Project Report: **8-Channel Transmitter with Trims Using NRF24L01 and Arduino**

Title: 8-Channel Transmitter and Receiver with Trims Using NRF24L01 and Arduino

Name: Neelam Kishor Baishya
Roll No: 2114134
College: National Institute Of Technology , Silchar
Department: ECE

Abstract:

This project demonstrates the creation of an 8-channel radio transmitter using an Arduino Nano and NRF24L01 wireless module for controlling a variety of devices. The system incorporates trim functionality for fine adjustments to joystick control, using EEPROM to store trim values. The transmitter sends data over a 2.4 GHz frequency band, utilizing SPI communication to interact with the NRF24L01 module. The project provides a low-cost, customizable solution for remote control systems in robotics, drones, and other wireless communication applications.

Introduction:

Wireless communication systems are essential in various remote control applications, from drones to robotics. This project aims to design a reliable and customizable 8-channel transmitter with trim functionality, using the NRF24L01 transceiver module and an Arduino Nano. The trim functionality allows for precise adjustments of control channels, enhancing the user's ability to fine-tune joystick controls for different channels. The wireless system operates in the 2.4 GHz ISM band, providing long-range communication while maintaining power efficiency.

Materials and Components:

1. **Hardware:**
2. **Arduino Nano**: Serves as the core controller.
3. **NRF24L01 Module**: A 2.4 GHz transceiver for wireless communication.

4. **Potentiometers and Joysticks**: Used to control various channels.
5. **Trim Buttons**: Used to fine-tune control signals.
6. **EEPROM**: Used to store and retrieve trim values.
7. **Capacitors, Resistors, Switches, and LEDs**: For circuit stability and power regulation.

Software:

- **Arduino IDE**: Programming environment.
- **Libraries**:
  - `SPI.h`: For SPI communication.
  - `nRF24L01.h` & `RF24.h`: For interacting with the NRF24L01 module.
  - `EEPROM.h`: To store and retrieve trim settings.

Theory

NRF24L01 Transceiver:

The NRF24L01 operates at 2.4 GHz and supports data rates up to 2 Mbps with a maximum range of 100 meters, depending on the environment. It communicates via the SPI protocol with the Arduino, making it ideal for wireless applications like this transmitter.

Trim Functionality:

Trim buttons allow users to make minor adjustments to joystick positions without physically altering the hardware. This is useful in applications where joystick precision is crucial, like drone control.

EEPROM Storage:

Trim values are stored in the EEPROM, ensuring that they are retained even when the system is powered off. Each trim channel is read and written using specific memory addresses, as shown in the code.

Circuit Design

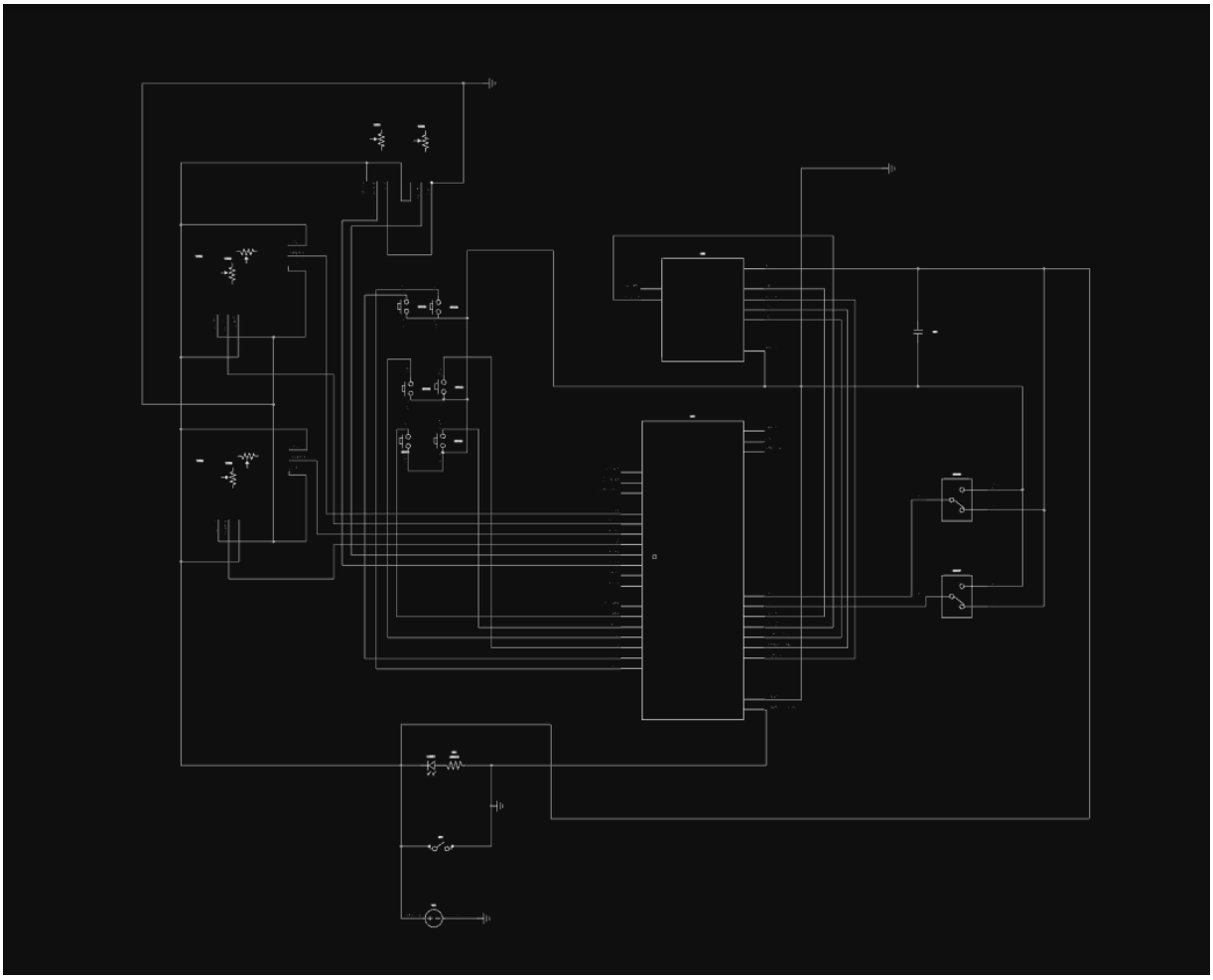
The circuit consists of an Arduino Nano connected to the NRF24L01 module via SPI communication. Joysticks and potentiometers control eight channels (throttle, yaw, pitch, roll, and auxiliary channels). Trim buttons adjust the values of specific channels by writing to the EEPROM.

