Lesson 7 Serial Port Controls Servo

1. Project Purpose

Learn the serial communication and use the serial communication to control the serial bus servo to execute the corresponding command.

2. Project Principle

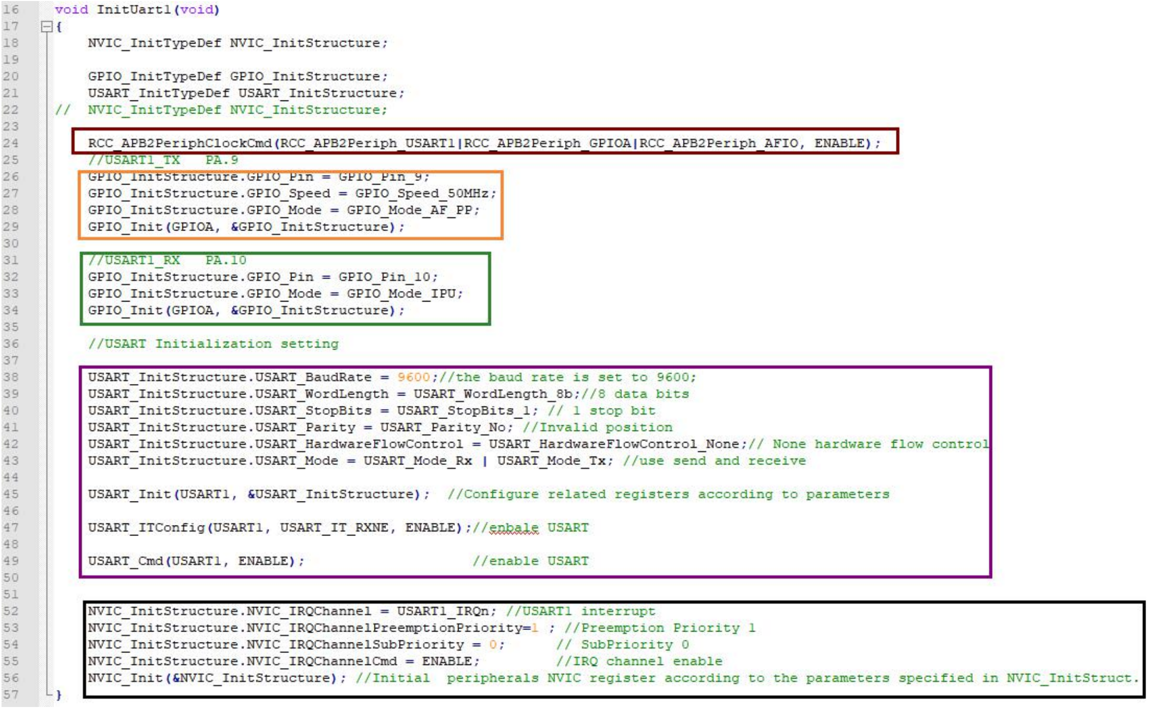
The serial communication interface is shorts for the serial port which refers to the communication interface transferring data one bit at a time. In the MCU and embedded environment, the serial port generally refers to the UART port.

According to the level standard of the interface, it can be divided into RS-232、RS-422、RS485、TTL, etc. TTL serial port is usually not converted by the specialized chip after we derive from the MCU chip.

Two types of physical ports are provided: DB9 connector and 4-pin header.

3. Program Analyst

1) We receive the serial data and make the servo perform corresponding operations based on the data. If want to use the serial communication, we need to configure the serial port first.

