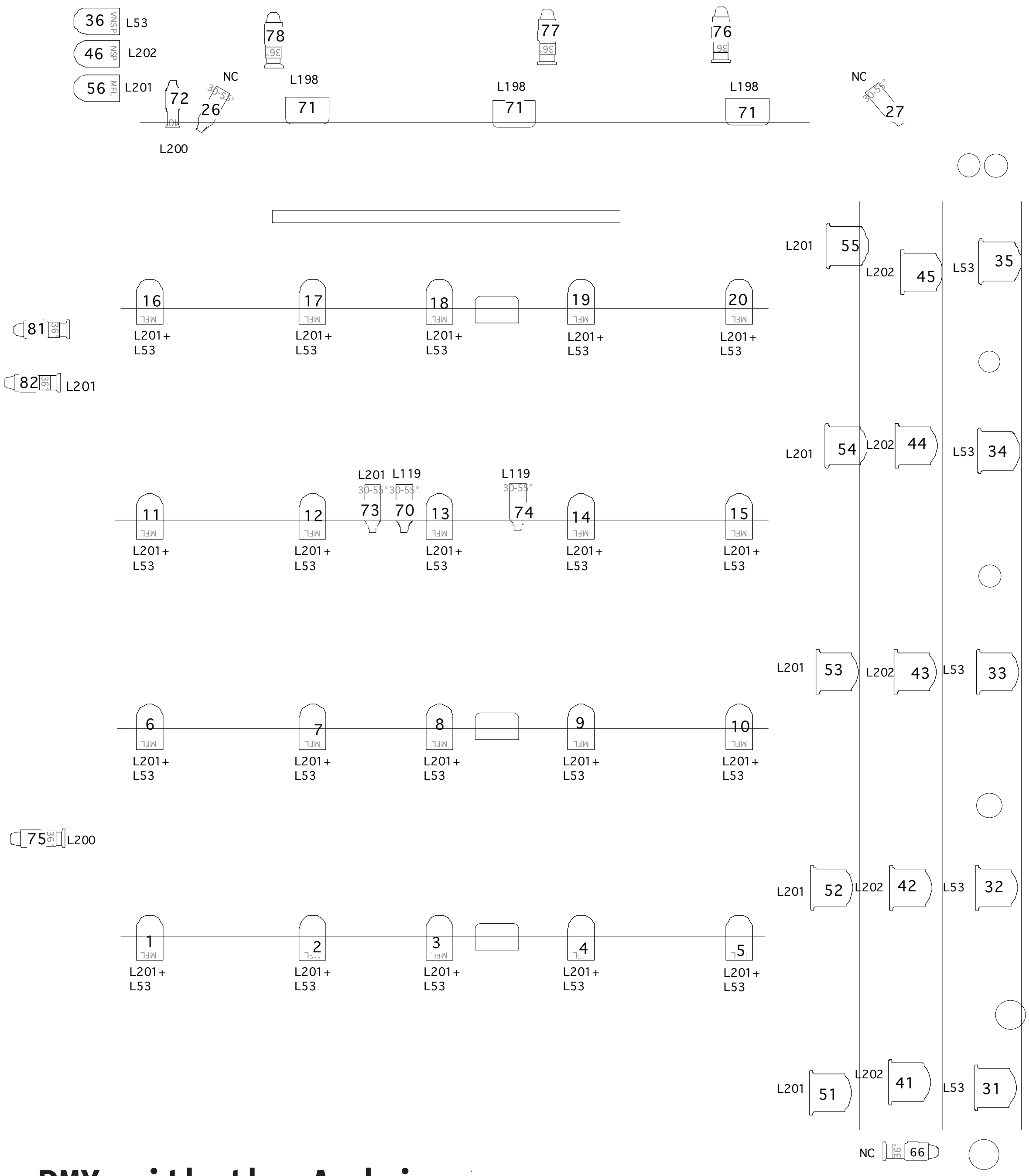
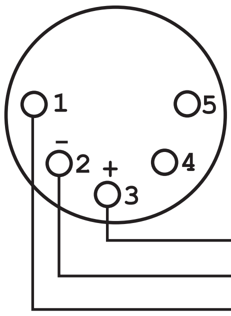