The **circuit diagram** you provided (see attached image) illustrates the detailed connection between the Arduino Nano, NRF24L01 module, trim buttons, potentiometers, and other components.

Key Connections:

- **NRF24L01**:
  - CE to D9
  - CSN to D10
  - SCK to D13
  - MOSI to D11
  - MISO to D12
- **Joysticks and Potentiometers**: Connected to analog pins A0 to A5.
- **Trim Buttons**: Connected to digital pins D1 to D6.

FIG:CIRCUIT FOR 8 CHANNELS TRANSMITTER WITH TRIM:

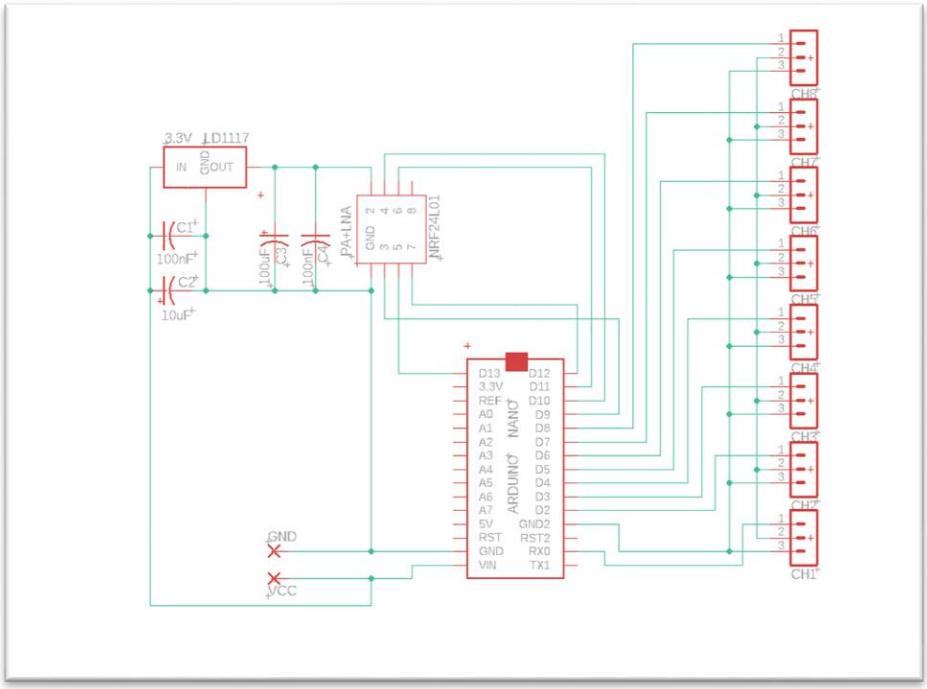
- 

Receiver Circuit Explanation

8. **Power Supply:**
    - The receiver operates at **3.3V**, which is supplied via the **LD1117 3.3V voltage regulator**.
    - The capacitors **C1 (100nF)** and **C2 (10uF)** are used to stabilize the input and output voltage of the voltage regulator. They help filter out any voltage spikes and maintain a smooth power supply.
    - Another capacitor **C3 (100uF)** further stabilizes the output voltage to ensure the NRF24L01 module gets a clean power supply.
9. **NRF24L01 Wireless Module:**
    - The NRF24L01 module is connected to the Arduino Nano via the SPI interface. The connections are as follows:
        - **CE (Pin 3)**: Connected to **D9** on the Arduino.
        - **CSN (Pin 4)**: Connected to **D10** on the Arduino.
        - **SCK (Pin 5)**: Connected to **D13** on the Arduino.
        - **MOSI (Pin 6)**: Connected to **D11** on the Arduino.
        - **MISO (Pin 7)**: Connected to **D12** on the Arduino.
    - The **GND** pin of the NRF24L01 is connected to the ground, and the **VCC** pin is powered by the 3.3V output from the LD1117 voltage regulator.
10. **Channels (CH1 to CH8):**
    - The receiver has **8 output channels**, each represented by the **CH1 to CH8** headers in the circuit. These channels are where the PWM signals will be output for controlling servos, motors, or other peripherals.
    - The output of each channel is connected to a separate pin on the Arduino Nano:
        - **CH1**: Connected to **D3**
        - **CH2**: Connected to **D5**
        - **CH3**: Connected to **D6**
        - **CH4**: Connected to **D7**
        - **CH5**: Connected to **A0**
        - **CH6**: Connected to **A1**
