先在这里讲缓冲区（队列结构）算法

             |-------------------|

head-->|-------------------|

自上而下地址递增

             |-------------------|

             |-------------------|

             |-------------------|

    tail-->|-------------------|

             |-------------------|

             |-------------------|

             |-------------------|

             |-------------------|

             |-------------------|

head和tail都是一个方向移动,当移到末位就回到开头位置.这个很重要。

读写函数不是真正的发送接收函数,他们只是负责将数填入上述的缓冲区,或者从缓冲区中读出。

怎么操作呢? 读写函数只是将数填入或读出缓冲区,然后移动其中一个指针..

那么什么时候结束呢? 就是head==tail..

这里还有,另一个指针是在发送接收中断函数中移动。这是这种你追我跑的方式发送接收.

这里开始读写函数

（1）读操作函数（在函数HalUARTRead中调用HalUARTReadISR函数）

1. /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*
2. \* @fn      HalUARTReadISR
3. \*
4. \* @brief   Read a buffer from the UART
5. \*
6. \* @param   buf  - valid data buffer at least 'len' bytes in size
7. \*          len  - max length number of bytes to copy to 'buf'
8. \*
9. \* @return  length of buffer that was read
10. \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/
11. **static** uint16 HalUARTReadISR(uint8 \*buf, uint16 len)
12. {
13. uint16 cnt = 0;
15. **while** ((isrCfg.rxHead != isrCfg.rxTail) && (cnt < len))   //这是读结束条件, isrCfg.rxHead 与 isrCfg.rxTail指向的空间还没有读出来或被写进去
16. { // isrCfg.rxHead = isrCfg.rxTail意味着指向的空间没有数据可以读出

     \*buf++ = isrCfg.rxBuf[isrCfg.rxHead++];//读操作，变化的是 //rxHead

1. **if** (isrCfg.rxHead >= HAL\_UART\_ISR\_RX\_MAX) //超过了队尾，再回到
2. {   //队首
3. isrCfg.rxHead = 0;
4. }
5. cnt++;  //读出的数据个数加一
6. }
8. **return** cnt;
9. }

（2）写操作函数（在函数HalUARTWrite中调用HalUARTWriteISR函数）

1. /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*
2. \* @fn      HalUARTWriteISR
3. \*
4. \* @brief   Write a buffer to the UART.
5. \*
6. \* @param   buf - pointer to the buffer that will be written, not freed
7. \*          len - length of
8. \*
9. \* @return  length of the buffer that was sent
10. \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/
11. **static** uint16 HalUARTWriteISR(uint8 \*buf, uint16 len)
12. {
13. uint16 cnt;
15. // Enforce all or none.

if (HAL\_UART\_ISR\_TX\_AVAIL() < len) //判断len长度是否大于当前有效长度，大

//于表示缓冲区放不下，则返回0

{

1. **return** 0;
2. }
4. **for** (cnt = 0; cnt < len; cnt++)
5. {
6. isrCfg.txBuf[isrCfg.txTail] = \*buf++;
7. isrCfg.txMT = 0;  //发送缓冲区是否为空的标志，为1为空，为0不为空

  if (isrCfg.txTail >= HAL\_UART\_ISR\_TX\_MAX-1) //缓冲区到了队尾， 又重新回到//队首

1. {
2. isrCfg.txTail = 0;
3. }
4. **else**
5. {
6. isrCfg.txTail++; //往缓冲区写数据，变化的是txTail
7. }
9. // Keep re-enabling ISR as it might be keeping up with this loop due to other ints.
10. IEN2 |= UTXxIE;  //发送缓冲区有数据后，就允许串口发送中断
11. }
13. **return** cnt;
14. }

这里是中断服务程序:

（1）串口接收中断函数

1. /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*
2. \* @fn      halUartRxIsr
3. \*
4. \* @brief   UART Receive Interrupt
5. \*
6. \* @param   None
7. \*
8. \* @return  None
9. \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/
10. #if (HAL\_UART\_ISR == 1)
11. HAL\_ISR\_FUNCTION( halUart0RxIsr, URX0\_VECTOR )
12. #else
13. HAL\_ISR\_FUNCTION( halUart1RxIsr, URX1\_VECTOR )
14. #endif
15. {
16. HAL\_ENTER\_ISR();
18. uint8 tmp = UxDBUF;
19. isrCfg.rxBuf[isrCfg.rxTail] = tmp;//把串口接收到的数据放入到接收缓//冲区里
21. // Re-sync the shadow on any 1st byte received.
22. **if** (isrCfg.rxHead == isrCfg.rxTail)//是接收缓冲区唯一的一个数据
23. {   //接着需要等待6ms才提醒缓冲区里来了新数据
24. isrCfg.rxShdw = ST0;  //用于计算等待时间是否到了
25. }

if (++isrCfg.rxTail >= HAL\_UART\_ISR\_RX\_MAX) //到了缓冲区的队尾，得要重新

//返回到队首位置，如此不断循环下去

1. {
2. isrCfg.rxTail = 0;
3. }
5. isrCfg.rxTick = HAL\_UART\_ISR\_IDLE;  //需要等待6ms时间，再提醒接收 //缓冲区来了新数据
7. HAL\_EXIT\_ISR();
8. }

（2）串口发送中断函数

1. /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*
2. \* @fn      halUartTxIsr
3. \*
4. \* @brief   UART Transmit Interrupt
5. \*
6. \* @param   None
7. \*
8. \* @return  None
9. \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/
10. #if (HAL\_UART\_ISR == 1)
11. HAL\_ISR\_FUNCTION( halUart0TxIsr, UTX0\_VECTOR )
12. #else
13. HAL\_ISR\_FUNCTION( halUart1TxIsr, UTX1\_VECTOR )
14. #endif
15. {
16. HAL\_ENTER\_ISR();

if (isrCfg.txHead == isrCfg.txTail)//表示发送缓冲区为空，没有数据可 //发

1. {
2. IEN2 &= ~UTXxIE;  //既然没有发送缓冲区为空，就不能允许串口发送中断
3. isrCfg.txMT = 1;  //发送缓冲区为空的标志
4. }
5. **else**
6. {
7. UTXxIF = 0;
8. UxDBUF = isrCfg.txBuf[isrCfg.txHead++]; //将发送缓冲区的数据放在
9. // UxDBUF，准备发送出去
10. **if** (isrCfg.txHead >= HAL\_UART\_ISR\_TX\_MAX)//当超过了队尾，必须重 //新回到队首，如此不断循环
11. {
12. isrCfg.txHead = 0;
13. }
14. }
16. HAL\_EXIT\_ISR();
17. }

 另外， 还有一个重要的函数：static void HalUARTPollISR(void)

这个函数是在OSAL每次循环中都调用一次,主要是设置串口事件,调用相应的回调函数.

1. **static** **void** HalUARTPollISR(**void**)
2. {
3. uint16 cnt = HAL\_UART\_ISR\_RX\_AVAIL();
4. uint8 evt = 0;
6. **if** (isrCfg.rxTick)  //剩下的等待时间
7. {
8. // Use the LSB of the sleep timer (ST0 must be read first anyway).
9. uint8 decr = ST0 - isrCfg.rxShdw; //表示已经过去的时间段
11. **if** (isrCfg.rxTick > decr)  //等待时间还没到
12. {
13. isrCfg.rxTick -= decr;  //还剩下的等待时间
14. }
15. **else**   //等待时间到了
16. {
17. isrCfg.rxTick = 0;
18. }
19. }
20. isrCfg.rxShdw = ST0;  //用于下一次等待时间段的计算
22. **if** (cnt >= HAL\_UART\_ISR\_RX\_MAX-1) //当缓冲区里还剩下一个字节空间可//放数据的时候，表示缓冲区满了
23. {
24. evt = HAL\_UART\_RX\_FULL;
25. }
26. **else** **if** (cnt >= HAL\_UART\_ISR\_HIGH)//整个缓冲区里还有近一半的数据
27. {   //缓冲区快要满了
28. evt = HAL\_UART\_RX\_ABOUT\_FULL;
29. }
30. **else** **if** (cnt && !isrCfg.rxTick) //串口接收缓冲区里有新数据,但需等待
31. {   //6ms时间，时间一到，才提醒有新数据来了
32. evt = HAL\_UART\_RX\_TIMEOUT;
33. }
35. **if** (isrCfg.txMT)//在串口发送中断函数那边表示发送缓冲区为空，没有数据
36. {   //可发了
37. isrCfg.txMT = 0;
38. evt |= HAL\_UART\_TX\_EMPTY;
39. }
41. **if** (evt && (isrCfg.uartCB != NULL))//有串口事件并且串口回调函数已经 //注册过
42. {
43. isrCfg.uartCB(HAL\_UART\_ISR-1, evt);  //调用回调函数
44. }
45. }