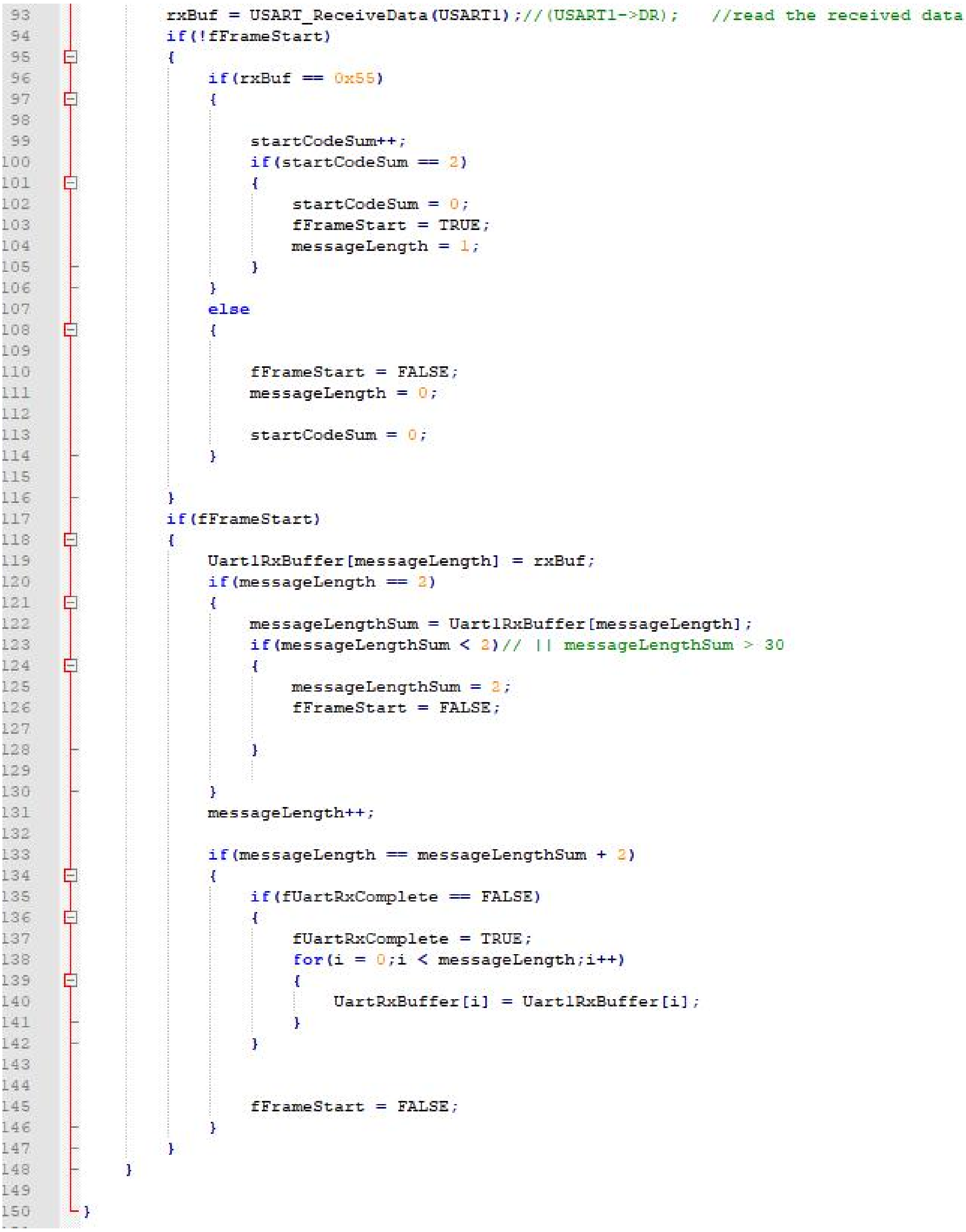


```
void InitUart1(void)
{
    NVIC_InitTypeDef NVIC_InitStructure;

    GPIO_InitTypeDef GPIO_InitStructure;
    USART_InitTypeDef USART_InitStructure;
//  NVIC_InitTypeDef NVIC_InitStructure;

    RCC_APB2PeriphClockCmd(RCC_APB2Periph_USART1|RCC_APB2Periph_GPIOA|RCC_APB2Periph_AFIO, ENABLE);
    //USART1_TX   PA.9
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_9;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;
    GPIO_Init(GPIOA, &GPIO_InitStructure);

    //USART1_RX   PA.10
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_10;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IPU;
    GPIO_Init(GPIOA, &GPIO_InitStructure);

    //USART Initialization setting

    USART_InitStructure.USART_BaudRate = 9600;//the baud rate is set to 9600;
    USART_InitStructure.USART_WordLength = USART_WordLength_8b;//8 data bits
    USART_InitStructure.USART_StopBits = USART_StopBits_1; // 1 stop bit
    USART_InitStructure.USART_Parity = USART_Parity_No; //Invalid position
    USART_InitStructure.USART_HardwareFlowControl = USART_HardwareFlowControl_None;// None hardware flow control
    USART_InitStructure.USART_Mode = USART_Mode_Rx | USART_Mode_Tx; //use send and receive

    USART_Init(USART1, &USART_InitStructure);  //Configure related registers according to parameters

    USART_ITConfig(USART1, USART_IT_RXNE, ENABLE);//enbale USART

    USART_Cmd(USART1, ENABLE);                    //enable USART

    NVIC_InitStructure.NVIC_IRQChannel = USART1_IRQn; //USART1 interrupt
    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority=1 ; //Preemption Priority 1
    NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;      // SubPriority 0
    NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;         //IRQ channel enable
    NVIC_Init(&NVIC_InitStructure); //Initial  peripherals NVIC register according to the parameters specified in NVIC_InitStruct.
}
```

2) The above figure is the code for initializing the serial port. The code in red box is for configuring the starting of the serial port and the clock of the used I/O port. The code in orange box is for configuring the sending pin of the serial port -PA9 to the multiplexed push-pull output;The code in green box is for configuring the receiving pin of the serial port PA10 to pull-up input.