        - **CH7**: Connected to **A2**
        - **CH8**: Connected to **A3**
    - Each channel has three pins: **signal, power, and ground**. The signal pin receives the PWM output from the Arduino, and the power and ground pins are for powering connected devices like servos.
11. **Arduino Nano:**

- The **Arduino Nano** acts as the controller, receiving data from the NRF24L01 module and controlling the output signals on the 8 channels.
- The **D13** pin provides the clock signal (SCK) for the SPI communication with the NRF24L01 module.
- **D9, D10, D11, D12** handle the remaining SPI pins (CE, CSN, MOSI, MISO).
- The PWM outputs for the channels are generated based on the data received wirelessly.

8 CHANNELS RECEIVER (RX) CIRCUT:



8CHANNELS TRANSMITTER MODULE CODE:

#include <SPI.h>

#include <nRF24L01.h>

#include <RF24.h>

const uint64_t pipeOut = 000322;

```
RF24 radio(9, 10);

struct Signal {
  byte throttle;
  byte pitch;
  byte roll;
  byte yaw;
  byte aux1;
  byte aux2;
  byte aux3;
  byte aux4;
};

Signal data;

void ResetData() {
  data.throttle = 0;
  data.pitch = 127;
  data.roll = 127;
  data.yaw = 127;
  data.aux1 = 0;
  data.aux2 = 0;
```

```
  data.aux3 = 0;
  data.aux4 = 0;
}

void setup() {
  radio.begin();
  radio.openWritingPipe(pipeOut);
  radio.setChannel(100);
  radio.setAutoAck(false);
  radio.setDataRate(RF24_250KBPS);
  radio.setPALevel(RF24_PA_MAX);
  radio.stopListening();
  ResetData();
}

int Border_Map(int val, int lower, int middle, int upper, bool reverse) {
  val = constrain(val, lower, upper);
  if (val < middle)
    val = map(val, lower, middle, 0, 128);
  else
    val = map(val, middle, upper, 128, 255);
```

```
  return (reverse ? 255 - val : val);
}

void loop() {
  data.roll = Border_Map(analogRead(A3), 0, 512, 1023, true);
  data.pitch = Border_Map(analogRead(A2), 0, 512, 1023, true);
  data.throttle = Border_Map(analogRead(A1), 570, 800, 1023, false);
  data.yaw = Border_Map(analogRead(A0), 0, 512, 1023, true);
  data.aux1 = Border_Map(analogRead(A4), 0, 512, 1023, true);
  data.aux2 = Border_Map(analogRead(A5), 0, 512, 1023, true);
  data.aux3 = digitalRead(7);
  data.aux4 = digitalRead(8);

  radio.write(&data, sizeof(Signal));
}
```

