SPC11x8/SPD11x8 UART 模块使用指南

版本 A/0 – 2022 年 8 月

概述

通用异步收发器（UART）能够灵活地与外部设备进行全双工数据交换，常用于短距离、低速的串行通信中。UART 通过可编程的波特率产生器提供了多种波特率。SPC11x8/SPD11x8 内建一个 UART 单元。

注：本文档主要以 SPC1168 为例进行介绍。

目录

图片列表

表格列表

版本历史

版本	日期	作者	变更
1	2019 年 7 月 25 日	Hengfang Huang	首次发布。
2	2022 年 4 月 18 日	Hengfang Huang	1. 更新表 1-1 和表 1-2。 2. 增加章节 2。 1. 增加章节 3.1。 2. 更新章节 3.2。 3. 更新章节 3.3。 4. 更新章节 3.4。 5. 增加章节 3.5。
3	2022 年 7 月 7 日	Hengfang Huang	1. 更新表 1-1。
A/0	2022 年 8 月 26 日	Hengfang Huang	1. 更新文档格式。

术语或缩写

术语或缩写	描述
MCU	Microcontroller Unit，微控制器单元
LDO	Low Dropout Regulator，低压差稳压器
BLDC	Brushless Direct Current，无刷直流
FET	Field Effect Transistor，场效应管

1 UART 特性

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）

在 UART 的驱动库中，已经有下列驱动函数可供调动，可大大方便用户使用和理解。表 1-1 和表 1-2 中 UARTx 表示 UART 模块编号，取值为 UART。

表 1-1：UART 宏定义列表

宏名	功能及参数说明
UART_Enable(UARTx)	使能 UARTx
UART_Disable(UARTx)	禁用 UARTx
UART_EnableInt(UARTx)	使能 UARTx 中断
UART_DisableInt(UARTx)	禁用 UARTx 中断
UART_EnableRxDataAvailableInt(UARTx)	使能 UARTx 的接收器数据有效中断 普通模式：UARTx 中是否有接收数据 FIFO 模式：UARTx RXFIFO 中的数据是否超过阈值
UART_DisableRxDataAvailableInt(UARTx)	禁止 UARTx 的接收数据中断
UART_EnableTxDataRequestInt(UARTx)	使能 UARTx 的 TXFIFO 发送数据请求中断
UART_DisableTxDataRequestInt(UARTx)	禁止 UARTx 的 TXFIFO 发送数据请求中断
UART_EnableRxErrorInt(UARTx)	使能 UARTx 的接收错误中断
UART_DisableRxErrorInt(UARTx)	禁止 UARTx 的接收错误中断
UART_EnableRxTimeoutInt(UARTx)	使能 UARTx 的接收超时中断
UART_DisableRxTimeoutInt(UARTx)	禁止 UARTx 的接收超时中断
UART_GetGlobalIntStatus(UARTx)	获取 UARTx 的中断挂起状态
UART_GetTxDataRequestIntStatus(UARTx)	获取 UARTx 的 FIFO 发送数据请求中断状态
UART_GetRxDataAvailableIntStatus(UARTx)	获取 UARTx 的接收到的数据有效中断状态 普通模式：UARTx 中是否有接收数据 FIFO 模式：UARTx RXFIFO 中的数据是否超过阈值
UART_GetRxErrorIntStatus(UARTx)	获取 UARTx 的接收错误中断状态
UART_GetRxTimeoutIntStatus(UARTx)	获取 UARTx 的接收超时中断状态
UART_DisableFIFO(UARTx)	禁止 UARTx FIFO 功能
UART_ClearRxFIFO(UARTx, eRxTrigLvl,	清空 UART RXFIFO UARTx：UART 模组基址 eRxTrigLvl：RxFIFO 阈值

