Беспроводной DMX512 приемник/передатчик

Общая информация

В данной статье я расскажу вам как создать передатчик и приемник DMX512 сигнала. Выложена принципиальная схема, схема платы, и вся разводка. Апаратная часть построена на микроконтролере avr ATMEGA 328 от компании Atmel. Приемник и передатчик могут передавать все 512 каналов управления. Без антенны базовая конфигурация работает до 100 метров в прямой видимости*(сможет работать до 1км при установке антенны)*

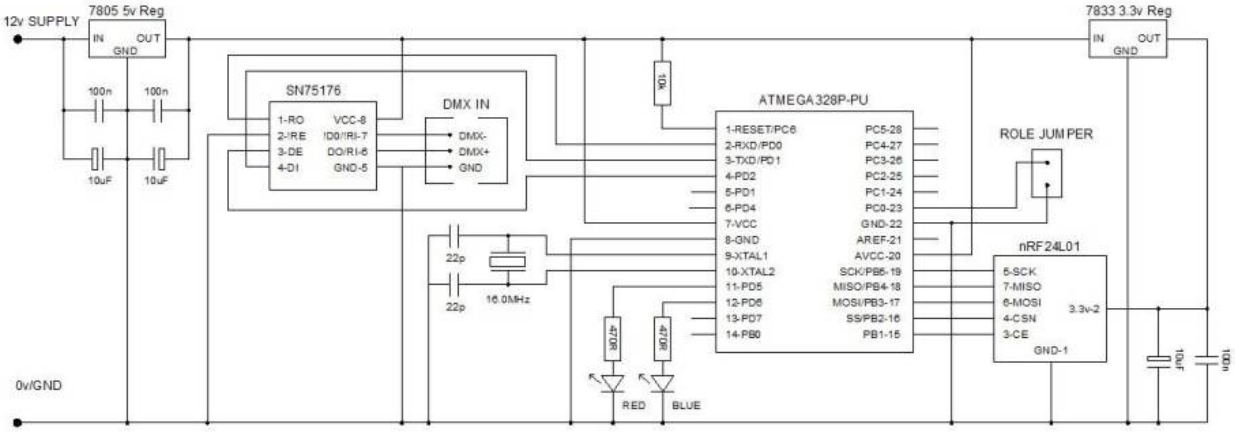
Элементы используемые в схеме

Сборка состоит из микроконтролера

- NRF24L01 два модуля.
- 12v Вход (5v и 3.3v регулятор напряжения)
- 2 x XLR вход и выход(папа и мама)
- 2 x Светодиода(для индикации)
- ATMEGA328p-pu микроконтроллер.
- 3 x 100ом резисторы.
- 1 x 10кОМ резистор.
- 3 x 10uF конденсаторы.
- переключатель каналов, джампер.
- 16mHZ кварц.

