SPC11X8/SPD11X8 UART 单元不定长数据接收

2022 年 3 月 – 版本 1

概述

在使用 UART 进行通信时，经常会遇到对不定长数据包的处理。发送数据时，只需根据指定长度发送数据包，处理相对容易；接收数据时就需要检查数据包的结束。本使用指南主要介绍通过 UART 的接收字符超时中断功能实现不定长数据的接收。

注：本文档主要以 SPC1168 为例进行介绍。

目录

表格列表

图片列表

版本历史

版本	日期	作者	变更
A/0	2019 年 7 月 9 日	Hengfang Huang	首次发布。

1 UART 特点

SPC11X8/SPD11X8 内建一个 Universal Asynchronous Receiver/Transmitter (UART)单元。SPC11X8/SPD11X8 的 UART 单元有以下特点：

- 最高支持 3.6Mbps
- 内建 64 Byte 接收缓存和 64 Byte 发送缓存
- 可编程的 FIFO 接收中断阈值
- 支持自动波特率检测
- 支持不归零编码（NRZ）

2 UART 接收字符超时中断功能

在 UART 通信中，针对速度较高、数据量较大的数据传输，接受/发送 FIFO 是一个非常有用的特性，在 SPC11X8/SPD11X8 的 UART 中，带有 64Byte 的发送/接受 FIFO。通信时还会经常遇到接收不定长数据包，可借助 UART 的接收字符超时中断功能实现该操作。本指南将通过使用 UART 接收中断和接收超时中断，完成对高速度大数据量的不定长数据的接收与回传。

当 RXFIFO 和接收超时中断被使能时，并且以下条件存在时，接收超时中断会发生：

- 最近一次收到的字符是在接收 4 个连续字符所需时间之前接收到的
- 最近一次 FIFO 读取是接收 4 个连续字符所需时间之前执行的

在从 RXFIFO 中读取一个字符之后或者接收到一个新的开始位，超时中断被清除并且超时计时器复位。如果超时中断没有发生，当接收到一个新的字符或者读取 RXFIFO 时，超时计时器复位。

接收超时中断发出信号通知尾部字节的存在，CPU 必须消除尾部字节。当处理接收超时中断服务时，CPU 使用下面的过程：

- 读取 UARTLSR 寄存器，检查错误。
- 通过 UARTIER.RTOIE 位段关闭接收器超时中断。
- 从 UART FIFO 中读取数据。
- 读取 UARTLSR 寄存器，检查错误，然后转到步骤 3。直到当 FIFO 中不在有数据，转到步骤 5。
- 通过 UARTIER.RTOIE 位段重新使能接收超时中断。
- 完成。

3 UART 接收不定长数据实例

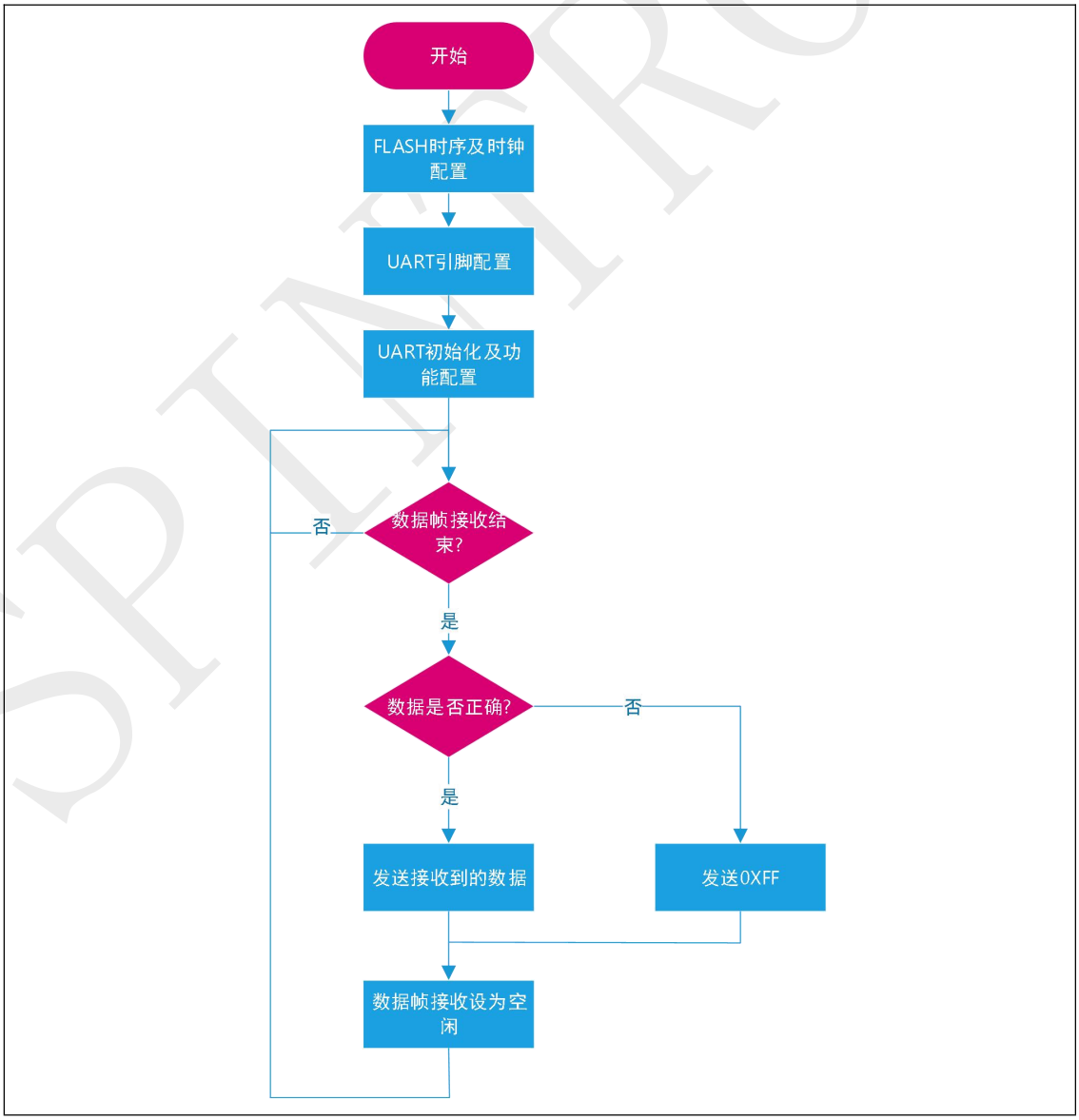
本实例中，使能了 UART 接收中断和接收超时中断，中断函数中处理 FIFO 中收到的数据，当 FIFO 错误时，丢弃掉错误之前的数据，直到 FIFO 错误状态清除。当进入接收超时中断后，则认为接收完一帧数据。