宏名	功能及参数说明
eTxTrigLvl)	eTxTrigLvl：TxFIFO 阈值
UART_ClearTxFIFO(UARTx, eRxTrigLvl, eTxTrigLvl)	清空 UART TXFIFO UARTx：UART 模组基址 eRxTrigLvl：RxFIFO 阈值 eTxTrigLvl：TxFIFO 阈值
UART_GetRxFIFOLevel(UARTx)	获取 UARTx RXFIFO 中的数据量
UART_EnableSetBreak(UARTx)	在 UARTx 的发送管脚强制发送低电平
UART_DisableSetBreak(UARTx)	解除 UARTx 发送管脚强制低电平
UART_EnableStickParity(UARTx)	强制校验位的值和 UARTLCR.EPS 位的值相反
UART_DisableStickParity(UARTx)	解除校验位的值和 UARTLCR.EPS 位的值相反，校验位的值根据数据变化
UART_IsFIFOError(UARTx)	UARTx 是否出现 FIFO 错误
UART_IsTxDone(UARTx)	UARTx 是否发送器中所有数据都被移出
UART_IsTxDataRequest(UARTx)	UARTx 是否出现发送数据请求
UART_IsRxBreak(UARTx)	UARTx 是否收到 Break(接收间断)信号
UART_IsRxFrameError(UARTx)	UARTx 是否出现接收帧错误
UART_IsRxParityError(UARTx)	UARTx 是否发现接收校验错误
UART_IsRxOverflow(UARTx)	UARTx 是否出现接收溢出错误
UART_IsRxNotEmpty (UARTx)	是否 UARTRBR 或者 FIFO 中的数据是有效的 当一个完整的字符被接收到并被传输到 RBR 或者 FIFO 时即为有效
UART_WriteByte(UARTx, u8Data)	UARTx 发送一个字节数据 UARTx：UART 模组基址 u8Data：要发送的数据
UART_ReadByte(UARTx)	通过 UARTx 读取一个字节数据

表 1-2：UART 函数定义列表

函数名	功能及参数说明
void UART_Init(UART_REGS* UARTx, uint32_t u32Baudrate)	根据波特率 u32Baudrate 初始化 UART UARTx：UART 模组基址 u32Baudrate: 通讯速率
void UART_SetFIFOTriggerThreshold (UART_REGS* UARTx, UART_RxTriggerLevelEnum eRxTrigLvl, UART_TxTriggerLevelEnum eTxTrigLvl)	配置 UARTx 的 FIFO 触发阈值 UARTx：UART 模组基址 eRxTrigLvl：设置 RxFIFO 阈值 eTxTrigLvl：设置 TxFIFO 阈值
void UART_WriteText(UART_REGS *UARTx, const uint8_t *pu8Text)	发送 pu8Text 指向的字符串数据
void UART_Write(UART_REGS* UARTx, uint8_t *pu8Buffer, uint32_t u32Offset, uint32_t u32Count)	发送 pu8Buffer 中偏移 u32Offset 的 u32Count 个字节的数据
void UART_Read(	接收 u32Count 个数据，存储至从 u32Offset 开

函数名	功能及参数说明
UART_REGS* UARTx, uint8_t *pu8Buffer, uint32_t u32Offset, uint32_t u32Count)	始的 pu8Buffer 中

2 UART 使用注意事项

使用 UART 时，以下几点是需要特别注意的，本节以 SPC1168 为例进行说明。

- UART 有三组寄存器是共地址的，如图 2-1 所示。对于①和②组寄存器，通过使用 UARTLCR.DLAB 寄存器位段选择需要访问的寄存器，如图 2-2 所示。对于③组寄存器，UARTIIR 是只读寄存器，UARTFCR 是只写寄存器，关于 UARTFCR 寄存器的正确使用方法，请参阅章节 3.3。

图 2-1：UART 寄存器列表

	寄存器	偏移地址	描述	复位值
	UARTRBR	0x00	UART 接收缓存寄存器	0x00000000
①	UARTTHR	0x00	UART 发送保持寄存器	0x00000000
	UARTDLL	0x00	UART 分频系数低字节寄存器	0x00000002
②	UARTDLH	0x04	UART 分频系数高字节寄存器	0x00000000
	UARTIER	0x04	UART 中断使能寄存器	0x00000000
③	UARTIIR	0x08	UART 中断识别寄存器	0x00000001
	UARTFCR	0x08	UART FIFO 控制寄存器	0x00000000
	UARTLCR	0x0C	UART 线路控制寄存器	0x00000000
	UARTMCR	0x10	UART 调制解调器控制寄存器	0x00000000
	UARTLSR	0x14	UART 线路状态寄存器	0x00000060
	UARTISR	0x20	UART 红外选择寄存器	0x00000000
	UARTFOR	0x24	UART 接收 FIFO 占用寄存器	0x00000000
	UARTABR	0x28	UART 自动波特率控制寄存器	0x00000000
	UARTACR	0x2C	UART 自动波特率计数寄存器	0x00000000