Video

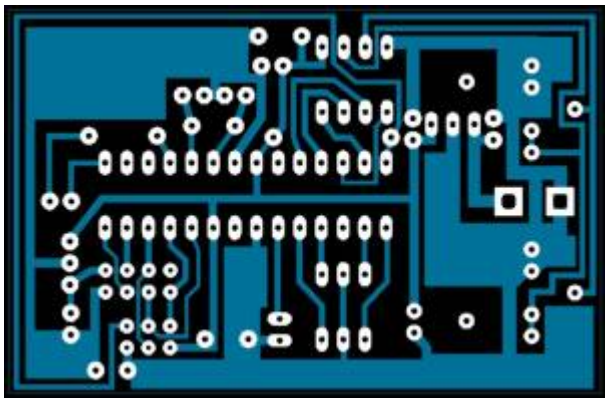
Схема подключения

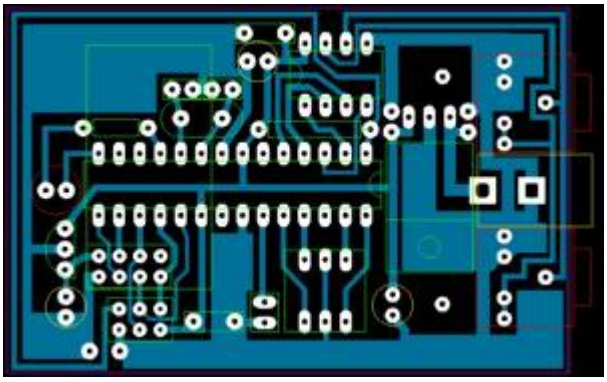


Беспроводной модуль в сборе

Для подключения радиомодуля использованы ножки аппаратного SPI микроконтроллера, поэтому разъемы подключения модуля и подключения программатор дублируют друг друга. Это сделано, чтобы удобней было прошивать микроконтроллер на отладночной платке, например, если использовать программатор который подает на схему 5 вольт, а для NRF24L01 это слишком большое напряжение. Чтобы перепрошить управляющий микроконтроллер, достаточно выдернуть трансивер с платы, перепрошить и всунуть его обратно - без лишней возни с перепайкой.

Готовая плата





*Существуют ситуации, когда где DMX кабель невозможно протянуть, или использование его просто не практично. Этот проект реализует беспроводную передачу сигнала DMX в диапазоне 2,4 ГГц, используя платформу Arduino и популярного модуля приемопередатчика NRF24L01 + из нордического полупроводника. ==== Протокол передачи ==== Прием и передача по протоколу DMX осуществляется на контроллере ATmega с помощью библиотеки DMXSerial. Реализация использует прерывания и является хорошей базой для реализации второго протокола на одном чипе. 2,4 Радиомодуль подключен к плате Arduino или микроконтроллеру ATmega с помощью интерфейса SPI. Библиотека для использования чипа находится здесь https://github.com/gcopeland/RF24 . После инициализации библиотеки можно передавать и принимать данные, используя прилагаемые функции библиотеки. Библиотека передает все команды и данные микросхемы nRF24L01 + также будет обрабатывать все детали передачи в фоновом режиме. ===== Скетч для микроконтроллера ===== <note warning>**ВНИМАНИЕ**: перед компиляцией и заливкой скетча в микроконтроллер, необходимо установить и добавить в компилятор библиотеки nRF24L01 и RF24</note> <file>* FOR ATMEGA328 16mhz ONLY connection information... http://arduino-info.wikispaces.com/Nrf24L01-2.4GHz-HowTo SN75176 configuration : *DMXrx NRFtx : pin 2 = 0v | pin 3 = 0v* NRFrx DMXtx : pin 2 = 0v | pin 3 = 5v *pin 6 = DMX+ | pin7 = DMX-* Ensure 3.3v is used to supply NRF24L01 board !! the Nrf24L01 board can only transmit from channel 0 to 83 in the USA, therefore we shall limit it to 0-80 in steps of 5

#include <EEPROM.h> #include <SPI.h> #include <nRF24L01.h> *library: https://github.com/maniacbug/RF24 #include <RF24.h> #include <Wire.h> #include <DMXSerial.h>* library: http://www.mathertel.de/Arduino/DMXSerial.aspx *#define MAX_DMX_CHANNELS 512* full DMX #define BYTES_PER_PACKET 16 *usable bytes per packet #define PACKET_OVERHEAD 2* group and time stamp #define MAXPAYLOAD (BYTES_PER_PACKET+PACKET_OVERHEAD) *max payload size for nrf24l01 #define MAXGROUPS (MAX_DMX_CHANNELS/BYTES_PER_PACKET)* 32 groups of 16 channels = 512 DMX channels

16way hex channel selector - hex switch mounted with zero at top (see PCB layout) #define HEX_0 A4

#define HEX_1 A1 #define HEX_2 A2 #define HEX_3 A5 nRF24L01 control #define nRF_CE 9 *Chip enable #define nRF_CSN 10* Chip select not *LEDs #define LED_BLUE 6* BLUE PWM #define LED_RED 5 *RED PWM* sn76175 control #define DMX_NOT_EN 2 #define DMX_MODE 3 *other #define ROLE A0* if low (jumper on) then RX otherwise TX

