BAT32G137（库函数版本）

Rev 1.0

修订历史

版本	日期	修订人	修订内容
Rev1.1	22.6.22	缪勤文 张刚	

目录

1. 前言

串口功能可在 BAT32G137 系列手册的“通用串行单元(SCI)”查看。SCI 使用了三合一功能，SCI 单元下的通道同一时刻只能复用为 IIC、SPI、UART 中的一种功能；对于 SCI 模块，每一个串口的 TX、RX 分别占用一个通道。

2. 通用串行通信单元通道分配

以 BAT32G137 64 PIN 为例:

单元	通道	用作SSPI	用作UART	用作简易I²C
0	0	SSPI00 （支持从属选择输入功能）	UART0 TX	IIC00
	1	SSPI01	RX	IIC01
	2	SSPI10	UART1	IIC10
	3	SSPI11		IIC11
1	0	SSPI20	UART2	IIC20
	1	SSPI21		IIC21

对于单元 0 ，SCI 模块分配了 4 个通道，其中串口 TX、RX 分别占用通道 0、通道 1;当芯片使用 UART0 功能，则 SSPI00/IIC00、SSPI01/IIC01 则无法使用;若只使用 UART0 TX，则 SSPI01/IIC01 仍然可以使用;别的以此类推。

BAT32G137 64PIN 芯片引脚图如下图 1，BAT32G137 芯片数字功能不是全映射，下图红框为引脚固定功能，即 P50 P51 为串口 0 的引脚；P02 P03 为串口 1 的引脚；P13 P14 为串口 2 的引脚；黄色框图为引脚可复用为串口功能，P11 P12 P16 P17 P40 P137 可复用为串口 0；P76 P77 可复用为串口 2。

注意：BAT32G137 64PIN 串口 1 只能使用 P02 P03；根据芯片 datasheet BAT32G21137 其余封装 48PIN/40PIN/32PIN 串口 1 默认引脚为 P00 P01；P73 P72 可复用为串口 1。