8CHANNELS RECIEVER CODE:

```
#include <SPI.h>
#include <nRF24L01.h>
#include <RF24.h>
#include <EEPROM.h>
```

```
const uint64_t pipeOut = 000322;
RF24 radio(9, 10);

#define trimbut_1 1
#define trimbut_2 2
#define trimbut_3 3
#define trimbut_4 4
#define trimbut_5 5
#define trimbut_6 6

int tvalue1 = EEPROM.read(1) * 4;
int tvalue2 = EEPROM.read(3) * 4;
int tvalue3 = EEPROM.read(5) * 4;

struct Signal {
  byte throttle;
  byte pitch;
  byte roll;
  byte yaw;
  byte aux1;
```

```
  byte aux2;
  byte aux3;
  byte aux4;
};

Signal data;

void ResetData() {
  data.throttle = 512;
  data.pitch = 127;
  data.roll = 127;
  data.yaw = 127;
  data.aux1 = 0;
  data.aux2 = 0;
  data.aux3 = 0;
  data.aux4 = 0;
}

void setup() {
  radio.begin();
  radio.openWritingPipe(pipeOut);
```

```
 radio.setChannel(100);
 radio.setAutoAck(false);
 radio.setDataRate(RF24_250KBPS);
 radio.setPALevel(RF24_PA_MAX);
 radio.stopListening();
 ResetData();

 pinMode(trimbut_1, INPUT_PULLUP);
 pinMode(trimbut_2, INPUT_PULLUP);
 pinMode(trimbut_3, INPUT_PULLUP);
 pinMode(trimbut_4, INPUT_PULLUP);
 pinMode(trimbut_5, INPUT_PULLUP);
 pinMode(trimbut_6, INPUT_PULLUP);

 tvalue1 = EEPROM.read(1) * 4;
 tvalue2 = EEPROM.read(3) * 4;
 tvalue3 = EEPROM.read(5) * 4;
}

int Border_Map(int val, int lower, int middle, int upper, bool
reverse) {
```

```
  val = constrain(val, lower, upper);
  if (val < middle)
    val = map(val, lower, middle, 0, 128);
  else
    val = map(val, middle, upper, 128, 255);
  return (reverse ? 255 - val : val);
}

void loop() {
  if (digitalRead(trimbut_1) == LOW && tvalue1 < 630) {
    tvalue1 = tvalue1 + 15;
    EEPROM.write(1, tvalue1 / 4);
    delay(130);
  }
  if (digitalRead(trimbut_2) == LOW && tvalue1 > 280) {
    tvalue1 = tvalue1 - 15;
    EEPROM.write(1, tvalue1 / 4);
    delay(130);
  }

  if (digitalRead(trimbut_3) == LOW && tvalue2 < 630) {
```

```
  tvalue2 = tvalue2 + 15;
  EEPROM.write(3, tvalue2 / 4);
  delay(130);
 }
 if (digitalRead(trimbut_4) == LOW && tvalue2 > 280) {
  tvalue2 = tvalue2 - 15;
  EEPROM.write(3, tvalue2 / 4);
  delay(130);
 }

 if (digitalRead(trimbut_5) == LOW && tvalue3 < 630) {
  tvalue3 = tvalue3 + 15;
  EEPROM.write(5, tvalue3 / 4);
  delay(130);
 }
 if (digitalRead(trimbut_6) == LOW && tvalue3 > 280) {
  tvalue3 = tvalue3 - 15;
  EEPROM.write(5, tvalue3 / 4);
  delay(130);
 }
```

```
  data.roll = Border_Map(analogRead(A3), 0, tvalue1, 1023, true);
  data.pitch = Border_Map(analogRead(A2), 0, tvalue2, 1023, true);
  data.throttle = Border_Map(analogRead(A1), 570, 800, 1023, false);
  data.yaw = Border_Map(analogRead(A0), 0, tvalue3, 1023, true);
  data.aux1 = Border_Map(analogRead(A4), 0, 512, 1023, true);
  data.aux2 = Border_Map(analogRead(A5), 0, 512, 1023, true);
  data.aux3 = digitalRead(7);
  data.aux4 = digitalRead(8);

  radio.write(&data, sizeof(Signal));
}
```

Testing and Results

The transmitter was tested using a corresponding NRF24L01 receiver connected to an Arduino. Various signals (throttle, yaw, pitch, roll, and auxiliary channels) were successfully transmitted over the 2.4 GHz band. The trim buttons effectively adjusted the signals, and the values were stored in EEPROM as expected.

Conclusion

The project successfully demonstrated an 8-channel wireless transmitter with trim functionality using the NRF24L01 and Arduino Nano. The system provides a cost-effective solution for remote

control applications, offering stability and customization through the use of trim buttons. Future improvements could include adding an LCD display for real-time data feedback and further optimizing power consumption for battery-operated systems.