RF24 radio(nRF_CE,nRF_CSN);

unsigned long RXtimer, flashTimer, taskTimer, lastFlash, refreshTimer, radioTimer, rePairTimer; uint64_t RXTXaddress, pairingAddress = 0xF0F0F0F0F0LL; uint16_t rfQuality; uint8_t payload[MAXPAYLOAD], shadow_DMX[MAX_DMX_CHANNELS]; uint8_t timeStamp, lastStamp, bestChannel; uint8_t HEXchannel, _HEXchannel, RXTXchannel, pairingChannel = 80; boolean role, _role, group_send;

void setup(void) {

```
// set DMX/rs485 interface
pinMode(DMX_NOT_EN, OUTPUT);
digitalWrite(DMX_NOT_EN, LOW);
pinMode(DMX_MODE, OUTPUT);
// set bicolour LED
pinMode(LED_BLUE, OUTPUT);
digitalWrite(LED_BLUE, LOW);
pinMode(LED_RED, OUTPUT);
digitalWrite(LED_RED, LOW);
// set up ROLE switch
pinMode(ROLE, INPUT_PULLUP);
// set up HEX channel select switch
pinMode(HEX_0, INPUT_PULLUP);
pinMode(HEX_1, INPUT_PULLUP);
pinMode(HEX_2, INPUT_PULLUP);
pinMode(HEX_3, INPUT_PULLUP);
if ( digitalRead(ROLE) ) { // jumper off : transmitter : initialise & read
EEPROM
  init_EEPROM();
}
radio.begin();
```

test nRF board, if it doesnt reply then flash LED indefinately, or until it does reply while (!radio.isPVariant()) { check if board connected

```
  if (digitalRead(ROLE)) { blue(); delay(500); off(); delay(500); }  //
Wireless TX : if no rf comms then flash BLUE 50%
  else { red(); delay(500); off(); delay(500); }  // Wireless RX : if no rf
comms then flash RED 50%
}
```

```
// generic radio stuff
radio.setAutoAck(false);
radio.setPayloadSize(MAXPAYLOAD);
radio.setPALevel(RF24_PA_HIGH);
```

```
radio.setDataRate(RF24_250KBPS);
radio.setRetries(0,0);
//radio.setDataRate(RF24_1MBPS);
//radio.setDataRate(RF24_2MBPS);
```

```
HEXchannel = HEXread();
```

if this is a transmitter, do a channel scan, transmit network RXTXaddress and clean RXTXchannel during rePair (every 2 secs) only send clean channel data if channel selector is set to zero (send 0xFF if channel other than zero : manualPair)

```
if ( digitalRead(ROLE) ) { // jumper off : transmitter
  bestChannelScan(); // only do channel scan if this is a transmitter -
bestChannel is required for manualPair and autoPair modes
  if (HEXchannel) { //
    bestChannelDisplay(); // if HEX switch not zero display best channel
number
  }
}
```

if this is a receiver, go into pairing mode and wait (indefinately) for pairing & channel data only use received (clean) channel data if channel selector is set to zero

```
else  {
  receivePairingAddress();
}
rePairTimer = flashTimer = taskTimer = millis(); // TXtimer =
DMXSerial.maxChannel(MAX_DMX_CHANNELS);
rfQuality = 0;
// read and initialise HEX switch and Role jumper
readPins(true);
```

} *end of init void loop(void) {* once per second, test for changes in HEX Switch and Role jumper (rx/tx)

```
if (millis() - taskTimer > 1000) {
  taskTimer = millis();
  readPins(false);
}
```