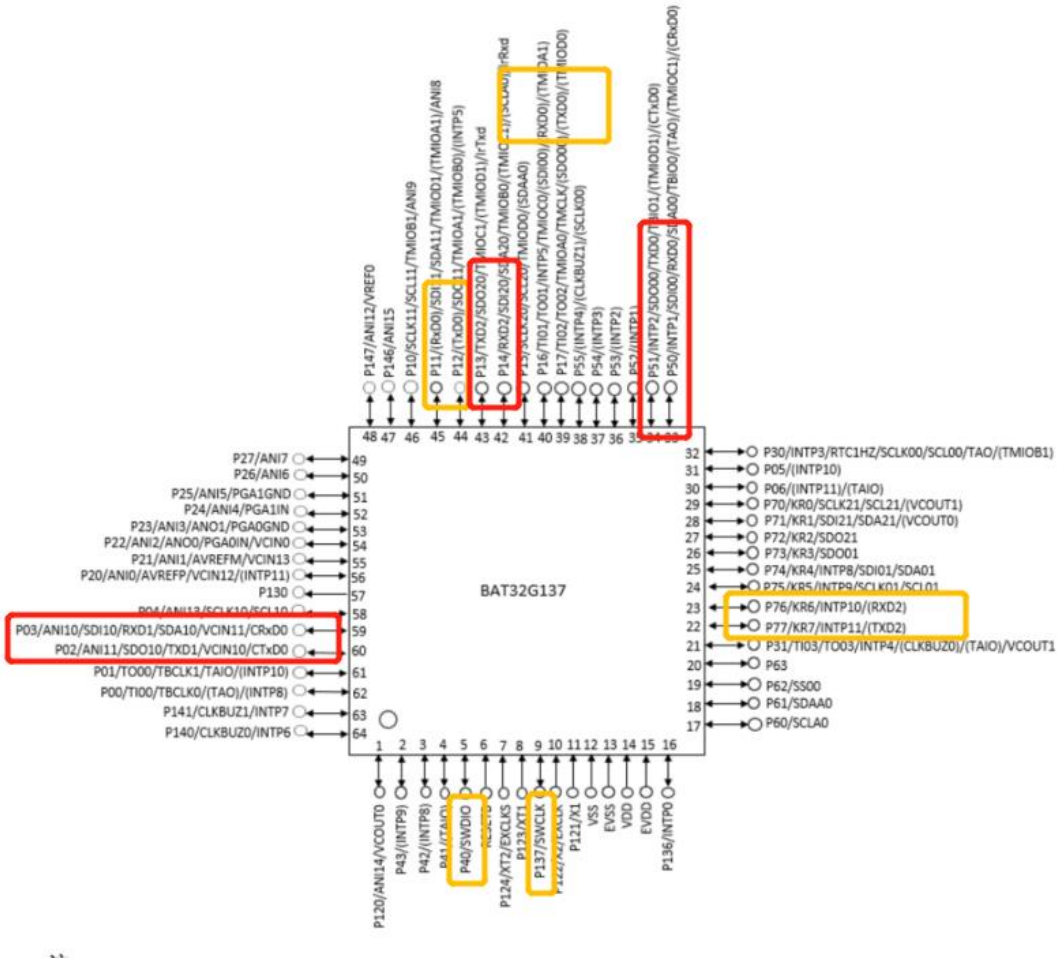

图 1 BAT32G137 64PIN 串口引脚分配

3. 中微 BAT32G137 系列 UART 应用库简介

中微 BAT32G137 系类软件 UART 应用库是一个便于移植的标准库代码风格，用户只需要对软件接口相关参数进行简单配置、以及封装接口函数调用即可实现所需功能，节约时间，提高开发效率。应用库提供了基于串口多种收发方式:DMA、定时器、单字节收发、中断收发。

使用方式：

需要将应用层 uart_demo.c uart_demo.h 驱动层 uart.c uart.h、gpio.c gpio.h、sci_common.c sci_common.h、isr.c isr.h 加入到工程中去；若搭配使用 DMA、TIM，则需要将相应驱动文件以及 demo 程序加入。

3.1.应用例程使用

包括 UART 初始化，读写函数以及 UART 相关接口

3.1.1. UART 初始化

串口初始化，传入参数为波特率，返回值为初始化结果：当通道被占用/通道中断被占用初始化将失败。

```
/*****************************************
* Function Name: Uart0_Init
* @brief  UART0 init demo
* @param  bound
* @return init status
*****************************************************/
int8_t Uart0_Init(uint32_t bound)
{
 int8_t ret;
    GPIO_InitTypeDef GPIO_InitStruct ={0};
 UART_InitTypeDef UART_InitStructure ={0};

 GPIO_PinAFConfig(GPIO_PORT5,GPIO_Pin_1,GPIO_P51,GROUP_AF_ODEFAULT);
 GPIO_PinAFConfig(GPIO_PORT5,GPIO_Pin_0,GPIO_P50,GROUP_AF_ ODEFAULT);

 /*TX GPIO CONFIG*/
 GPIO_InitStruct.GPIO_Pin    = GPIO_Pin_1;
 GPIO_InitStruct.GPIO_Mode   = GPIO_Mode_OUT;
 GPIO_InitStruct.GPIO_Ctrl  = GPIO_Control_DIG;
 GPIO_Init(GPIO_PORT5,&GPIO_InitStruct);

 /*RX GPIO CONFIG*/
 GPIO_InitStruct.GPIO_Pin = GPIO_Pin_0 ;
 GPIO_InitStruct.GPIO_Mode = GPIO_Mode_IN;
 GPIO_InitStruct.GPIO_Ctrl = GPIO_Control_DIG;
 GPIO_Init(GPIO_PORT5,&GPIO_InitStruct);

 /*USART CONFIG*/
 UART_InitStructure.UART_BaudRate = bound;
 UART_InitStructure.UART_WordLength = UART_WordLength_8b;
 UART_InitStructure.UART_StopBits = UART_StopBits_1;//一个停止位
 UART_InitStructure.UART_Parity = UART_Parity_No;//无奇偶校验位
 UART_InitStructure.phase = UART_Phase_Normal;
 UART_InitStructure.bitorder = UART_Bit_LSB;
```

```
35.  UART_InitStructure.UART_Mode = UART_Mode_Rx | UART_Mode_Tx; // 收发模式
36.
37.  ret = UART_Init(UART0, &UART_InitStructure); //初始化串口
38.  if(ret)
39.  {
40.   SCI_ERROR_LOG(ret);
41.   return ret;
42.  }
43.  ISR_Register(ST0_IRQn,uart0_interrupt_send);      //串口 0 发送中断服务路径注册
44.  ISR_Register(SR0_IRQn,uart0_interrupt_receive);   //串口 0 接收中断服务路径注册
45.
46.     INTC_EnableIRQ(ST0_IRQn);        // enable INTST1 interrupt
47.     INTC_EnableIRQ(SR0_IRQn);        // enable INTSR1 interrupt
48.  return SCI_SUCCESS;
49. }
```