3) The code in purple box is for configuring the parameters of the serial port, starting the serial port and related interrupt. Serial port parameters: baud rate 9600, 8 data bits, 1 stop bit, no parity bit. The code in black box is for configuring the serial port interrupt control parameters

4) After configuring the serial port, it can receive and send. We use the receive interrupt to receive, while the send uses the polling register method.

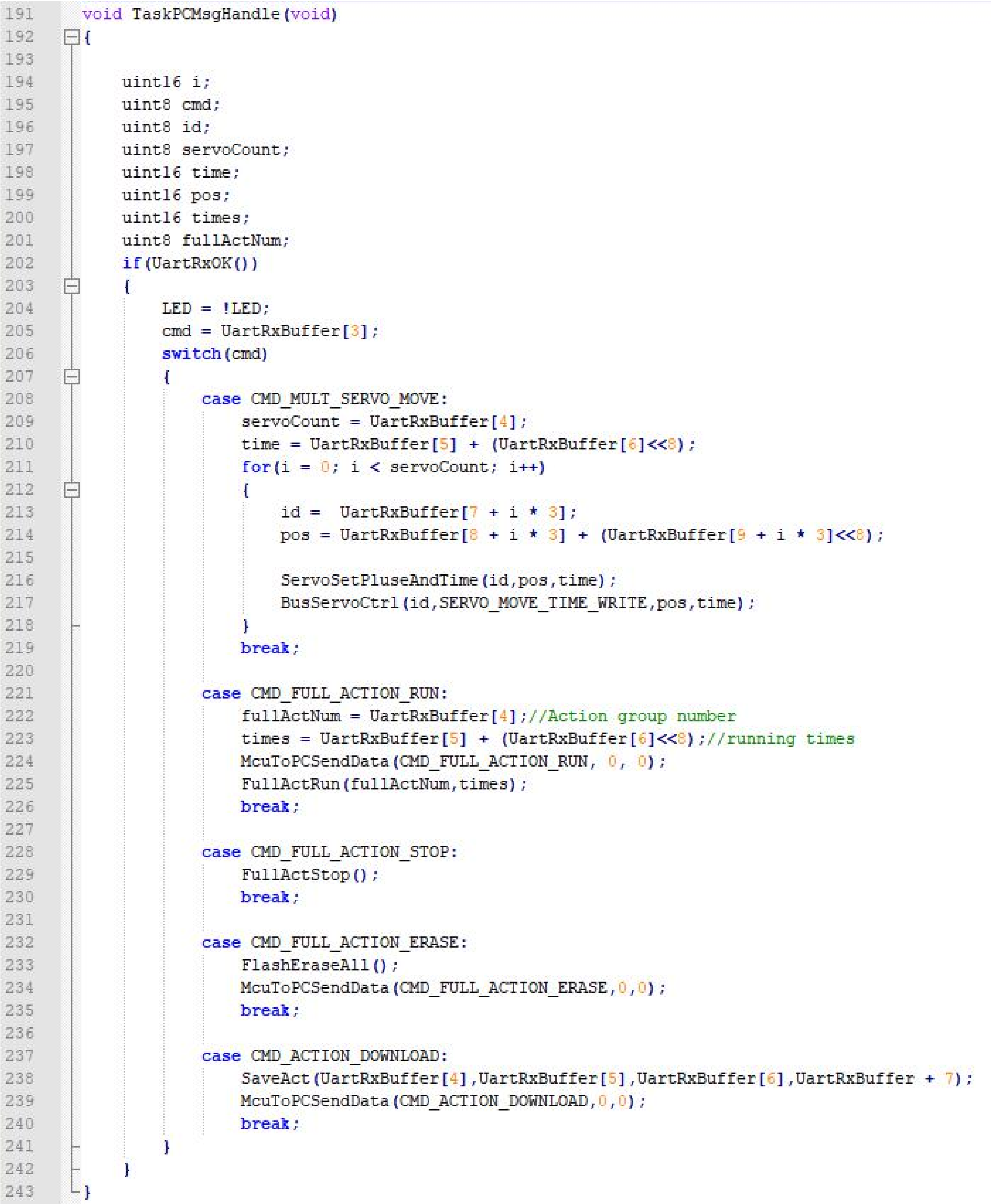


```
        rxBuf = USART_ReceiveData(USART1);//(USART1->DR);    //read the received data
        if(!fFrameStart)
        {
            if(rxBuf == 0x55)
            {

                startCodeSum++;
                if(startCodeSum == 2)
                {
                    startCodeSum = 0;
                    fFrameStart = TRUE;
                    messageLength = 1;
                }
            }
            else
            {

                fFrameStart = FALSE;
                messageLength = 0;

                startCodeSum = 0;
            }

        }
        if(fFrameStart)
        {
            Uart1RxBuffer[messageLength] = rxBuf;
            if(messageLength == 2)
            {
                messageLengthSum = Uart1RxBuffer[messageLength];
                if(messageLengthSum < 2)// || messageLengthSum > 30
                {
                    messageLengthSum = 2;
                    fFrameStart = FALSE;

                }

            }
            messageLength++;

            if(messageLength == messageLengthSum + 2)
            {
                if(fUartRxComplete == FALSE)
                {
                    fUartRxComplete = TRUE;
                    for(i = 0;i < messageLength;i++)
                    {
                        UartRxBuffer[i] = Uart1RxBuffer[i];
                    }
                }


                fFrameStart = FALSE;
            }
        }
    }

}
```

