCmd	= 0xb8
JW_relayStateCmd	= 0xb7

JSB2944 Values

JW_vendorId	= 0x07c3
JW_productId	= 0x2280
JW_serialNumberIndex	= 5
JW_flashLedCmd	= 0xb1
JW_returnSize	= 8
JW_numRelayCmd	= 0xb2
JW_byteForNum	= 0
JW_clearRelayCmd	= 0xb9
JW_setRelayCmd	= 0xb8
JW_relayStateCmd	= 0xb7
JW_relaySetAllCmd	= 0xb5

JSB310 Values

JW_vendorId	= 0x07c3
JW_productId	= 0x1310
JW_serialNumberIndex	= 5
JW_flashLedCmd	= 0xb1
JW_returnSize	= 8
JW_clearRelayCmd	= 0xb6
JW_setRelayCmd	= 0xb7
JW_relayStateCmd	= 0xb8
JW_ReadInCmd	= 0xb9

JSB342 Values

JW_vendorId	= 0x07c3
JW_productId	= 0x1400
JW_serialNumberIndex	= 3
JW_flashLedCmd	= 0xb9
JW_returnSize	= 8
JW_clearRelayCmd	= 0xc1
JW_setRelayCmd	= 0xc0
JW_relayStateCmd	= 0xb5
JW_ReadInCmd	= 0xb2
JW_relaySetAllCmd	= 0xb1

JSB383 Values

JW_vendorId	= 0x07c3
JW_productId	= 0x8383
JW_serialNumberIndex	= 5
JW_flashLedCmd	= 0xb9
JW_returnSize	= 8
JW_ReadInCmd	= 0xb2

JSB394 Values

JW_vendorId	= 0x07c3
JW_productId	= 0x8383
JW_serialNumberIndex	= 5
JW_flashLedCmd	= 0xb9
JW_returnSize	= 8
JW_ReadInCmd	= 0xb2

JSB503 Values

JW_vendorId	= 0x07c3
JW_productId	= 0x8503
JW_serialNumberIndex	= 6
JW_returnSize	= 8
JW_numCountBytes	= 3
JW_readCount	= 0xd2
JW_resetCount	= 0xd3
JW_setEdgeMode	= 0xd4
JW_readEdgeMode	= 0xd5

JSB624 Values

JW_vendorId	= 0x07c3
JW_productId	= 0x0624
JW_serialNumberIndex	= 5
JW_returnSize	= 8
JW_flashLedCmd	= 0xb1
JW_readADC	= 0xc4
JW_setPGA	= 0xc5

returnData byte 5 is full scale voltage (FS)

Output code

16 bit version
7fffh = FS
0 = 0
8000h = -FS

12 bit version
7ff0h = FS
0 = 0
8000h = -FS

returnData byte 7 gain for JSB624 is as follows

0 = Gain of 2/3
1 = Gain of 1
2 = Gain of 2
3 = Gain of 4
4 = Gain of 8
5 = Gain of 16

SSB118 Values

JW_vendorId	= 0x07c3
JW_productId	= 0x1218
JW_serialNumberIndex	= 3
JW_flashLedCmd	= 0x01
JW_returnSize	= 8
JW_numRelayCmd	= 0x03
JW_byteForNum	= 0
JW_clearRelayCmd	= 0x07
JW_setRelayCmd	= 0x06
JW_relayStateCmd	= 0x04

To control the usb port and simulate an actual plug action, the signal lines should be closed first, then the power.
Relay 1 is the signal
Relay 2 is the power

WDT205 Values

JW_vendorId	= 0x07c3
JW_productId	= 0x2205
JW_serialNumberIndex	= 5
JW_returnSize	= 8
JW_setWdt	= 0xb1
JW_enableWdt	= 0xb2
JW_clrWdt	= 0xb3
JW_resetWdt	= 0xb4
JW_readWdt	= 0xb5

8. Revision History

Version 1.00 Initial Release
Version 1.01 Added WDT205 Information
Version 1.02 Added JSB503 Information
Version 1.03 Added JSB310 Information
Version 1.04 Added JSB2851 Information
Version 1.05 Added CSB147x Information
Version 1.06 Added JSB342, JSB383, JSB394
Version 1.07 Corrected typos
Version 1.08 Added JSB274, SSB118
Version 1.09 Added analog, revised layout
Version 1.10 Added DSB1616
Version 1.11 Added JSB2944
Version 1.12 Added JSB624
Version 1.13 Added JSB273