- GPIO 复用，调用 GPIO_PinAFConfig 函数，其可以将普通 GPIO 引脚复用为数字功能，对于 TXD0/RX0, TXD1/RX1, TXD2/RX2 可以映射到任意管脚；比如 GROUP_AF_TXD0 功能可以映射到 P51 引脚，也可以映射到 P12 P00……等
- 配置串口相关参数，包括波特率、传输长度、发送停止位、奇偶校验、输出相位、以及收发方式
- 中断服务函数注册，中断收发，将 uart0_interrupt_send 注册到中断号为 ST0_IRQ 的中断函数函数中去，uart0_interrupt_receive 注册到中断号为 SR0_IRQ 的中断函数中。用户在 uart_demo.c 中可以自己改写中断服务函数 uart0_interrupt_send，uart0_interrupt_receive；若不使用中断方式进行收发则不用注册中断服务函数

3.1.2. UART 轮询收/发函数

对于 UART 接口读写函数，UART 接口提供了 3 种读写方式实现 UART 数据传输；第一种为轮询方式：

发送函数：UART_SendByte(SCIAFSelect_TypeDef UARTx, uint8_t Data)

```
1.  void UART_SendByte(SCIAFSelect_TypeDef UARTx, uint8_t Data)
2.  {
```

```
3.   if (UARTx == UART0)
4.   {
5.    UART0_TypeDef *UART_Instance = &SCI0->UART0;
6.    UART_Instance->TXD = Data;
7.    while (UART_GetFlagStatus(UART_Instance->TSSR, UART_FLAG_TSF | UAR
    T_FLAG_BFF));
8.   }
9.   else if (UARTx == UART1)
10.  {
11.   UART1_TypeDef *UART_Instance = &SCI0->UART1;
12.      UART_Instance->TXD = Data;
13.   while (UART_GetFlagStatus(UART_Instance->TSSR, UART_FLAG_TSF | UAR
    T_FLAG_BFF));
14.  }
15.  else if (UARTx == UART2)
16.  {
17.   UART2_TypeDef *UART_Instance = &SCI1->UART2;
18.      UART_Instance->TXD = Data;
19.   while (UART_GetFlagStatus(UART_Instance->TSSR, UART_FLAG_TSF | UAR
    T_FLAG_BFF));
20.  }
21. }
```

轮询发送，选择使用的串口号，等待单字节数据发送成功之后才会继续发送下一个数据

接收函数 UART_ReceiveByte(SCIAFSelect_TypeDef UARTx)

```
1.  char UART_ReceiveByte(SCIAFSelect_TypeDef UARTx)
2.  {
3.   char ch = 0;
4.
5.   if (UARTx == UART0)
6.   {
7.    UART0_TypeDef *UART_Instance = &SCI0->UART0;
8.       while (!UART_GetFlagStatus(UART_Instance->RSSR, UART_FLAG_BFF));
9.    ch = UART_Instance->RXD;
10.  }
11.  else if (UARTx == UART1)
12.  {
13.   UART1_TypeDef *UART_Instance = &SCI0->UART1;
14.      while (!UART_GetFlagStatus(UART_Instance->RSSR, UART_FLAG_BFF));
15.   ch = UART_Instance->RXD;
16.  }
```

```
17.  else if (UARTx == UART2)
18.  {
19.   UART2_TypeDef *UART_Instance = &SCI1->UART2;
20.      while (!UART_GetFlagStatus(UART_Instance->RSSR, UART_FLAG_BFF));
21.   ch = UART_Instance->RXD;
22.  }
23.
24.     return ch;
25. }
```

等待 UART 接收缓冲区有数据时，将数据读取出来

3.1.3. UART 中断收/发函数

对于 CMS32L051 系列的 UART，中断收发函数，在中断服务函数中发送或者获取数据。

中断发送函数 Uart0_IntSend

```
1.  void Uart0_IntSend(uint8_t * tx_buf, uint16_t tx_num)
2.  {
3.   pData.data = tx_buf;
4.   pData.len = tx_num;
5.
6.   INTC_SetPendingIRQ(ST0_IRQn);
```