JUMPER OFF : DMX –> WIRELESS Tx LED Wireless TX : Solid BLUE, broken with very short flash of RED only if DMX data is being received.

```
if (digitalRead(ROLE)) { // if a wireless transmitter
// SEND OUT BURSTS OF RADIO DATA EVERY 30ms....
  for (uint8_t group = 0; group < MAXGROUPS; group++) { // scan through the
groups of DMX data, 30 bytes at a time
    uint16_t group_ptr = group*BYTES_PER_PACKET; //30; // create group
pointer for array
    if (millis() - refreshTimer > 1000) { // allow ALL radio data (full DMX
array) to be send once per second, regardless of changes
```

```
      group_send = true; // force ALL send
      refreshTimer = millis();
    } //if refresh timer
    else {
      group_send = false; // preset flag to false, only set it if there has
been a data change since last time
    } // not refresh timer
    for (uint8_t chan = 1; chan < BYTES_PER_PACKET+1; chan++) {
      if ( DMXSerial.read(group_ptr+chan-1) != shadow_DMX[group_ptr+chan-1]
) { // compare test : current DMX against old DMX
        shadow_DMX[group_ptr+chan-1] = DMXSerial.read(group_ptr+chan-1); //
if changed, update shadow array of DMX data and payload
        group_send = true; // set flag so that THIS group packet gets sent
      } // if data compare
      payload[chan+1] = DMXSerial.read(group_ptr+chan-1); // ensure ALL up-
to-date data gets through on this packet
    } // for chan
    if (group_send) { // only send the data that has changed, any data
change in a group will result in whole group send
      payload[0] = group; // set first byte to point to group number (groups
of 30 bytes)
      payload[1] = timeStamp++; // second byte helps us monitor consistency
of reception at receiver with a continuous frame counter
      radio.write( payload, sizeof(payload) ); // dump payload to radio
    } // if group_send
  } // for group
  if ( (DMXSerial.noDataSince() < 2000) && (millis() > 1000) ) { // if DMX
has been received within the last 2 seconds make LED flash
    if (!(millis() & 0b1111000000)) {
      red();
    }
    else { // SIMPLE DMX FLASHER
      digitalWrite(LED_RED,0);
      analogWrite(LED_BLUE,5); // low level Blue
    }
  }
  else { // no DMX data present, no flash
    digitalWrite(LED_RED,0);
    analogWrite(LED_BLUE,5);
  }
  // trigger rePair information send, just once every 2 seconds
  if (millis() - rePairTimer > 2000) {
    rePairTimer = millis();
    rePairTX(); // single burst of pairing information on pairing channel
using pairing network address
  }
} // if role
```

JUMPER ON WIRELESS –> DMX *LED Wireless RX : Solid RED, broken with flash of BLUE if wireless is*

being received (channel match) the gaps between the BLUE flashes are dependant on consistency of consecutive packets, relating to timeStamp

```
else {
  if (millis() - RXtimer > 20000) { // after 20 seconds without receving any
radio data, try and rePair
    receivePairingAddress();
  }
  else if (millis() - RXtimer > 2000) { // after 2 seconds of no radio data,
stop blinking
    rfQuality = 0;
  }
  if ( radio.available() ) {
    if (!rfQuality) {
      rfQuality = 50; // just to show that channel and address are locked on
!
    }
    RXtimer = millis();
    radio.read( payload, sizeof(payload) ); // get data packet from radio
    if ( payload[1] == (lastStamp+1) ) { // check for continuous timestamps,
increase quality indicator if good
      if (rfQuality < 255) { rfQuality++; }
    }
    else { // if incontinuous, reduce quality indicator/LED brightness
     if (rfQuality > 50) { rfQuality--; }
    }
    lastStamp = payload[1]; // reset last time stamp to current for checking
next time
    for (uint8_t i = 0; i <BYTES_PER_PACKET; i++) {
      DMXSerial.write((BYTES_PER_PACKET*payload[0])+i, payload[i+2]); //
spill radio data into dmx data array
    }
  } // if radio avail
  lastFlash = millis() - flashTimer;
  if (lastFlash < 64) {
    digitalWrite(LED_RED,0);
    analogWrite(LED_BLUE,rfQuality); // show rfQuality as brightness of BLUE
LED
  }
  else if (lastFlash >= 64) {
    digitalWrite(LED_BLUE,0);
    analogWrite(LED_RED,5); // switch back to RED at low level
  }
  if (lastFlash > 1024) {
    flashTimer = millis(); // reset timer after 1 second(ish)
  }
} // else
```

