Final Project Report of NRF24L01 Wireless Communication Module

Shaofeng Zhang, Yechun Ruan, Changzhen Zhang, Ziyue Pang

December 2020

1 Introduction

NRF24L01 is a single-chip wireless transceiver chip operating in the 2.4-2.5 GHz universal ISM band. This project uses NRF24L01 module to realize the wireless communication function between MCU. The purpose of this project is to understand and realize functions of each module and master embedded programming through STM32 software programming.

2 Scheme Analysis and Design Ideas

This project mainly includes three parts: wireless data transmission, instruction input of serial port debugging, and LCD screen effect display. We divided the three modules and implemented and tested them separately. For the wireless data transmission part, it includes connection establishment and disconnection as well as data sending and receiving. We connect the modules by establishing a finite state machine and realize the overall functional logic.

3 System Function

3.1 Implementation Introduction

- *Check Mode*: Detect whether there is 24L01 module in the current MCU. That is, by writing a 5 byte addresses and then reading them out, if there is no error in the data, there is 24L01 module. Otherwise, MCU does not contains 24L01 module;
- *Instruction Mode*: Input instruction through serial port. It includes modifying the set module address, setting the send and receive channel, and setting the timeout period three main functions. These data are the basic configuration of communication between two MCU.
- *Connection Mode*: The connection state of two MCU is established by sending and receiving the connection message. Click the button Key 1

to make the MCU in the state of establishing connection, then click the button Key 0 in the state of receiving connection. At this point, sender MCU sends connection request, receiver MCU receives connection request. If the connection message is successfully sent and received, the connection is established. Otherwise continue to send or receive messages until the request times out. When the two MCUs want to disconnect the established connection, click the button Key 0 to jump out of the connection state and enter the idle state.

- *transmission Mode*: The connection state is mainly implemented by heartbeat mechanism. At this point, the two MCU are in the state of connection. In order to ensure that the two MCU can communicate with each other at any time, the two MCU alternately send and receive special messages. If a MCU sent data successfully, it change state into the receving mode; if the MCU received data successfully, it change state into the sending mode. When the MCU wants to send chat messages, it can encode the chat messages to the sending special message. When the MCU receives the message, the header can be deleted to obtain the received chat messages.

Figure 1 shows the Finite state machine (FSM) of the Wireless communication program.

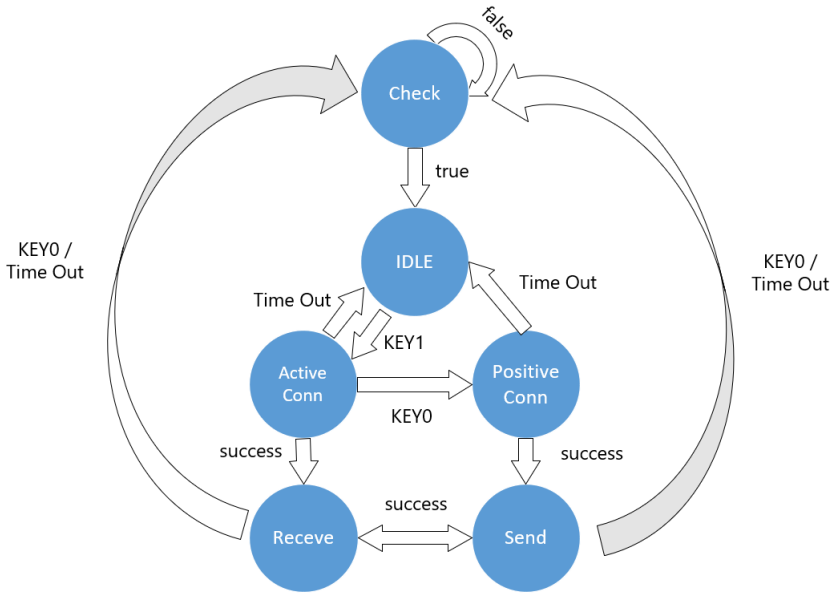

Figure 1: The FSM

3.2 Operating Instructions

After turning on the power, the LCD screen displays the initial interface, the upper display address of the sender and receiver and RF communication frequency.

The upper right rectangle color shows three connection status: red for the module does not exist, yellow for the communication status abnormal, green for the communication status normal.

The number shown on the left side of the rectangle is the upper limit of the connection time.

LED lights have three modes: not connected when the LED lights are not on, the red light blinks to indicate that the connection is being attempted, and the two LED lights blink alternately when the connection is successful.

Figure 2: Not connect

Press KEY1 to try to connect actively, and turn to connect passively after connection failure.

Chat window is displayed at the bottom of the LCD screen after successful connection, and chat messages can be sent by serial port. The sending message text color is brownish red and displayed on the right side; the receiving message color is black and displayed on the left side. New messages are displayed below the old messages.