图 2-2：UARTLCR.DLAB 寄存器位段描述

表 17-20： UART 线路控制寄存器（UARTLCR）位段描述

位段	位段名	属性	复位值	描述
31:8	RESERVED_31_8	RO	0x0	保留
7	DLAB	RW	0x0	分频寄存器访问锁定位 在读或者写操作期间，必须置“1”，以便可以访问波特率产生器的分频系数锁存寄存器（Divisor Latch registers）。当要访问 RBR 寄存器，THR 寄存器或者 UARTIER 寄存器时，必须清零。 0：访问 UARTRBR，UARTTHR 和 UARTIER 1：访问分频系数锁存寄存器（UARTDLL 和 UARTDLH）

- 当 UART 单元被关闭时，发送器或者接收器完成当前的字节传输后，会停止发送或者接收更多的数据。在 FIFO 中的数据不会被清除，当 UART 使能时，这些数据的传输会重新开始。
- 当 FIFO 的 64 字节空间已满时，新传送的字节数据将会被丢弃。

3 UART 实例

3.1 UART 采用 polling 方式收发实例

本示例中，有两个配套代码，分别为 UART 数据接收端代码和 UART 数据发送端代码。

UART 数据发送端代码，采用 polling 方式发送数据给接收端，先逐个发送字（4 字节）数据，总共发送 10 个，每发送一个都会等待接收端返回的数据，并进行对比。然后再逐个发送字节数据，也发送 10 个，并与接收到的数据进行对比。

两个实例代码中有部分程序是完全相同的。比如公用的宏和全局变量，以及 UART 的接收和发送函数：

Example Code

```
#define DatumCount 10

int i;
uint32_t u32RXData, u32TXData;
uint8_t u8RXData, u8TXData;
```

Example Code

```
uint32_t UART_RX_Word(void)
{
  volatile uint32_t i = 0;
  uint32_t u32Data = 0;

  for(i = 0; i < 4; i++)
  {
    /* Wait until data is available in RBR or the FIFO */
    while(!UART_IsRxNotEmpty(UART));

    /* In receive process, the MSB come first and the LSB comes the last
*/
    u32Data |= UART_ReadByte(UART) << ( ( 3 - i ) * 8 );
  }

  return u32Data;
}

uint32_t UART_RX_Byte(void)
{
  uint8_t u8Data = 0;

  /* Wait until data is available in RBR or the FIFO */
  while(!UART_IsRxNotEmpty(UART));

  /* In receive process, the MSB come first and the LSB comes the last
*/
  u8Data = UART_ReadByte(UART);
```

```
  return u8Data;
}

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));
}

void UART_TX_Word(uint32_t u32Data)
{
  volatile uint32_t i = 0;

  for(i = 0; i < 4; i++)
  {
    /*
     * Write the datum to UART Transmit Holding Register, which will
be sent out, the MSB sent out first.
     */
    UART_WriteByte(UART, (u32Data & ( 0xFF000000 >> ( i * 8 ) ) ) >>
( ( 3 - i ) * 8 ) );

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

UART 的数据接收端代码，采用 polling 方式接收发送端的数据，并将接收的数据发送回发送端。在 main()中，进行时钟及 UART 的基本配置和操作：

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();

  /*
   * Init the UART
   *
   * 1.Set the GPIO34/35 as UART FUNC
   *
   * 2.Enable the UART CLK
   *
```

```
   * 3.Set the rest para
   */
  GPIO_SetPinChannel(GPIO_34,GPIO34_UART_TXD);
  GPIO_SetPinChannel(GPIO_35,GPIO35_UART_RXD);
  CLOCK_EnableModule(UART_MODULE);
  UART_Init(UART,38400);

  /* Receive the world data comes from the transmitting end, and sent
it back */
  for(i = 0; i < DatumCount; i++)
  {
    u32RXData = UART_RX_Word();
    UART_TX_Word(u32RXData);
  }

  /* Receive the byte data comes from the transmitting end, and sent
it back */
  for(i = 0; i < DatumCount; i++)
  {
    u8RXData = UART_RX_Byte();
    UART_TX_Byte(u8RXData);
  }

  while(1)
  {

  }
}
```

UART 数据发送端代码，采用 polling 方式发送数据给接收端，并将接收端返回的数据进行数据对比。在 main()中，进行时钟及 UART 的基本配置和操作：

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();

  /*
   * Init the UART
   *
   * 1.Set the GPIO34/35 as UART FUNC
   *
   * 2.Enable the UART CLK
   *
   * 3.Set the rest para
```

```
   */
  GPIO_SetPinChannel(GPIO_34,GPIO34_UART_TXD);
  GPIO_SetPinChannel(GPIO_35,GPIO35_UART_RXD);
  CLOCK_EnableModule(UART_MODULE);
  UART_Init(UART,38400);

