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

BAT32G137 芯片上带有 CAN 控制器，符合 ISO 11898 中标准的 CAN 协议。CAN 协议基础知识：

1. 具有多主控制（无中心、总线空闲任一时间均可竞争发送消息，消息广播，由节点仲裁决定是否过滤）

2. 若软性（节点与总线相连的单元没有类似于地址的信息，增加节点时软硬件都不需改变）

3. 错误检测（节点检测：stuffcheck(接收节点检测位填充错误，6 个同极性位就报错)、 formcheck(接收节点检测 crc 界定符和 ack 界定符以及 eof 区域是否出现显性位)、 bit monitorint(发送节点发送显性位、而总线隐性位报错)、crc(接收方快速生成值与发送 crc 比对) ack check(接收方在收到消息后会在 ack 应答位给出显性电平，发送方检测到该位为隐性报错)）等功能。

2. CAN 控制器特征

- 符合 ISO 11898 并且按照 ISO/DIS 16845(CAN 符合性)测试
- 使用标准帧和扩展帧实现接收和发送
- 通信速度最大：1Mbps
- 1 个通道有 16 个报文缓存
- 自动块传输功能
- 多缓存接收块功能
- 每个通道 4 种模式屏蔽设置

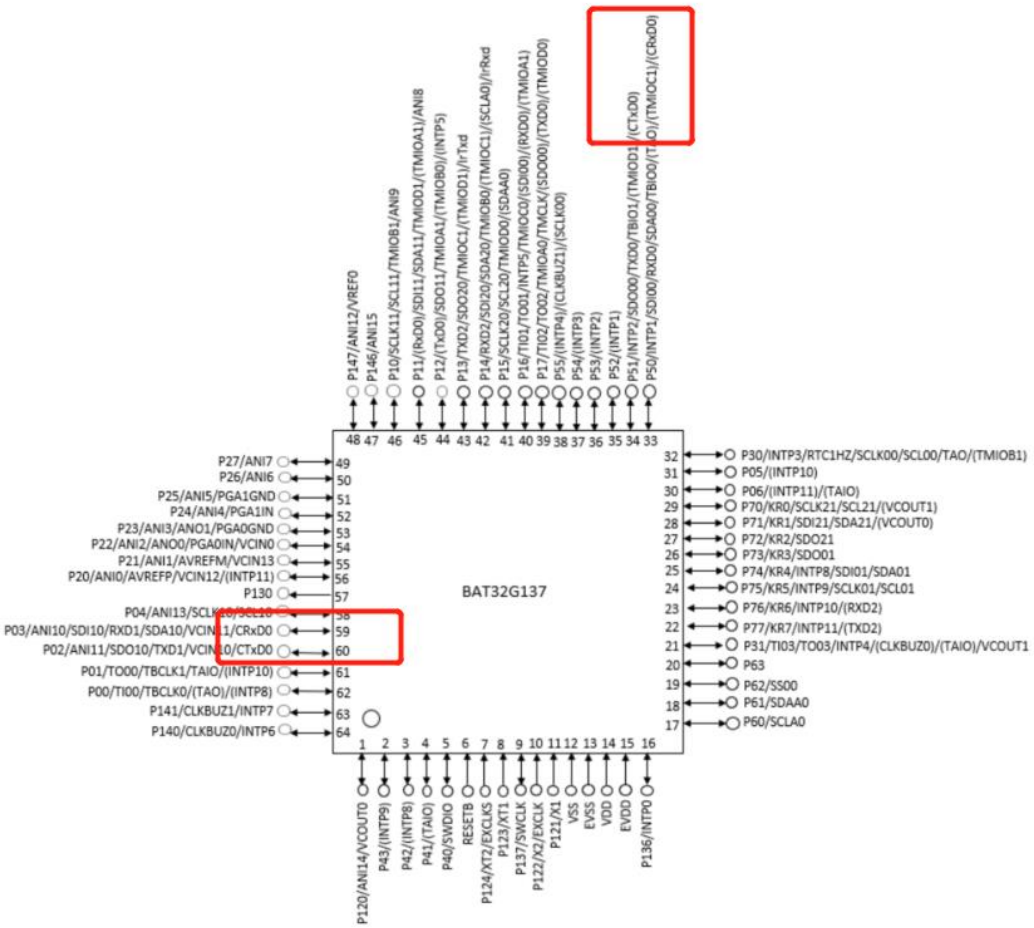

图 1 BAT32G137 64PIN CAN 引脚分配

对于 BAT32G137 CRxD0/CTxD0 是 CAN 控制器连接 CAN 收发器的两个引脚，对于 P03 P02 引脚 CRxD0/CTxD0 默认功能，P51 P50 引脚可以复用作为 CRxD0/CTxD0 功能。