>>> sending DMX with the Arduino

```
*(cleft) 2006 by Tomek Ness and D. Cuartielles
* K3 - School of Arts and Communication
* fhp - University of Applied Sciences
* <http://www.arduino.cc>
* <http://www.mah.se/k3>
* <http://www.design.fh-potsdam.de>
*
* @date: 2006-09-30
* @idea: Tomek Ness
* @code: D. Cuartielles and Tomek Ness
* @acknowledgements: Johny Lowgren for his DMX devices
*
```



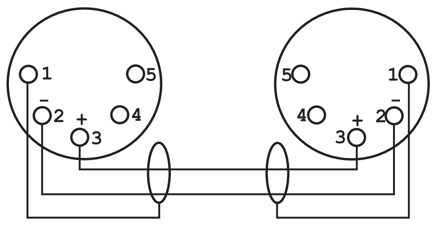
DMX 512

-DMX512 (Digital MultipleXed) is an agreement over the connection between lighting controllers, dimmers, scrollers, scanners, etc..

-The voltage between the conductors of the cable is either approximately +2,5[v] or approximately -2,5[v]. If the voltage is for 4[us] long positive, then a ‚ 1 ‚ is transferred. If the voltage is for 4[us] long negative, then ‚ 0 ‚ istransferred.

-The elektronic foundation for DMX is RS 485.

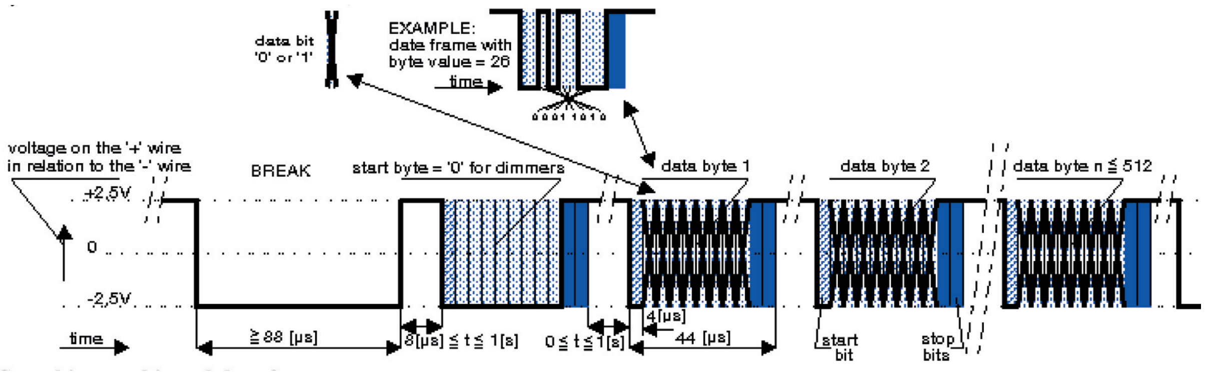
-Which connectors and how to connect them?
If connectors are used, then it must be 5-Pol XLR types.
Pin 1 = signal reference = cable shield
Pin 2 = signal inversion = ‚ - ‚
Pin 3 = signal = ‚ + ‚
Pin 4 = optional (e.g. acknowledgment)
Pin 5 = optional (e.g. acknowledgment)



The structure of the DMX512 signals

The DMX512 signal consists of a repetitive bit stream, which is structured as follows (see also following figure):

The beginning of the bit stream is marked by the fact that during 88[us] a ‚ 0 ‚ is transmitted. This mark in the DMX signal is called „break“. After the break signal a so called „start byte“ follows. This start byte determines the target from the following bytes. For the
dimmers the start byte consists of eight ‚0-s‘. The start byte extends the possibilities of the DMX512 protocol. After the start byte maximal 512 bytes will follow with information for the control channels. The sequence number of the bytes determines the channel number, for which the information is intended.



Start bit, stop bit and date frame

In the data flow of the ‚0-s‘ and ‚1-s‘ the separate bytes are to be differentiated by the following appointments:
A byte is preceded by a ‚0‘: this is called „start bit „
A byte is terminated by two ‚1-s‘: these are called „ stop bits „
With the receipt of the start bit always eleven bits „ are read „: these eleven bits form the date frame. The transmission of a date frame lasts 44[us]. The time between the transmission of the successive bytes and break is arbitrary, within certain boundaries. DMX512 is called therefore an asynchronous data protocol.