  /* Generate the world datum to be sent and sent them out */
  for(i = 0; i < DatumCount; i++)
  {
    u32TXData = rand() & 0xFFFFFFFF;
    UART_TX_Word(u32TXData);
    u32RXData = UART_RX_Word();
    if(u32TXData != u32RXData)
    {
      printf("UART TX/RX Worlds test FAIL\n");
      return 0;
    }
  }
  printf("UART TX/RX Worlds test PASS\n");

  /* Generate the byte datum to be sent and sent them out */
  for(i = 0; i < DatumCount; i++)
  {
    u8TXData = rand() & 0xFF;
    UART_TX_Byte(u8TXData);
    u8RXData = UART_RX_Byte();
    if(u8TXData != u8RXData)
    {
      printf("UART TX/RX BYTE test FAIL\n");
      return 0;
    }
  }
  printf("UART TX/RX BYTE test PASS\n");

  while(1)
  {

  }
}
```

3.2 UART 采用接收中断收发实例

本示例中，会展示 UART 最基本的收发功能，以及最基本的接收中断的使用方法。

首先定义一个要用到的全局变量：

Example Code

```
uint8_t u8data;
```

在 main()中，进行时钟及 UART 的基本配置：

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();

  /*
   * Init the UART
   *
   * 1.Set the GPIO34/35 as UART FUNC
   *
   * 2.Enable the UART CLK
   *
   * 3.Set the rest para
   */
  GPIO_SetPinChannel(GPIO_34,GPIO34_UART_TXD);
  GPIO_SetPinChannel(GPIO_35,GPIO35_UART_RXD);
  CLOCK_EnableModule(UART_MODULE);
  UART_Init(UART,38400);

  /* Set the receive fifo INT trigger level as 1 bytes */

  UART_SetFIFOTriggerThreshold(UART,UART_RXFIFO_1BYTE_OR_MORE,
                               UART_TXFIFO_EMPTY);

  /* Enable data reaching INT */
  UART_EnableRxDataAvailableInt(UART);

  /* Enable UART INT to CPU */
  UART_EnableInt(UART);

  /* Enable the Gloable INT of UART */
  NVIC_EnableIRQ(UART_IRQn);

  while(1)
```

```
    {
    }
}
```

在中断函数中，将接收的数据转换后打印到控制台：

Example Code

```
void UART_IRQHandler(void)
{
  /* Read data until the receive FIFO is empty */
  while(UART_GetRxFIFOLevel(UART))
  {
    u8data = UART_ReadByte(UART) - '0';
    printf("INT rece data(0~10): %d\n",u8data);
  }
}
```

3.3 UART FIFO 使用实例

在速度较高、数据量较大的 UART 通信中，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。经过一些系统实测，这个特性可以让 SPC11x8/SPD11x8 的 UART 工作在 3.6Mbps 持续发送接收的状态。

同样，在 SCP1168 提供的 SDK 中，已经包含了很多针对 UART FIFO 功能优化过的 API 函数。用户如果需要使用 UART 的 FIFO 功能，只需要简单的调用这些 API 即可，无需详细地理解 UART FIFO 的实现细节。

这里要特别注意的是 FIFO 寄存器的设置，由于这个寄存器是一个 Write-Only 寄存器，即对它进行读取操作不会有返回值，所以一般使用的位与、位或操作是无法作用到这个寄存器上的。这里限定使用 SDK 中的函数进行设置：

```
void UART_SetFIFOTriggerThreshold(UART_REGS* UARTx,
                                  UART_RxTriggerLevelEnum eRxTrigLvl,
                                  UART_TxTriggerLevelEnum eTxTrigLvl)
```

同理，当需要清空 TXFIFO 或 RXFIFO 时，限定使用 SDK 中的宏函数进行操作：

```
UART_ClearRxFIFO(UARTx, eRxTrigLvl, eTxTrigLvl)
UART_ClearTxFIFO(UARTx, eRxTrigLvl, eTxTrigLvl)
```

用户需要设定 eRxTrigLvl（接收 FIFO 触发阈值）和 eTxTrigLvl（发送 FIFO 触发阈值），同时，这三个函数会默认使能 FIFO 功能（必须使能 FIFO 功能，才能正确写本寄存器的其他位段，否则的话，其他位段不会被写入）。

该实例需要短接 TXD 和 RXD 引脚。

3.3.1 例 1：UART 接收 FIFO 使用例程

在 mian()中，对时钟及带 FIFO 的 UART 进行初始化配置：

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, 38400);