1、串口读操作的正确顺序是：先执行串口接收中断函数HAL\_ISR\_FUNCTION( halUart0RxIsr, URX0\_VECTOR )，再执行读操作服务函数HalUARTReadISR(uint8 \*buf, uint16 len) ，其结构示意图为(有2个数组，一个是左边的isrCfg.rxBuf，它有2个指针，一个是rxTail，用来添加数据，另一个是rxHead，用来读出数据的，右边的数组是buf)：

isrCfg.rxTail是往数组isrCfg.rxBuf添加数据的指针

是通过下面两条语句对数组isrCfg.rxBuf添加数据：  tmp = UxDBUF;  isrCfg.rxBuf[isrCfg.rxTail] = tmp;

数组isrCfg.rxBuf是为了保存串口接收过来的数据

isrCfg.rxHead是从数组isrCfg.rxBuf读出数据的指针

是通过下面语句从数组buf读出数据传递给数组isrCfg.rxBuf：\*buf++ = isrCfg.rxBuf[isrCfg.rxHead++];

数组buf是为了把数组isrCfg.rxBuf的数据转移出来以备后面使用，并且数组isrCfg.rxBuf就可以腾出空间继续存放通过串口接收来的数据

数组数据传递方向,从左往右

|  |
| --- |
|  |
| ┇ |
|  |
|  |
|  |
| ┇ |
|  |

|  |
| --- |
|  |
| ┇ |
|  |
|  |
|  |
| ┇ |
|  |

2、串口写操作的正确顺序是：先执行写操作服务函数HalUARTWriteISR(uint8 \*buf, uint16 len) ，再执行串口发送中断函数HAL\_ISR\_FUNCTION( halUart0TxIsr, UTX0\_VECTOR )，其结构示意图为(有2个数组，一个是左边的isrCfg.txBuf，它有2个指针，一个是txTail，用来添加数据，另一个是txHead，用来读出数据的，右边的数组是buf)：

isrCfg.txTail是往数组isrCfg.txBuf写数据的指针

是通过下面一条语句从数组isrCfg.txBuf读出数据添加到UxDBUF等待通过串口发出：UxDBUF = isrCfg.txBuf[isrCfg.txHead++];

数组isrCfg.txBuf是为了保存即将从串口发送出去的数据

isrCfg.txHead是从数组isrCfg.txBuf读出数据的指针

是通过下面两条语句从数组buf读出数据传递给数组isrCfg.txBuf： isrCfg.txBuf[isrCfg.txTail] = \*buf++; isrCfg.txTail++;

数组buf的功能是保存即将发出的数据，先要把它里面的数据传给数组isrCfg.txBuf，以便数组buf就可以腾出空间继续存放即将要发送出去的数据

数组数据传递方向,从右往左

|  |
| --- |
|  |
| ┇ |
|  |
|  |
|  |
| ┇ |
|  |

|  |
| --- |
|  |
| ┇ |
|  |
|  |
|  |
| ┇ |
|  |