} *loop* read HEX switch and Role jumper, if there has been a change in settings then an update can be made void readPins(boolean initialise) {

```
_HEXchannel = HEXread(); // get temp HEXchannel
_role = digitalRead(ROLE); // get temporary Role
if (_role != role || initialise) { // has the ROLE changed ?? - initialise
will override any change in settings
  role = _role;
  if (role) { // JUMPER OFF : this is a wireless transmitter
    digitalWrite(DMX_MODE,LOW); // enable rs485 for input
    DMXSerial.init(DMXReceiver); // enable DMX to receive
    radio.openWritingPipe(RXTXaddress); // set network address
    radio.stopListening(); // start talking !
    digitalWrite(LED_RED,0);
    analogWrite(LED_BLUE,5); // low level BLUE to start...
  }
  else { // JUMPER ON : this is a wireless receiver
    digitalWrite(DMX_MODE,HIGH); // enable rs485 for output
    DMXSerial.init(DMXController); // enable DMX for output only
    radio.openReadingPipe(1,RXTXaddress); // set network address
    radio.startListening(); // start listening for data
    digitalWrite(LED_BLUE,0);
    analogWrite(LED_RED,5); // low level RED to start...
  }
}
if (_HEXchannel != HEXchannel || initialise) { // has the CHANNEL changed ?
- 'initialise' overrides any change in settings
  HEXchannel = _HEXchannel;
  rfQuality = 0; // reset this so that led stops flashing
  if (!role) {
    radio.stopListening(); // if this is a receiver...
  }
  if (HEXchannel) {
    RXTXchannel = HEXchannel*5; // create real channel number from HEX read
  }
  radio.setChannel(RXTXchannel); // set the channel
  if (!role) {
    radio.startListening(); // if this is a receiver...
  }
}
```

}

void red(void) {

```
digitalWrite(LED_BLUE,0);
digitalWrite(LED_RED,1);
```

} void blue(void) {

```
digitalWrite(LED_BLUE,1);
digitalWrite(LED_RED,0);
```

} void off(void) {

```
digitalWrite(LED_BLUE,0);
digitalWrite(LED_RED,0);
```

}

manualPair & autoPair information selecting HEXchannel 0 instigates «autoPair» mode on transmitter and receivers, therefore channel zero cannot be used in manualPair mode *The transmitter and receivers MUST be set to either ALL autoPair(0) or ALL manualPair(1-15) to work correctly* manualPair mode can only select RXTXchannels 5 to 75 (via hex switch positions 1-15) *manualPair & autoPair (on the Transmitter) has a '*bestChannelScan*' on power-up* autoPair : The best(quietest)channel will be transmitted with the network address on pairing *manualPair : The best(quietest)channel is shown by number of blinks (shown as 3 groups of blinks) (bestChannelDisplay)* The RXTXchannel number is 5 times that indicated - example 5 blinks = channel 60 *In manualPair, the transmitter and all receivers must be manually set to that channel via the hex switch* Transmitter : every 2 seconds, the transmitter sends its network address/pipe number as data on channel 80, using a a pairing address/pipe *Receivers : on power up, receiver(s) are in pairing mode, receiver goes to channel 80 and listens on the pairing address/pipe for the network address and channel number to go to* receivers change to the network channel and wait for DMX/Radio data to arrive. If any receiver loses pairing, after 20 seconds it will start to repair again.

void bestChannelScan(void) { uint8_t carriers[16], reps, i, best=200; for (reps=0; reps<200; reps++) { *scan all channels 200 times for (i=1; i<16; i++) { radio.setChannel(i*5);* scan from channel 5 to 75 in steps of 5

```
    radio.startListening();
    delayMicroseconds(128);
    radio.stopListening();
    if ( radio.testCarrier() )
      ++carriers[i]; // if there is any signal present, increase value in
array
  } // i
} // reps
for (i=15; i>0; i--) { //  find quietest channel, starting from highest
channel
  if (carriers[i] < best) {
    best = carriers[i]; // save quietest value
    RXTXchannel = i*5; //and save corresponding channel number
  }
}
```