对于中断发送函数，其功能只是将数据给到中断发送服务函数，通过发送中断服务发送出去，每发送一个数据触发一次发送中断；发送中断服务函数如下：

```
1.  void uart0_interrupt_send(void *msg)
2.  {
3.      ATE_FRAME_t *pFrame =(ATE_FRAME_t *)msg;
4.      INTC_ClearPendingIRQ(ST0_IRQn);
5.      if((pFrame->len > 0U) && pFrame->data)
6.      {
7.          U0_TX = *pFrame->data;
8.          pFrame->data++;
9.          pFrame->len --;
10.     }
11.     else //send finished
12.     {
13.     }
14. }
```

中断接收服务函数 uart1_interrupt_receive

```
/************************************************************
* Function Name: uart1_interrupt_receive
* @brief  UART1 Receive interrupt service routine
* @param  None
* @return None
************************************************************/
void uart1_interrupt_receive(void *msg)
{
 volatile uint8_t rx_data;
 volatile uint8_t err_type;

 INTC_ClearPendingIRQ(SR1_IRQn);
 err_type = UART_GetErrStaus(UART1, UART_FLAG_FEF | UART_FLAG_PEF | UART_FLAG_OVF);

 if (err_type)
 {
  uart_callback_error(err_type);
 }

 rx_data = UART1_RX;

 if((UART1_RX_STA & 0x8000U) == 0) //接收未完成
 {
  if(UART1_RX_STA &0x4000U)  //接收到0x0d
  {
   if(rx_data !=0x0a)
    UART1_RX_STA = 0;
   else
   {
    UART1_RX_STA|=0x8000;
    UART1_RX_BUF[UART1_RX_STA&0x3fff] = rx_data;
    UART1_RX_STA ++;
   }
  }
  else                          //还未接收到0x0d
  {
   if(rx_data == 0x0d)
   {
    UART1_RX_STA |=0x4000;
    UART1_RX_BUF[UART1_RX_STA&0x3fff] = rx_data;
    UART1_RX_STA ++;
   }
```

```
43.    else
44.    {
45.     UART1_RX_BUF[UART1_RX_STA&0x3fff] = rx_data;
46.     UART1_RX_STA ++;
47.    }
48.   }
49.  }
50.  else if((UART1_RX_STA & 0x8000U) == 1) //received finished
51.  {
52.  }
53. }
```

通常我们都在中断接收服务函数中去接收数据，当接收完成之后给外部一个接收完成标志位；这部分通常由客户自己来编写；在所给的例程中提供了如下几种接收数据的处理：

- 接收数据是以 0x0D 0x0A 结束的（见串口 1 例程）
- 接收数据为不定长，使用 DMA 方式（见串口 0 例程）
- 接收数据不定长，超时接收，即串口间隔超过 20ms 没有数据则认为数据结束（见串口 2 例程）

3.1.4. UART DMA 收/发函数

DMA 接收不定长数据，配置 uart0 的 DMA 接收 DMA_Uart0_Rx，DMA 触发源 DMA_VECTOR_SR0，重复模式，源地址为 U0_RX 目的地址 UART0_RX_BUF，最大接收长度为 USART_MAX_RECV_LEN;将 U0_RX 接收到的数据搬移到 UART0_RX_BUF 中去。

```
1.  /*串口 0 通过 DMA 接收不定长数据*/
2.   DMA_Uart0_Rx（DMA_VECTOR_SR0，DMA_Mode_Repeat，（void *）&U0_RX，
    UART0_RX_BUF，USART_MAX_RECV_LEN）;//配置 dma 传输
3.   INTC_SetPendingIRQ（SR0_IRQn）;
```

