ASI Arduino Library User Guide

Rev	Change
Jan 2011	Preliminary
Jen 2011	Updated fine tuned Baud rate and altered constructor

Introduction

The ASI Library is common to all ASI interface type devices supplied by www.byvac.com these have the advantage of being able to be connected directly to a COM port if required and also multiple devices can exist on the same serial bus as they are addressable.

The address is settable by the user and stored on internal EEPROM and can be changed by the user.

All of the software presented here was written using a Arduino Nano fitted with an ATmega328. Other types of processor or Arduino boards have not been tested.

Bserial Class

This is a cut down version of NewSoftSerial as the BV devices do not need such sophistication and it cuts the size down quite a bit. The ASI devices do however need buffered serial and so the SoftSerial library will not do.

You can substitute this library for the NewSoftSerial if required but it will need all of the puts and putch changing for print.

The class uses some C code that does all of the work and there is a good opportunity to write the put and get in assembler thus possibly improving the speed. It has a 32 byte circular buffer that will stop receiving when it becomes full.

BSerial(receivePin, transmitPin);

Sets the tx and rx pins.

baud(rate);

Sets baud rate and clears buffer. This works up to 38400 on the Nano but that is without doing anything else. The rate need not be a fixed standard rate. For your system it may be advisable to adjust it slightly – see appendix.

handshake(uint8_t rtsPin, uint8_t ctsPin);

This method can be left out and the hardware handshaking will be ignored. **RTS** is an output that is set normally high. If the buffer should become full then the line goes low. This will indicate to the connected device to stop sending bytes until it goes high again. The **CTS** pin is an input and if activated must be held high by the connected device, if the line is put low the Arduino will stop sending characters and wait for the line to be put high again. It is more normal for this pin alone to be active as the connected device is usually slower then the Arduino.

If only one is required then set the other to 0xff. Foe example if pin 9 was used for the CTS the method would look like this:

```
handshake(0xff, 9);
```

flush();

Resets the buffer back to 0, effectively empties it.

putch(char c)

Puts a single character to the serial output.

unsigned char puts(char *s)

Puts a string to the serial interface and returns the number of bytes sent. This is limited to a maximum of 255 bytes, the method will return after that.

unsigned char buffer();

Returns the number of characters waiting in the buffer.

char getch();

Gets a single character from the buffer. NOTE that this method will return -1 if there are no characters in the buffer.

ASI Class

This class is common to all ASI devices. These devices share a common command set and it is presented in the following methods. The class can be used stand alone but it is normally incorporated into a specific device class. When incorporated in this way the methods here are available for use.

Methods inherited from BSerial:
baud(rate)
handshake(uint8_t rtsPin, uint8_t ctsPin)
flush()
putch(char c)
unsigned char puts(char *s)
unsigned char buffer()
char getch()

ASI(rxPin, txPin);

The constructor provides a connection to BSerial.

begin();

Initialises the connected devices by sending 3 carriage returns and inverting the output which is required for this serial interface. The device on the bus will not work properly without this method being called.

resetall(char *dev, char max)

This will reset all of the devices on the bus regardless of their address.

devices(char *dev, char max)

Returns a list of device addresses on the bus as a string. This can be very useful for checking what should be on there or be used in an automated system. The method requires a buffer to put the devices in, max is the size of the buffer. As an example if there are three devices on the bus with addresses a, k and p then the returned string would be "a>k>p>"

The above commands will work on all of the devices attached to the bus. The following commands require a device address as they refer to a particular device. So that more than one device can be referred, all the following commands require DEVadr *which is a character representing the device address, i.e. 'a'.*

char wait(char *inp, char term, char max);

When a command is sent to the ASI device, in most cases the device will return with a prompt '>'. Because of this it is easy to tell when the ASI device is ready for another command. This method will wait for a particular character in the input stream (term) and return when it receives it. There is also a timeout of about 1 second if the character is never received. This version will return all of the characters received from sending the command to receiving the ‘term’. To safeguard the buffer max should be set to the size of the provided buffer. As an example to get the version of an ASI device the command is “aV\r”, assuming the device address is ‘a’. To capture this with the wait method:

```
wait(b, ‘>’, 5);
```