  /* RX FIFO trigger level: 16 */
  /* TX FIFO trigger level:  0 */
  UART_SetFIFOTriggerThreshold(UART, UART_RXFIFO_16BYTE_OR_MORE,
                               UART_TXFIFO_EMPTY);

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

  /* Enable UART interrupt (main switch of UART interrupt) */
  UART_EnableInt(UART);
  /* Enable UART ISR in CPU side */
  NVIC_EnableIRQ(UART_IRQn);

  while(1)
  {
    Delay_Ms(50);
    UART_WriteByte(UART, 0x55);
  }
}
```

在中断函数中，进行数据接收处理：

Example Code

```
void UART_IRQHandler(void)
{
  /* RX data more than 16 */
  if(UART_GetRxDataAvailableIntStatus(UART))
  {
    /* Check RX FIFO Level */
    if(UART_GetRxFIFOLevel(UART) >= 16)
    {
```

```
      /* Read data from FIFO  */
      UART_Read(UART, UARTRead, 0, 16);
    }
  }
}
```

3.3.2 例 2：UART 发送 FIFO 使用例程

在下面的例子中，展示了如何使用 UART 的发送 FIFO，这个例子是基于之前的接收中断的例子上增加了发送中断的部分。在发送 FIFO 的数据被发送完之后，可以向 CPU 发送中断，进入中断后重新进行数据的装载。

该实例需要短接 TXD 和 RXD 引脚。

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, 38400);

  /* RX FIFO trigger level: 16 */
  /* TX FIFO trigger level:  0 */
  UART_SetFIFOTriggerThreshold(UART, UART_RXFIFO_16BYTE_OR_MORE,
                               UART_TXFIFO_EMPTY);

  /* Enable TX FIFO interrupt */
  UART_EnableTxDataRequestInt(UART);
  /* Enable RX interrupt */
  UART_EnableRxDataAvailableInt(UART);
  /* Enable UART interrupt (main switch of UART interrupt) */
  UART_EnableInt(UART);

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

  while(1){}
}

void UART_IRQHandler(void)
{
  uint32_t i;
```

```
  switch(UART->UARTIIR_UARTFCR.UARTIIR.bit.IID)
  {
    case UARTIIR_BIT_IID_TX_FIFO_REQUEST_DATA_INT:
      /* Fill Tx FIFO */
      for(i = 0; i <= 5; i++)
      {
        UART_WriteByte(UART, 0x55);
      }
      Delay_Ms(20);
      break;
    case UARTIIR_BIT_IID_RX_DATA_READY_INT:
      /* Check Rx FIFO level */
      if(UART_GetRxFIFOLevel(UART) >= 16)
      {
        /* Read data from RX FIFO */
        UART_Read(UART, UARTRead, 0, 16);
        Delay_Ms(100);
      }
      break;
    default:
      break;
  }
}
```

3.4 UART 自动波特率检测实例

SPC11x8/SPD11x8 的 UART 具有自动波特率计算的能力，它内建一个计数器，会计算在 RX 上接收到的从第一个下降沿到第一个上升沿之间的时间间隔。根据 UART 的协议，如果接收到的第一位是“1”的话，这个间隔应该是传输 1 个 UART 数据位的时间。根据这个假设，如果对端发送 0bxxxxxx1 的数据，则可以计算出正确的波特率。

下面的例子实现了 UART 的自动波特率计算，不过在具体流程上作了一定简化，即对端第一个数据必须发送 0x1。该实例需要使用 USB 转串口模块连接芯片的 TXD 和 RXD 引脚。

UART 的初始化代码如下：

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);
  CLOCK_EnableModule(UART_MODULE);

AUTOBAUD:
```

```
  /* Enable Autobaud detect */
  UART->UARTABR.bit.ABE   = UARTABR_BIT_ABE_ENABLE;
  /* UART unit programs Divisor Latch registers */
  UART->UARTABR.bit.ABUP  = UARTABR_BIT_ABUP_UART_PROG_DIVISOR;
  /* Enable Autobuad lock interrupt */
  UART->UARTABR.bit.ABLIE = UARTABR_BIT_ABLIE_ENABLE;

  /* 8 databits */
  UART->UARTLCR.bit.WLS   = UARTLCR_BIT_WLS_8_DATA_BIT;
  /* 1 stop bit */
  UART->UARTLCR.bit.STB   = UARTLCR_BIT_STB_1_STOP_BIT;
  /* No parity  */
  UART->UARTLCR.bit.EPS   = UARTLCR_BIT_PEN_NO_PARITY;

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

  /* Wait auto-baud lock */
  while(autoBaud == 0);