Press KEY0 to actively disconnect and the message window disappears.

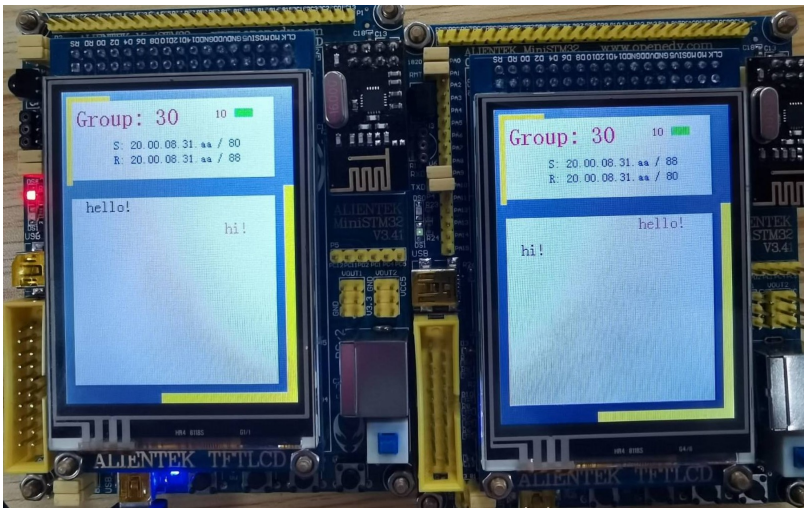

Figure 3: Connect successfully

```
DEBUG: Module pass check
DEBUG: data input from UART --> 4:80
DEBUG: Change send_ch to 80
DEBUG: data input from UART --> 3:88
DEBUG: Change recv_ch to 88
DEBUG: Try active conn
DEBUG: Try conn other.
DEBUG: Active request connection successful.
DEBUG: recv --> hello!
DEBUG: data input from UART --> hi!
DEBUG: send --> hi!
```

Figure 4: Content in serial connection assistant

The status can be changed by entering a specific form of message through the serial connection assistant, in the following form.

Modify message receive address:

1:xxxxxxxxxx(Ten digit hexadecimal number)

Modify message send address:

2:xxxxxxxxxx(Ten digit hexadecimal number)

Modify receive RF communication frequency:

3:xxx(Decimal number less than 255)

Modify send RF communication frequency:

4:xxx(Decimal number less than 255)

Modify the upper limit of the connection time:

5:xxx(Integer number in decimal, unit is second)

3.3 Code Design

Instruction Mode

```
void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart) {
  if (huart->Instance == USART1) {
    if (rxBuffer[0] == '\n') {
      inputData[uLength] = '\0';
      if (inputData[0] > '0' && inputData[0] < '9'
          && inputData[1] == ':') { //Instruction
        cmd_flag = inputData[0] - '0';
        strcpy(cmd_buff, inputData + 2);
        switch (cmd_flag) {
        case 1: //Change RX_ADDRES
          for (int i = 0; i < 5; ++i)
            RX_ADDRESS[i] = get_hex_num(cmd_buff[2 * i])
                << 4 + get_hex_num(cmd_buff[2 * i + 1]);
          break;
        case 2: //Change TX_ADDRESS
          for (int i = 0; i < 5; ++i)
            TX_ADDRESS[i] = get_hex_num(cmd_buff[2 * i])
                << 4 + get_hex_num(cmd_buff[2 * i + 1]);
          break;
        case 3: //Change recv_ch
          recv_ch = atoi(cmd_buff);
          break;
        case 4: //Change send_ch
          send_ch = atoi(cmd_buff);
          break;
        case 5: // Change timeout limit
          TLE_SCALE = atoi(cmd_buff);
          break;
        default:
          break;
        }
      } else { // Chat messages
        strcpy(send_buff, inputData);
        send_flag = 1;
      }
      uLength = 0;
    } else { // Get instructions or chat msg
      inputData[uLength] = rxBuffer[0];
      ++uLength;
    }
  }
}
```