主程序中对帧数据的正确性进行异或校验判断，若正确，则返回源数据，否则返回 0XFF 错误标志。所以使用者测试时，需注意，发送的最后一个字节数据是前面所有字节数据的异或校验码。

3.1 主程序框图

如图 3-1 所示，为主程序处理流程框图。

图 3-1： 主程序流程图



3.2 实例程序

首先定义使用的宏及全局变量：

Example Code

```
/* Baudrate for UART */
#define UART_BAUDRATE 38400

/* Maximum size of buffer frame */
#define UART_RX_BUF_SIZE_MAX 256
/* Minimum size of buffer frame */
#define UART_RX_BUF_SIZE_MIN 2

/* UART Receive FIFO interrupt trigger threshold */
#define UART_RXFIFO_THRESHOLD UART_RXFIFO_32BYTE_OR_MORE

/* UART Transmit FIFO interrupt trigger threshold */
#define UART_TXFIFO_THRESHOLD UART_TXFIFO_EMPTY


typedef struct DeviceUartPack
{
  uint8_t* pu8RxBuf;
  uint16_t u16RxLength;

  /* A frame was received */
  uint8_t u8RxDataEnd;
} DeviceUartPack_t, *pDeviceUartPack_t;


DeviceUartPack_t MyDeviceUartPackTest;

uint8_t gu8UartRxBuff[UART_RX_BUF_SIZE_MAX];
```

定义使用的函数（其中 BCCXOR_Calculate()为异或检验函数）：

Example Code

```
void UartDataPack_Init(DeviceUartPack_t* myDataPack,
                       uint8_t* pu8Buffer)
{
  myDataPack->pu8RxBuf = pu8Buffer;
  myDataPack->u16RxLength = 0;

  myDataPack->u8RxDataEnd = 0;
}
```

```
uint8_t BCCXOR_Calculate(uint8_t* pu8Data, uint16_t u16DataLen)
{
  uint8_t result = 0;
  uint16_t index = 0;

  if(NULL == pu8Data)
  {
    return 0;
  }

  while(u16DataLen > 0)
  {
    result ^= pu8Data[index ++];
    u16DataLen--;
  }

  return result;
}

void UART_TX_Byte(uint8_t u8Data)
{
  /* Write the datum to UART Transmit Holding Register */
  UART_WriteByte(UART, u8Data);

  /* Wait for the TX FIFO empty */
  while(!UART_IsTxDone(UART));
}
```