  /* Disable auto-baud detect */
  UART->UARTABR.bit.ABE = 0;
  /* Disable auto-baud lock interrupt */
  UART->UARTABR.bit.ABLIE = 0;

  /* Wait data received */
  while(!UART_IsRxNotEmpty(UART));

  /* Read data and validate it */
  while(UART_ReadByte(UART) != 0x01)
  {
    UART_Disable(UART);
    goto AUTOBAUD;
  }

  printf("Auto-baud success!\n");

  /* Enable receive data available interrupt */
  UART_EnableRxDataAvailableInt(UART);


  while(1)
  {
    Delay_Ms(1000);
  }
}
```

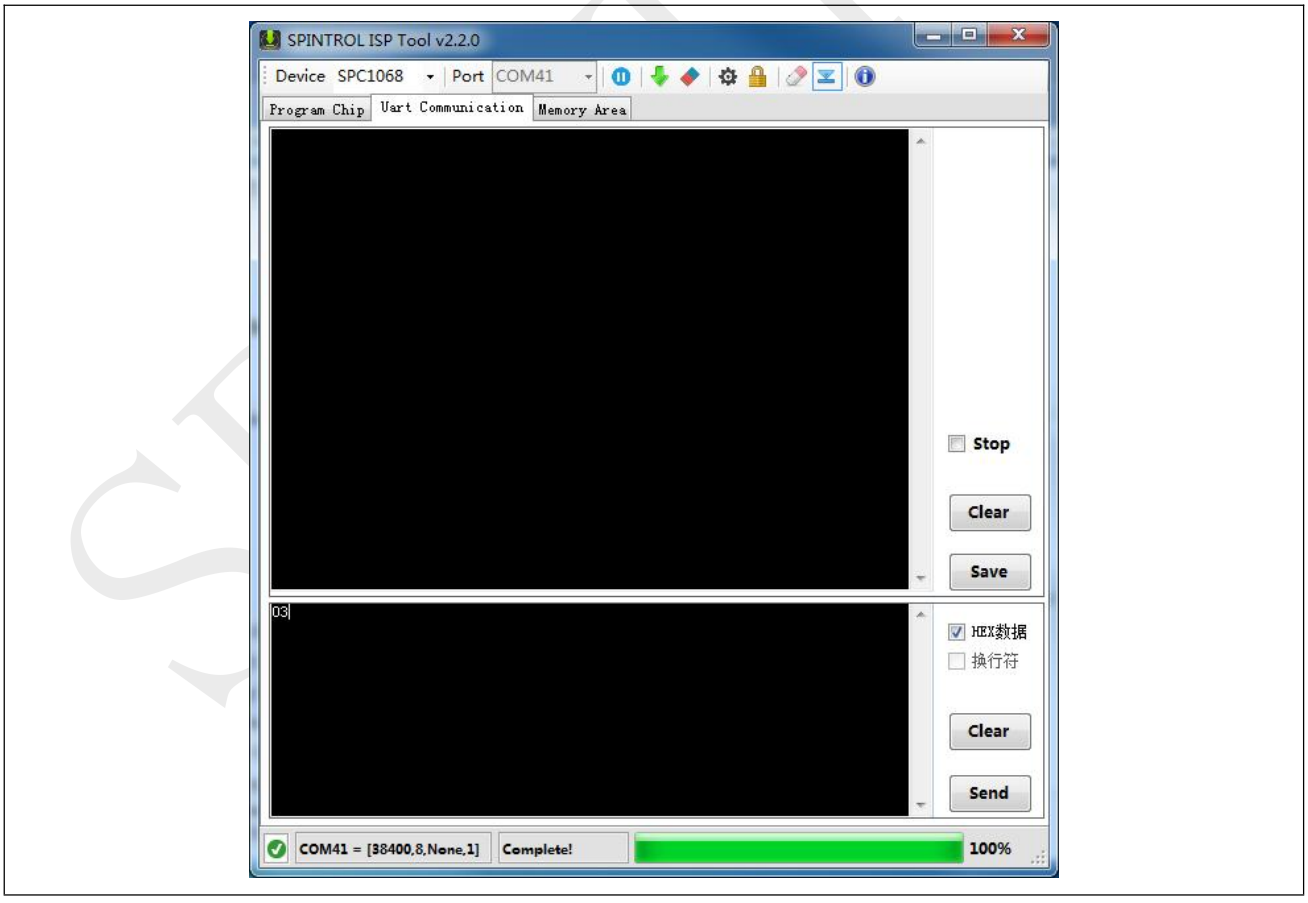
```
void UART_IRQHandler(void)
{
  /* Auto-baud locked */
  if(UART->UARTIIR_UARTFCR.UARTIIR.bit.ABL == 1)
  {
    autoBaud = 1;
  }