3. 中微 BAT32G137 系列 CAN 应用库简介

中微 BAT32G137 系类软件 CAN 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。应用库提供了基于多种收发方式：轮询、中断方式收发。

使用方式：

需要将应用层 can_demo.c can_demo.h 驱动层 can.c can.h、gpio.c gpio.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 CAN 初始化，读写函数以及 CAN 相关接口

3.1.1. CAN 初始化

CAN 初始化，定义 CAN 控制器的引脚以及 CAN 波特率、以及通道屏蔽设置。

```
1.  void CAN_Port_Config(void)
2.  {
3.   GPIO_InitTypeDef GPIO_InitStruct;
4.   CAN_InitTypeDef  CAN_InitStruct;
5.
6.   /* GPIO AF set for CAN bus */
7.   GPIO_PinAFConfig(GPIO_PORT0, GPIO_Pin_2, GPIO_P02, GROUP_AF_CTXD);
8.   GPIO_PinAFConfig(GPIO_PORT0, GPIO_Pin_3, GPIO_P03, GROUP_AF_CRXD);
9.
10.  /****** GPIO for CAN bus init ******/
11.  /* CTXD pin init */
12.     GPIO_InitStruct.GPIO_Pin    = GPIO_Pin_2 ;
13.     GPIO_InitStruct.GPIO_Mode   = GPIO_Mode_OUT;
14.  GPIO_InitStruct.GPIO_Level  = GPIO_Level_HIGH;
15.  GPIO_InitStruct.GPIO_Ctrl   = GPIO_Control_DIG;
16.     GPIO_Init(GPIO_PORT0, &GPIO_InitStruct);
17.
18.  /* CRXD pin init */
19.     GPIO_InitStruct.GPIO_Pin    = GPIO_Pin_3 ;
20.     GPIO_InitStruct.GPIO_Mode   = GPIO_Mode_IN;
21.  GPIO_InitStruct.GPIO_Ctrl   = GPIO_Control_DIG;
22.     GPIO_Init(GPIO_PORT0, &GPIO_InitStruct);
23.
24.  /****** GPIO for STB init ******/
25.  GPIO_InitStruct.GPIO_Pin    = GPIO_Pin_0;
26.  GPIO_InitStruct.GPIO_Mode   = GPIO_Mode_OUT;
27.  GPIO_InitStruct.GPIO_Ctrl   = GPIO_Control_DIG;
28.  GPIO_InitStruct.GPIO_PuPd   = GPIO_PuPd_DOWN;
29.  GPIO_InitStruct.GPIO_Level  = GPIO_Level_LOW;
30.  GPIO_Init(GPIO_PORT12, &GPIO_InitStruct);
31.
32.  /****** CAN bus periphal init ******/
33.  CAN_InitStruct.CAN_Prescaler = 2; //32M/2=16MHz
34.  CAN_InitStruct.CAN_BitRatePrescaler = 2;//16MHz/2 = 8MHz
```

```
35.  CAN_InitStruct.CAN_SJW = CAN_SJW_2tq;  //再同步补偿段
36.  CAN_InitStruct.CAN_BS1 = CAN_BS1_14tq; //传播段+相位段1
37.  CAN_InitStruct.CAN_BS2 = CAN_BS2_1tq;  //相位段2 8MHZ/6 = 500Kbps
38.  CAN_InitStruct.CAN_OperateMode = CAN_OpMode_Initial;
39.  CAN_InitStruct.MASK1 = 0x1fffffffU;
40.  CAN_InitStruct.MASK2 = 0x1fffffffU;
41.  CAN_InitStruct.MASK3 = 0x1fffffffU;
42.  CAN_InitStruct.MASK4 = 0x1fffffffU;
43.  CAN_Init(CAN0, &CAN_InitStruct);
44. }
```