Connection Mode

```
void try_conn() {
  uint8_t tmp_buff[33] = "-\0";
  uint8_t nrf_state = 0;

  if (state == STATE_REQ_CONN) {  //Active conn
    NRF24L01_TX_Mode();

    for (int i = 0; i < 600; ++i) {
      nrf_state = NRF24L01_TxPacket(tmp_buff);
      HAL_Delay(5);
      ++counter;
      LED_control();

      if (nrf_state == TX_OK) {  //Active conn success
        state = STATE_TRY_RECV;
        conn_init_flag = 1;
        return;
      }
      if (state != STATE_REQ_CONN)
        break; // Quit active conn
    }
  }

  if (state == STATE_WAIT_CONN) { //Passive conn
    NRF24L01_RX_Mode();

    for (int j = 0; j < 2400; ++j) {
      nrf_state = NRF24L01_RxPacket(tmp_buff);
      HAL_Delay(5);
        ++counter;
      LED_control();

      if (nrf_state == 0) {
        state = STATE_TRY_SEND; //Passive conn success
        conn_init_flag = 1;
        return;
      }
      if (state != STATE_WAIT_CONN)
        return; //Quit passive conn
    }

    reset_flags(); //Try connect timed out
  }
}
```

Transmission Mode

```
void try_recv() {
  uint8_t tmp_buff[60] = "+\0";
  uint8_t nrf_state = 0;
  NRF24L01_RX_Mode();

  for (int i = 0; i < 2000; ++i) {
    if (state != STATE_TRY_RECV)  break;
    nrf_state = NRF24L01_RxPacket(tmp_buff);
    HAL_Delay(5);

    if (nrf_state == 0) {
      state = STATE_TRY_SEND;
      if (tmp_buff[0] == '@') { // Chat msg
        recv_flag = 1;
        strcpy(recv_buff, tmp_buff + 1);
      }
      return;
    }
  }
  reset_flags(); // Connection timeout, disconnect
}

void try_send() {
  uint8_t tmp_buff[60] = " \0";
  uint8_t nrf_state = 0;
    NRF24L01_TX_Mode();
  if (send_flag == 1) { // Chat msg
    tmp_buff[0] = '@';
    strcpy(tmp_buff + 1, send_buff);
  }

  for (int i = 0; i < 2000; ++i) {
    if (state != STATE_TRY_SEND) break;
    nrf_state = NRF24L01_TxPacket(tmp_buff);
    HAL_Delay(5);
    if (nrf_state == TX_OK) {
      state = STATE_TRY_RECV;
      if (send_flag == 1)
        send_flag = 0;
      return;
    }
  }
  reset_flags(); // Connection timeout, disconnect
}
```

FSM

```
while (1) {
  ++counter; // Frequency division counter
  HAL_Delay(20);
  if (counter % 25 == 0 && NRF24L01_Check() == 1) {
    state = STATE_UNCHECK_MODE; // Module exception
    }

  LCD_Head();
  switch (state) {
  case STATE_UNCHECK_MODE:
    LCD_Unconn();
    for (int i = 0; i < 300; ++i) {
      if (NRF24L01_Check() == 0) { // Module pass check
        state = STATE_IDLE;
        break;
      }
    }
    break;

  case STATE_REQ_CONN:
    LCD_Unconn();
    HAL_GPIO_WritePin(LED0_GPIO_Port,
        LED0_Pin, GPIO_PIN_RESET);
    HAL_GPIO_WritePin(LED1_GPIO_Port,
        LED1_Pin, GPIO_PIN_SET);
    try_conn(); // Try to establish a connection
    break;

  case STATE_TRY_RECV:
    LCD_Conn();
    try_recv(); // Heartbeat mechanism
    break;

  case STATE_TRY_SEND:
    LCD_Conn();
    try_send(); // Heartbeat mechanism
    break;

  default:
    LCD_Unconn();
    break;
  }
}
```

Detail:

HALUARTRxCpltCallback(UART_HandleTypeDef *huart): Receive the chat message or command transmitted by serial port and , decode command.

try_conn(): Try an active (request) connection or a passive (accept) connection. It can cancel the connection by pressing the key or attempt connection timeout.

try_recv(): Heartbeat mechanism, trying to send a special message, which may contain chat message.

try_send(): Heartbeat mechanism, trying to receive a special message, which may contain chat message.

FSM : Controlling the state of the finite state machine.

4 Personal Division

- Shaofeng Zhang: Configure NRF24L01 wireless communication module to complete module detection and one-way data transmission has been received. LCD module UI design, data display; Module docking and assembly;
- Changzhen Zhang: Complete module connection and status detection, design the mode of LED lights in different states;
- Yechun Ruan: Design and implementation of finite state machine, heartbeat mechanism and serial port command;
- Ziyue Pang: Complete LCD screen display, make LCD adapt to this project requirement. Access LCD with module.

5 Problems and Solutions

- Dithering may occur when a key is pressed, that is, the key will be clicked twice.We added a delay function after each click to prevent the occurrence of two clicks.
- The logic between each module is more complex, we use the mode of finite state machine to connect.

6 Conclusion

This project mainly adopts the way of finite state machine to logically design and assemble each module. Through this project, we are familiar with and master the NRF24L01 wireless communication module. Besides, we deepened the understanding to the embedded programming.