Shifts data in DMX format out to DMX enabled devices it is extremely restrictive in terms of timing. Therefore the program will stop the interrupts when sending data.

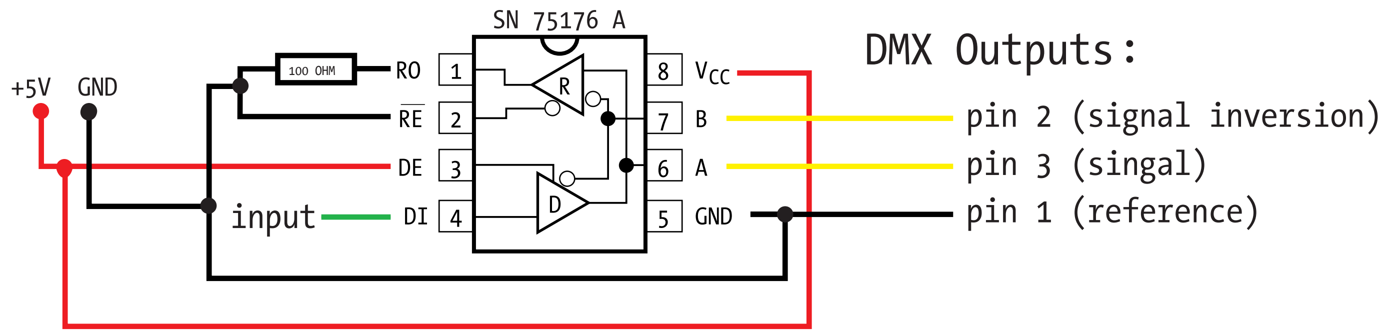
Almost all information about dmx 512 is copied from:
http://www.theater-technisch-lab.nl/dmxen.htm

In the case of Arduino

To send dmx from Arduino we use a driver block like MAX485 resp. a 75176.

To use a driver block like MAX485 resp. a 75176 is the comment way of transmitting and receiving dmx. You find this driver blocks in almost every dimmer/scroller etc.
The 75176 is just a cheaper version of the Max 485.

The MAX-485



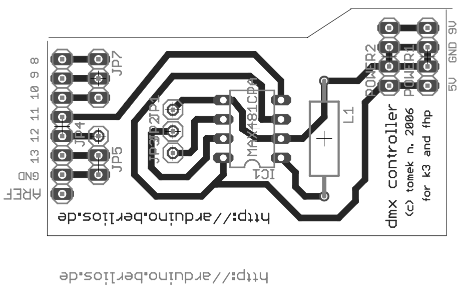
PIN #	NAME	DESCRIPTION
1	RO	Receiver Output. If the receiver output is enabled ($\overline{RE}$ low), then if A > B by 200mV, RO will be high. If A < B by 200mV, then RO will be low.
2	$\overline{RE}$	Receiver Output Enable. A low enables the receiver output, RO. A high input forces the receiver output into a high impedance state.
3	DE	Driver Outputs Enable. A high on DE enables the driver output. A and B, and the chip will function as a line driver. A low input will force the driver outputs into a high impedance state and the chip will function as a line receiver.
4	DI	Driver Input. If the driver outputs are enabled (DE high), then a low on DI forces the outputs A low and B high. A high on DI with the driver outputs enabled will force A high and B low.
5	GND	Ground Connection.
6	A	Driver Output/Receiver Input.
7	B	Driver Output/Receiver Input.
8	V_{CC}	Positive Supply; 4.75< V_{CC} < 5.25

LTC485 Transmitting

INPUTS			LINE	OUTPUTS	
$\overline{RE}$	DE	DI	CONDITION	B	A
X	1	1	No Fault	0	1
X	1	0	No Fault	1	0
X	0	X	X	Z	Z
X	1	X	Fault	Z	Z

Dmx transmitting Shield

Shields are boards to be mounted on top of the Arduino board and that extend the functionality of Arduino to control different devices, acquire data, etc.