在 mian()中，对时钟及带 FIFO 的 UART 进行初始化配置以及进行数据回传：

Example Code

```
int main()
{
  FLASH_WALLOW();
  FLASH_SetTiming(200000000);
  /* Disable flash write access after flash operation had done */
  FLASH_WDIS();

  CLOCK_InitWithRCO(CLOCK_HCLK_200MHZ);

  Delay_Init();

  /* Configure GPIO pin as UART function */
  GPIO_SetPinChannel(GPIO_34, GPIO34_UART_TXD);
```

```
GPIO_SetPinChannel(GPIO_35, GPIO35_UART_RXD);

/* UART Init */
UART_Init(UART, UART_BAUDRATE);

/* RX FIFO trigger level: 16 */
/* TX FIFO trigger level:  0 */
UART_SetFIFOTriggerThreshold(UART, UART_RXFIFO_THRESHOLD,
                                UART_TXFIFO_THRESHOLD);

/* Enable RX interrupt */
UART_EnableRxDataAvailableInt(UART);

/* Enable Receiver Time-out Interrupt */
UART_EnableRxTimeoutInt(UART);

/* Enable UART interrupt (main switch of UART interrupt) */
UART_EnableInt(UART);

/* Enable UART interrupt in CPU side */
NVIC_EnableIRQ(UART_IRQn);


/* Init UART data pack struct */
UartDataPack_Init(&MyDeviceUartPackTest, gu8UartRxBuff);


while(1)
{
  if(MyDeviceUartPackTest.u8RxDataEnd)
  {
    uint8_t* pu8Data = MyDeviceUartPackTest.pu8RxBuf;
    uint16_t u16Len = MyDeviceUartPackTest.u16RxLength;

    if((u16Len >= UART_RX_BUF_SIZE_MIN) && (pu8Data[u16Len - 1] ==
        BCCXOR_Calculate(pu8Data, u16Len - 1)))
    {
      /* If the data is correct, we return the value */
      UART_Write(UART, pu8Data, 0, u16Len);
    }
    else
    {
      /* An error occured, Send an error value */
      UART_TX_Byte(0XFF);
```

```
      }

      MyDeviceUartPackTest.u16RxLength = 0;
      MyDeviceUartPackTest.u8RxDataEnd = 0;
    }
  }
}
```

在中断函数中，进行数据接收处理：

Example Code

```
void UART_IRQHandler(void)
{
  uint8_t u8data;
  uint8_t u8error;

  /* RX data more than 32 */
  if(UART_GetRxDataAvailableIntStatus(UART))
  {
    u8error = UART->UARTLSR.all;

    /* Read data untill the receive FIFO is empty */
    while(UART_GetRxFIFOLevel(UART))
    {
      u8data = UART_ReadByte(UART);

      if((!MyDeviceUartPackTest.u8RxDataEnd) && (!(u8error &
         UARTLSR_ALL_FIFOERR_FIFO_ERROR_OCCUR)))
      {
        gu8UartRxBuff[MyDeviceUartPackTest.u16RxLength] = u8data;
        MyDeviceUartPackTest.u16RxLength =
        ((MyDeviceUartPackTest.u16RxLength + 1) % UART_RX_BUF_SIZE_MAX);
      }

      u8error = UART->UARTLSR.all;
    }
  }

  if(UART_GetRxTimeoutIntStatus(UART))
  {
    u8error = UART->UARTLSR.all;
    UART_DisableRxTimeoutInt(UART);

    /* Read data untill the receive FIFO is empty */
```

```
    while(UART_GetRxFIFOLevel(UART))
    {
      u8data = UART_ReadByte(UART);

      if((!MyDeviceUartPackTest.u8RxDataEnd) && (!(u8error &
          UARTLSR_ALL_FIFOERR_FIFO_ERROR_OCCUR)))
      {
        gu8UartRxBuff[MyDeviceUartPackTest.u16RxLength] = u8data;
        MyDeviceUartPackTest.u16RxLength =
        ((MyDeviceUartPackTest.u16RxLength + 1) % UART_RX_BUF_SIZE_MAX);
      }

      u8error = UART->UARTLSR.all;
    }

    MyDeviceUartPackTest.u8RxDataEnd = 1;
    UART_EnableRxTimeoutInt(UART);
  }
}
```

具体的测试方法：

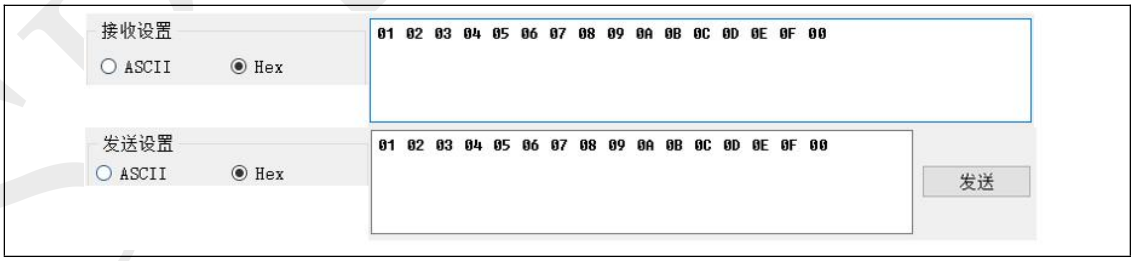
使用串口软件工具，设置参数（波特率 38400，无校验，1 位停止位），发送以下十六进制数据（最后一个字节 0x00 是前面 15 个数据的异或校验值）：

0x01 0x 02 0x 03 0x 04 0x05 0x 06 0x 07 0x08 0x 09 0x 0A 0x 0B 0x 0C 0x 0D

0x 0E 0x 0F 0x 00

发送完成后串口软件会接收到 MCU 返回的相同的数据。

图 3-2： 正确数据传输测试



如果串口软件发送一串错误数据，MCU 会返回 0XFF 值。

图 3-3： 错误数据传输测试