The buffer b will now have a string in it that represents the version.

char wait();

This is as above but will always wait for ‘>’ and it throws away any received bytes.

changeaddress(char DEVadr, char newadr)

Changes the device address. The change is stored in EEPROM so do not do this each time the software starts, it only needs doing once.

eepromwrite(char DEVadr, char addr, char *s)

Writes a string of bytes to the EEPROM, the string should be 0 terminated. This does mean that 0 cannot be written to the EEPROM. The command is primarily intended for text.

ackoff(char DEVadr)

Turns the ACK mechanism off – this is not recommended as without it this library will not work.

erroroff(char DEVadr)

Turns off error reporting. The library still works with this switched on.

eepromread(char DEVadr, char addr, char bytes, char *read, char max);

Reads the EEPROM at a given address for the specified number of bytes into a supplied buffer. Max is the size of the buffer that has been supplied.

deviceid(char DEVadr, char *id)

Returns the device ID, useful for automatically identifying which address belongs to which device. The value returned is a number. For example the BV4108 returns 4108. This is returned as a string in ‘id’. The size of the string buffer supplied should be at least 5 bytes.

macro(char DEVadr, char mon)

This turns on or off the macro which will be run at start up, see the datasheet for the device for more information about this. setting ‘mon’ to 1 will deactivate the macro, setting it to 0 will deactivate it.

reset(char DEVadr)

Resets the device

version(char *s)

Puts the firmware version as a sting in the supplied buffer. The size of the string buffer supplied should be at least 5 bytes.

Appendix Baud Rate - Example

The baud rate is simply a delay and for this software serial class there is no need to fix it at a standard rate. For whatever reason the baud rate on the Arduino and the target system may vary. If reliability is a problem then adjust the rate up or down. To help with this use this small program:

```
#include <BSerial.h>
```

```
#define rxPin 5
#define txPin 4

BSerial bs(rxPin, txPin);

void setup()  {
}

void loop() {
char b[10];
unsigned long j;
  for(j=8000;j<11000;j+=40) {
    delay(250);
    bs.baud(j);
    itoa(j,b,10);
    bs.puts("\n");bs.puts(b);
  }
  while(1); // stop
}
```

Connect a terminal up to pins 4 and 5 in this case, set eh Baud rate 9600, run the program and watch the output. Initially rubbish should be displayed but then a series of numbers then rubbish again. To get the optimum Baud rate take the first and last number that was printed out okay and average it. In the example on this system this was:

(9100 + 10200) / 2 = 9650

So the best baud rate would be 9650, so 9600 is well within the range. This is just testing the output. The input will follow closely but at higher baud rates only the output may work.

Simple Echo

```
  bs.puts("\nReady ");
  while(1) {
    c=bs.getch();
    if(c!=-1) bs.putch(c);
  }
```

When connected to a terminal the above will echo back any character typed. getch() return -1 if there are no keys in the buffer. The buffer() methods could also be used instead to see the number of characters waiting.

Appendix ASI Example

To try this you will of course need an ASI device connected to pins 4 and 5 of the Arduino. *Change the example if you wish to use other pins*. See the ASI web site and relevant data sheets for more information about ASI devices: http://www.asi.byvac.com

The only method that will sow anything without becoming device specific is getting the addresses of the devices connected to the bus as follows:

```
#include <BSerial.h>
#include <ASI.h>

#define rxPin 5
#define txPin 4

ASI asi(rxPin, txPin);

void setup()  {
  Serial.begin(9600); // to show results from example
  asi.baud(9600);
  asi.begin();
}

void loop() {
char b[20];
  Serial.print("\nDevices ");
  asi.devices(b,19);
  Serial.println(b);
  while(1); // stop
}
```

For this to work an ASI device has to be connected as above and also a terminal should be connected to the TX and RX (UART) pins. If two devices are connected with addresses a,b then the output is:

```
Devices a>b>
```