code example

```
/* DMX Shift Out for arduino - 004 and 005
 * -------------
 *
 * Shifts data in DMX format out to DMX enabled devices
 * it is extremely restrictive in terms of timing. Therefore
 * the program will stop the interrupts when sending data
 *
 * The elektronic foundation for DMX is RS 485, so you have to use
 * a MAX-485 or a 75176.
 *
 * wirring for sending dmx with a MAX-485

    1 - RO  - Receiver Output --- set to ground with a 100 ohm resistor
    2 - RE  - Receiver Output Enable -- set to ground
    3 - DE  - Driver Output Enable -- set to 5v
    4 - DI  - Driver Input -- Input from Arduino
    5 - GnD - Ground Connection -- set to ground -- refence for the DMX singal --- (DMX pin 1)
    6 - A   - Driver Output / Receiver Input -- DMX Signal (hot)------------------ (DMX pin 3)
    7 - B   - Driver Output / Receiver Input -- DMX Signal inversion ( cold)------ (DMX pin 2)
    8 - Vcc - Positive Supply -- 4,75V < Vcc < 5,25V

 * Every dmx packet contains 512 bytes of information (for 512 channels).
 * The start of each packet is market by a start byte (shiftDmxOut(sig,0);),
 * you should always send all 512 bytes even if you don*t use all 512 channels.
 * The time between every dmx packet is market by a break
 * between 88us and 1s ( digitalWrite(sig, LOW); delay(10);)
 *
 * (cleft) 2006 by Tomek Ness and D. Cuartielles
 * K3 - School of Arts and Communication
 * fhp - University of Applied Sciences
 * <http://www.arduino.cc>
 * <http://www.mah.se/k3>
 * <http://www.design.fh-potsdam.de>
 *
 * @date: 2006-09-30
 * @idea: Tomek Ness
 * @code: D. Cuartielles and Tomek Ness
 * @acknowledgements: Johny Lowgren for his DMX devices
 *
 */


int sig = 11;           // signal (hot / dmx pin 3)

int count = 0;
int swing = 0;
int updown = 0;


/* Sends a DMX byte out on a pin.  Assumes a 16 MHz clock.
 * Disables interrupts, which will disrupt the millis() function if used
 * too frequently. */


void shiftDmxOut(int pin, int theByte)
{
    int theDelay = 1;
    int count = 0;
    int portNumber = port_to_output[digital_pin_to_port[pin].port];
    int pinNumber = digital_pin_to_port[pin].bit;
    // the first thing we do is to write te pin to high
    // it will be the mark between bytes. It may be also
    // high from before
        _SFR_BYTE(_SFR_IO8(portNumber)) |= _BV(pinNumber);
    delayMicroseconds(20);

    // disable interrupts, otherwise the timer 0 overflow interrupt that
    // tracks milliseconds will make us delay longer than we want.
    cli();

        // DMX starts with a start-bit that must always be zero
        _SFR_BYTE(_SFR_IO8(portNumber)) &= ~_BV(pinNumber);
        //we need a delay of 4us (then one bit is transfert)
        // at the arduino just the delay for 1us is precise every thing between 2 and 12 is jsut luke
        // to get excatly 4us we have do delay 1us 4 times
        delayMicroseconds(theDelay);
        delayMicroseconds(theDelay);
        delayMicroseconds(theDelay);
        delayMicroseconds(theDelay);

         for (count = 0; count < 8; count++) {

                if (theByte & 01) {
                  _SFR_BYTE(_SFR_IO8(portNumber)) |= _BV(pinNumber);
                }
               else {
                  _SFR_BYTE(_SFR_IO8(portNumber)) &= ~_BV(pinNumber);
                }

              delayMicroseconds(theDelay);
              delayMicroseconds(theDelay);
              delayMicroseconds(theDelay);
              delayMicroseconds(theDelay);
               theByte>>=1;
                }

    // the last thing we do is to write the pin to high
    // it will be the mark between bytes. (this break is have to be between 8 us and 1 sec)
        _SFR_BYTE(_SFR_IO8(portNumber)) |= _BV(pinNumber);

    // reenable interrupts.
    sei();
}
void setup() {
  pinMode(sig, OUTPUT);
  digitalWrite(13, HIGH);
}
void loop()  {

   // sending the break (the break can be between 88us and 1sec)
   digitalWrite(sig, LOW);
   delay(10);
   //sending the start byte
   shiftDmxOut(sig,0);
   //sending the 512 bytes for the channels
   shiftDmxOut(sig, 150); //1
   shiftDmxOut(sig, 150); //2
   shiftDmxOut(sig, 150); //3
   shiftDmxOut(sig, 150); //4
   for (count = 1; count<=508; count++){
     shiftDmxOut(sig, 0);
   }
 }
```