DMA 使用的是重复模式，因此对 UART0_RX_BUF 写数据是环形 buffer 写方式。

Uart0_Dma_Rcv 函数使用环形 buff 方式读取 UART0_RX_BUF 中的数据。

```
1.  uint8_t Uart0_Dma_Rcv(uint8_t *buf)
2.  {
3.   static uint16_t read = 0,write = 0;
4.   static uint16_t raw_len = USART_MAX_RECV_LEN;
5.   int16_t readlen =0;
```

```
6.
7.   if(buf ==NULL)
8.   {
9.    return 0;
10.  }
11.  write = raw_len - DMA_TRANS_TIMES;  //写指针位置
12.  readlen = write -read;                    //读数据长度
13.  if(!readlen)
14.  {
15.   return 0;
16.  }
17.  if(readlen  < 0)
18.  {
19.   readlen +=raw_len;
20.  }
21.
22.  if(readlen < (raw_len - read))      //无需折返
23.  {
24.   memcpy(buf,&UART0_RX_BUF[read],readlen);
25.  }
26.  else
27.  {
28.   memcpy(buf,&UART0_RX_BUF[read],raw_len -read);
29.   memcpy(&buf[raw_len -read],UART0_RX_BUF,readlen -(raw_len -read));
30.  }
31.  read = (readlen+read)%raw_len;   //更新读指针
32.  return readlen;
33. }
```

DMA 发送方式，配置 uart0 的 DMA 发送 DMA_Uart0_Tx，DMA 触发源 DMA_VECTOR_ST0，普通模式，源地址为 tx_buf，目的地址 U0_TX，最大发送长度为 tx_num；将 tx_buf 数据搬移到串行总线&U0_TX。

```
1.  void Uart0_Dma_Send(uint8_t * tx_buf, uint16_t tx_num)
2.  {
3.   DMA_Uart0_Tx(DMA_VECTOR_ST0,DMA_Mode_Normal,(void *)tx_buf,(void *)
    &U0_TX,tx_num); //config dma transmission
4.   DMA_Trigger(DMA_VECTOR_ST0);
```

3.1.5. UART 开始/关闭运行

```
1.  void UART_Cmd(SCIAFSelect_TypeDef UARTx, FunctionalState NewState)
```

ENABLE：打开 UART

DISABLE：关闭 UART 运行

3.1.6. UART 获取运行状态

```
1.  FlagStatus UART_GetErrStaus(SCIAFSelect_TypeDef UARTx, uint16_t UART
    _FLAG)
```

根据 UART_FLAG 获取 UART 指定的接收状态

3.1.7. 关闭 UART 外设

```
1.  void UART_DeInit(SCIAFSelect_TypeDef UARTx)
```

将 UART 外设关闭，包括关闭其运行时钟，释放 UART 所占用通道

3.3. UART 例程示例

在 main 函数中演示了：1、串口 0 的 DMA 收发模式，2、串口 1 中断接收以 0X0D 0X0A 数据结尾的数据，通过中断方式发送出去 3、串口 2 中断接收不定长数据（若每个 BYTE 之间超过 20ms，则认为接收一包数据结束），中断发送

```
45
46      Uart0_Init(115200);
47  #ifdef UART0_DMA_RCV
48      /*串口0通过DMA接收不定长数据*/
49      DMA_Uart0_Rx(DMA_VECTOR_SR0,DMA_Mode_Repeat, (void *)&U0_RX,UART0_RX_BUF,USART_MAX_RECV_LEN); //config dma t
50      INTC_SetPendingIRQ(SR0_IRQn);
51  #endif
52      Uart1_Init(38400);
53
54      Uart2_Init(9600);
55
56
57      while(1)
58      {
59          delayMS(100);
60
61  #ifdef UART0_DMA_RCV
62          len = Uart0_Dma_Rcv(rxbuf);
63          if(len)
64          {
65              Uart0_Dma_Send(rxbuf,len);
66          }
67  #else
68          if(UART0_RX_STA & 0x8000U)
69          {
70  //          Uart0_IntSend(UART0_RX_BUF,UART0_RX_STA&0x3fff); //中断发送
71  //          Uart0_Dma_Send(UART0_RX_BUF,UART0_RX_STA&0x3fff);//DMA 发送
72              for(i=0;i<(UART0_RX_STA&0x3fff);i++)            //轮询发送
73              {
74                  Uart0_Send(UART0_RX_BUF[i]);
75              }
76              UART0_RX_STA =0;
77          }
78  #endif
79          if(UART1_RX_STA & 0x8000U)      //如果接收完成
80          {
81              Uart1_IntSend(UART1_RX_BUF,UART1_RX_STA&0x3fff);
82              UART1_RX_STA =0;
83          }
84
85          if(UART2_RX_STA & 0x8000U)
86          {
87              Uart2_IntSend(UART2_RX_BUF,UART2_RX_STA&0x3fff);
88              UART2_RX_STA =0;
89          }
90      }
```

1

2

3

4. 示例演示

串口数据正常收发