- 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 CTXD0/CRX0,可以映射到指定管脚；
- 配置 STB 控制引脚，根据开发板的原理图，P120 作为 STB 控制引脚
- 配置 CAN 分频因子，比如主频为 32Mhz，则 2 分频为 16Mhz
- 配置 CAN 位速率分频因子，如果设置 CAN 位速率分频因子为 2，则 CAN 时钟为 8Mhz
- 分别设置 SJW/再补偿段、BS1/(传播段+相位段 1)、BS2/(相位段 2)的时间，其中 Tq 为最小时间单位构成，因此 CAN 波特率为 8Mhz/16 =500Kbps
- 初始化模式为 CAN_OpMode_Initial 模式
- 设置通道的屏蔽功能，通过设置掩码，根据其屏蔽位来减少报文 ID 比较，从而允许将多个不同的 ID 接收到一个缓冲区种，接收报文中掩码定义位 1 的标识位不会与报文缓冲区中的相应标识位进行比较

3.1.2. CAN 报文设置以及初始化

通过设置报文，来确定所要发送或者接收所使用的具体帧类型

```
1.   CAN_SendMsg.IDE = CAN_Id_Standard;
2.   CAN_SendMsg.Id = 0x730;
3.   CAN_SendMsg.DLC = 8;
4.   CAN_SendMsg.CacheType = CAN_CacheType_Tx;
5.   CAN_SendMsg.RTR = CAN_RTR_Data;
6.   CAN_SendMsg.Interrupt = DISABLE;
7.   for (int i = 0; i < 8; i++)
8.   {
9.    CAN_SendMsg.Data[i] = i  * 5;
10.  }
```

- 设置是标准格式还是扩展格式，对于数据帧和遥控帧有标准格式和扩展格式两种
- 设置报文 ID，标准格式有 11 位标识符，扩展格式有 29 个位的 ID
- 设置数据长度为 8
- 设置报文缓存类型为发送
- 设置报文为数据帧
- 报文中断为失能，不打开报文中断
- 赋值要发送的具体数据