  /* Receive data available */
  if(UART_GetRxDataAvailableIntStatus(UART))
  {
    /* Write data back */
    UART_WriteByte(UART, UART_ReadByte(UART));
  }
}
```

具体的观察方法：启动 ISP 工具后，在任意波特率下，向 SPC11x8/SPD11x8 的 UART 发送一个字节数据：

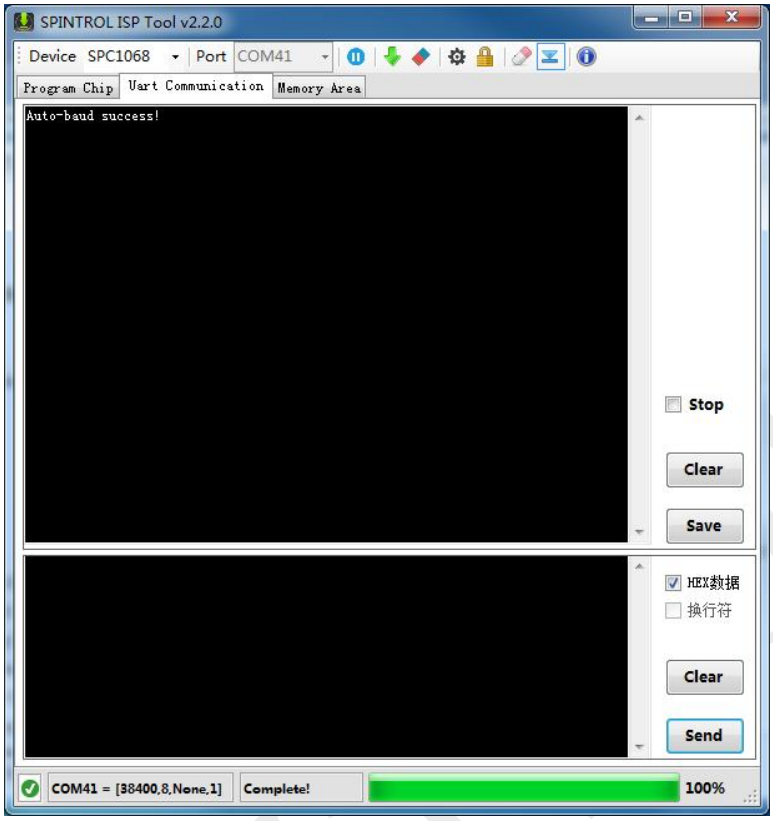
1. 如果数据不等于 0x1，则没有任何反应

图 3-1：自动波特率检测失败



2. 如果数据等于 0x1，则可以收到 SPC11x8/SPD11x8 返回的正确数据，可以判定 Autobaud 工作正常。

图 3-2：自动波特率检测成功



3.5 UART 不定长数据接收实例

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，可借助 UART 的接收字符超时中断功能实现该操作。本节将通过使用 UART 接收中断和接收超时中断，配合 RXFIFO 特性，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：

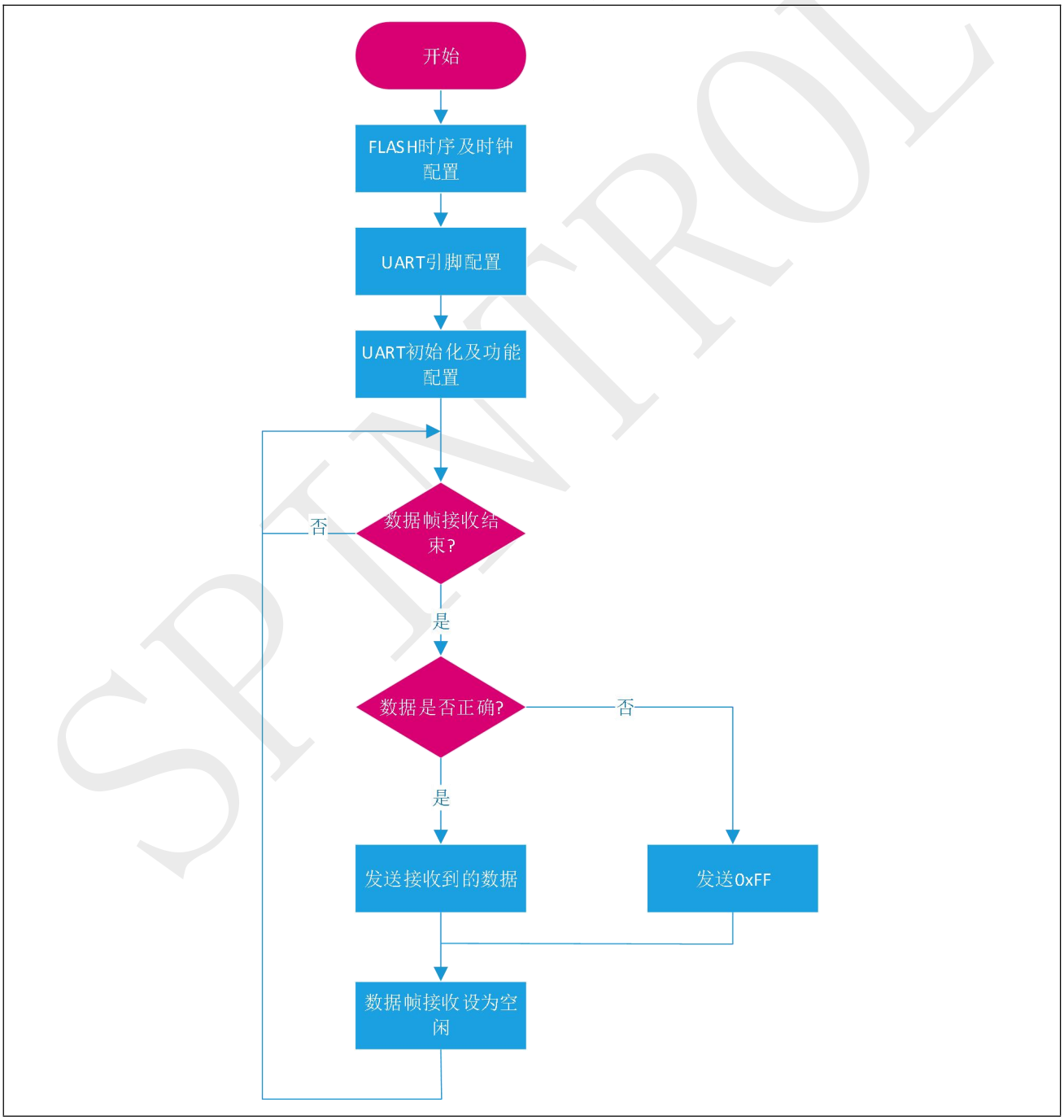
- 步骤 1：读取 UARTLSR 寄存器，检查错误；
- 步骤 2：通过 UARTIER.RTOIE 位段关闭接收器超时中断；
- 步骤 3：从 UART RXFIFO 中读取数据；
- 步骤 4：读取 UARTLSR 寄存器，检查错误，然后转到步骤 3，直到当 RXFIFO 中不再有数据，转到步骤 5。

- 步骤 5：通过 UARTIER.RTOIE 位段重新使能接收超时中断。
- 完成。

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。如图 3-3 所示，为主程序处理流程框图。

图 3-3：主程序流程图



以下为实例代码。

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 until the receive FIFO is empty */
    while(UART_GetRxFIFOLevel(UART) > 1)
    {
      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);
  }
}
```