}

BestChannel will only display if transmitter is not on HEX SWITCH 0 void bestChannelDisplay(void) { uint8_t i, x; for (x=0; x<3; x++) { for (i=0; i<RXTXchannel/5; i++) { blue(); delay(25); off(); delay(375); } delay(1000); } } read the value of the Hex Switch uint8_t HEXread(void) {

```
  return (!digitalRead(HEX_0)) | (!digitalRead(HEX_1) << 1) |
(!digitalRead(HEX_2) << 2) | (!digitalRead(HEX_3) << 3);
```

}

reguarily, the transmitter sends out a short burst of the config data on the pairing channel and using the pairing address void rePairTX(void) { radio.openWritingPipe(pairingAddress); radio.setChannel(pairingChannel); radio.stopListening(); payload[0] = (uint8_t) (RXTXaddress); payload[1] = (uint8_t) (RXTXaddress » 8); payload[2] = (uint8_t) (RXTXaddress » 16); payload[3] = (uint8_t) (RXTXaddress » 24); payload[4] = (uint8_t) (RXTXaddress » 32); if (HEXread()) { payload[5] = 0xFF; } else { payload[5] = RXTXchannel; } radio.write(payload, sizeof(payload)); dump pairing data to radio

```
radio.openWritingPipe(RXTXaddress);
radio.setChannel(RXTXchannel);
radio.stopListening();
```

}

void receivePairingAddress(void) {

```
uint8_t p, q;
radio.stopListening();
radio.openReadingPipe(1,pairingAddress);
radio.setChannel(pairingChannel);
radio.startListening();
digitalWrite(LED_BLUE,LOW);
while ( !radio.available() ) {
  analogWrite(LED_RED,p++); delay(2); // slowly pulse RED LED while awaiting
pairing connection
}
radio.read( payload, sizeof(payload) ); // get data packet from radio if
available
RXTXaddress = (uint64_t)(payload[0]) | (uint64_t)(payload[1] << 8) |
(uint64_t)(payload[2] << 16);
RXTXaddress |= (uint64_t)(payload[3] << 24) | (uint64_t)(payload[4] << 32);
// get RXTX address from payload
if ( payload[5] == 0xFF ) {
  RXTXchannel = HEXread()*5; // use channel from hex switch
}
else {
  RXTXchannel = payload[5]; // use best channel from scan
}
radio.stopListening();
radio.openReadingPipe(1,RXTXaddress);
radio.setChannel(RXTXchannel);
radio.startListening();
RXtimer = millis();
```

}

void init_EEPROM(void) { *do this only if a transmitter* if the EEPROM hasnt already been set (all bytes at 0xFF) then set an address using 2 random numbers to make up unique 40bit address *the address*

will be used in pairing. Once it has been set the first time it wont need to be set again if (digitalRead(ROLE)) { jumper off : transmitter

```
  if ( (EEPROM.read(0) == 0xff) && (EEPROM.read(1) == 0xff) ) {// check for
uninitialised/used eeprom memory
    randomSeed(analogRead(A3)); // seed from unused floating analog port
    EEPROM.write(0,random(255)); // unique lowest byte only
    EEPROM.write(1,random(255)); // initialise 40 bit address number into
eeprom, this makes up bytes 1-4 in address
  }
  RXTXaddress = (uint64_t) (EEPROM.read(0)) | (uint64_t)(EEPROM.read(1) <<
8) | (uint64_t)(EEPROM.read(1) << 16);
  RXTXaddress |= (uint64_t)(EEPROM.read(1) << 24) |
(uint64_t)(EEPROM.read(1) << 32); // reconstruct RXTXaddress from eeprom,
byte 0 is unique, bytes 1-4 are the same
}
```

}

</file> **#define ROLE A0** - Эта строка отвечает за прием и ли передачу. Если нужен приемник то A0, передатчик A1

Следите за правильным напряжением, радиомодули очень чувствительны, напряжение должно быть в пределах 3.3v