报文缓存初始化，启用 CAN 模块后，缓冲区包含未定义值，将 CAN 模块切换到其中某种模式（正常模式/正常模式伴 ABT/仅接收模式/单次模式/自检模式）前，需要为报文缓冲区进行最小初始化。

```
/**
  * @brief  CAN message cache register init.
  * @param  CANxMSGy: where x can be 0 to select the CAN peripheral.
  *                   where y can be 0 to 15 to select the cache.
  * @param  TxRxMessage: the message for tx or rx message to change RTR ID and CacheType.
  * @retval status of the requested mode which can be
  *         - CAN_MsgcacheInit_Failed:  CAN failed to init message cache
  *         - CAN_MsgcacheInit_Success: CAN Succeed to init message cache
  */
uint8_t CAN_MessageCache_Init(CANMSG_Type *CANxMSGy, CanTxRxMsg *TxRxMessage)
{
    /* Check the parameters */
    assert_param(IS_CAN_ALL_MSGCACHE(CANxMSGy));
    assert_param(IS_CAN_CACHETYPE(TxRxMessage->CacheType));
    assert_param(IS_CAN_IDTYPE(TxRxMessage->IDE));
    assert_param(IS_CAN_RTR(TxRxMessage->RTR));
    assert_param(IS_CAN_DLC(TxRxMessage->DLC));

    /* Set message cache used flag */
 CANxMSGy->CMCONF |= CAN_MCONF_MA0;

    /* Set message cache type */
    CANxMSGy->CMCONF |= (TxRxMessage->CacheType << 3);
```

```
24.
25.     /* Message cache frame ID setting by IDType */
26.     if(TxRxMessage->IDE == CAN_Id_Extended)
27.     {
28.         CANxMSGy->CMIDL = TxRxMessage->Id & ((uint16_t)0xFFFF);
29.         CANxMSGy->CMIDH = ((TxRxMessage->Id >> 16U) & ((uint16_t)0xF
    FFF)) | ((uint16_t)0x8000);
30.     }
31.     else if(TxRxMessage->IDE == CAN_Id_Standard)
32.     {
33.         CANxMSGy->CMIDL = 0;
34.         CANxMSGy->CMIDH = (TxRxMessage->Id << 2U) & ((uint16_t)0x1FF
    C);
35.     }
36.
37.     /* RTR setting to select standard frame or remote frame */
38.     if (TxRxMessage->RTR == CAN_RTR_Remote)
39.     {
40.         CANxMSGy->CMCONF |=  CAN_MCONF_RTR;
41.     }
42.     else if (TxRxMessage->RTR == CAN_RTR_Data)
43.     {
44.         CANxMSGy->CMCONF &= ~CAN_MCONF_RTR;
45.     }
46.
47. /* When frame type is tx type, set frame data and length */
48. if (TxRxMessage->CacheType == CAN_CacheType_Tx)
49. {
50.  /* Set the frame data length */
51.  CANxMSGy->CMDLC = TxRxMessage->DLC;
52.
53.  /* Set the frame data value from TxRxMessage->Data */
54.         CANxMSGy->CMDB0 = TxRxMessage->Data[0];
55.         CANxMSGy->CMDB1 = TxRxMessage->Data[1];
56.         CANxMSGy->CMDB2 = TxRxMessage->Data[2];
57.         CANxMSGy->CMDB3 = TxRxMessage->Data[3];
58.         CANxMSGy->CMDB4 = TxRxMessage->Data[4];
59.         CANxMSGy->CMDB5 = TxRxMessage->Data[5];
60.         CANxMSGy->CMDB6 = TxRxMessage->Data[6];
61.         CANxMSGy->CMDB7 = TxRxMessage->Data[7];
62. }
63.
64. /* When the message cache interrupt enable, to set IE bit */
65. if (TxRxMessage->Interrupt != DISABLE)
```

```
66. {
67.  CANxMSGy->CMCTRL = CAN_MCTRL_SET_IE;
68. }
69. else
70. {
71.  CANxMSGy->CMCTRL = CAN_MCTRL_CLR_IE;
72. }
73.
74. /* When the message cache init, we should set RDY bit */
75. CANxMSGy->CMCTRL = CAN_MCTRL_SET_RDY;
76.
77. return CAN_MsgcacheInit_Success;
78. }
```

3.1.3. CAN 发送函数

```
1. /**
2.   * @brief  Initiates and transmits a CAN frame message.
3.   * @param  CANxMSGy: where x can be 0 to select the CAN peripheral.
4.   *                   where y can be 0 to 15 to select the cache.
5.   * @param  TxMessage: pointer to a structure which contains CAN Id,
    CAN DLC and CAN data.
6.   * @retval 0 is failed and true value is success for send data leng
   th.
7.   */
8. uint8_t CAN_Transmit(CANMSG_Type *CANxMSGy, CanTxRxMsg* TxMessage)
```

对于 CAN 发送函数需要在外部传入参数为 CAN 外设选择、CAN 报文缓冲区、以及发送报文帧

3.1.4. CAN 轮询接收函数

```
1. /**
2.   * @brief  A CAN frame message receive cache to memory to store.
3.   * @param  CANx: where x can be 0 to select the CAN peripheral.
4.   * @param  CANxMSGy: where x can be 0 to select the CAN peripheral.
5.   *                   where y can be 0 to 15 to select the cache.
6.   * @param  RxMessage: pointer to a structure which contains CAN Id,
    CAN DLC and CAN data.
7.   * @param  Timeout: wait timeout value for wait time.
```

```
8.    * @retval 0 is failed and true value is success for receive data l
    ength.
9.    * @note   This function is used by polling type.
10.   */
11. uint8_t CAN_Receive(CAN_Type* CANx, CANMSG_Type *CANxMSGy, CanTxRxMs
    g* RxMessage, uint32_t Timeout)
```

对于 CAN 轮询接收函数需要在 while 中调用，获取数据，外部传入参数为 CAN 外设选择、CAN 报文、以及超时时间

3.3. CAN 例程示例

4. 示例演示