1) The code in the above figure is the serial receiving code. According to the communication protocol, the starting of the frame header is composed of two 0x55 so the function needs to use a static variable to record whether the frame header is received. If it is, then proceed to the next step, otherwise, discard the data and re-receive.
2) After the receiving is completed, the third byte is the data length except the frame header.If a byte is received in the program, the data length of the command of the current byte will be queried. If it meets the length indicated by the third byte, then it is considered that this command is received.

A complete data at least requires two frame headers 0x55 and a data length. A command, that is, its minimum data length is 2. Therefore, if the data length is less than 2, then there is a problem with the data transfer, and it will restart.
If there is no problem, the next frame will continue to be received until the total received frame length (the total frame length of the data we sent is 7) is equal to the data length (5) plus 2, that is, a complete data is received.

3) Finally, copy the received data to a buffer. Then call the following function to perform operations according to different commands..



```
void TaskPCMsgHandle(void)
{

    uint16 i;
    uint8 cmd;
    uint8 id;
    uint8 servoCount;
    uint16 time;
    uint16 pos;
    uint16 times;
    uint8 fullActNum;
    if(UartRxOK())
    {
        LED = !LED;
        cmd = UartRxBuffer[3];
        switch(cmd)
        {
            case CMD_MULT_SERVO_MOVE:
                servoCount = UartRxBuffer[4];
                time = UartRxBuffer[5] + (UartRxBuffer[6]<<8);
                for(i = 0; i < servoCount; i++)
                {
                    id =  UartRxBuffer[7 + i * 3];
                    pos = UartRxBuffer[8 + i * 3] + (UartRxBuffer[9 + i * 3]<<8);

                    ServoSetPluseAndTime(id,pos,time);
                    BusServoCtrl(id,SERVO_MOVE_TIME_WRITE,pos,time);
                }
                break;

            case CMD_FULL_ACTION_RUN:
                fullActNum = UartRxBuffer[4];//Action group number
                times = UartRxBuffer[5] + (UartRxBuffer[6]<<8);//running times
                McuToPCSendData(CMD_FULL_ACTION_RUN, 0, 0);
                FullActRun(fullActNum,times);
                break;

            case CMD_FULL_ACTION_STOP:
                FullActStop();
                break;

            case CMD_FULL_ACTION_ERASE:
                FlashEraseAll();
                McuToPCSendData(CMD_FULL_ACTION_ERASE,0,0);
                break;

            case CMD_ACTION_DOWNLOAD:
                SaveAct(UartRxBuffer[4],UartRxBuffer[5],UartRxBuffer[6],UartRxBuffer + 7);
                McuToPCSendData(CMD_ACTION_DOWNLOAD,0,0);
                break;
        }
    }
}
```

4) This function will be called in TaskRun in the main function. It calls the UartRxOk function to detect weather a command had been received. If a command is received, then it will execute commands such as run action group, ease action group and download action group.