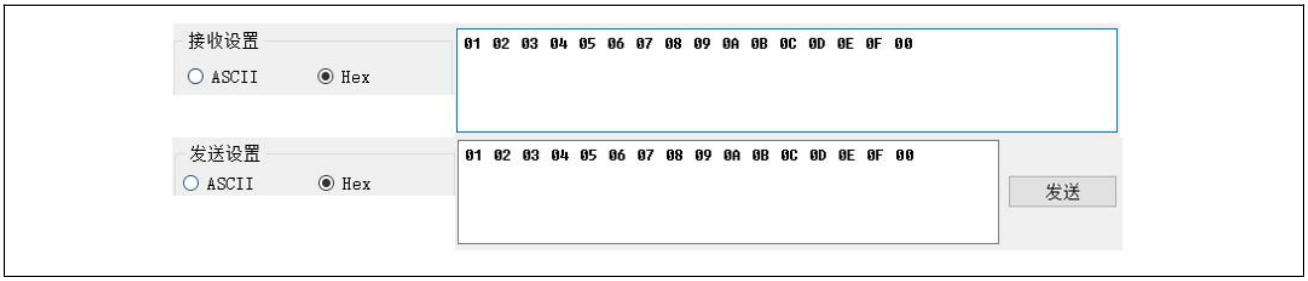
具体的测试方法：

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 3-4：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 3-5：错误数据传输测试