1553B 总线控制器芯片

用户手册

（版本：V2.1）

珠海欧比特控制工程股份有限公司

地址:广东省珠海市唐家东岸白沙路 1 号欧比特科技园　邮编:519080

电话:0756-3391979　传真:0756-3391980　网址:www.myorbita.net

修改记录

版本变更	日期	修改内容	备注
V2.0	2012-06-17	对文件格式、版式等进行修订后，重新发布。	
V2.1	2014-06-17	对 1.4 节产品信息进行了修改	

目 录

表目录

图目录

1. 简介

1.1 概要说明

OBT1553B 芯片是依据 GJB289A-97 标准设计的，并且该芯片在操作方式、寄存器设置以及存储器布局等方面同 DDC 公司的 BU-61580 芯片兼容的一款具有完全自主知识产权的芯片。该芯片的可以通过修改寄存器的方式配置成总线控制器（Bus Controller）、RT(Remote Terminal)、BM(Bus Monitor)三种类型的控制器,内部带有 4*16Kbit 的双口 RAM。

OBT1553B 芯片主要应用在工业控制、船舶、航空和航天测控网络等技术领域。

1.2 结构框图

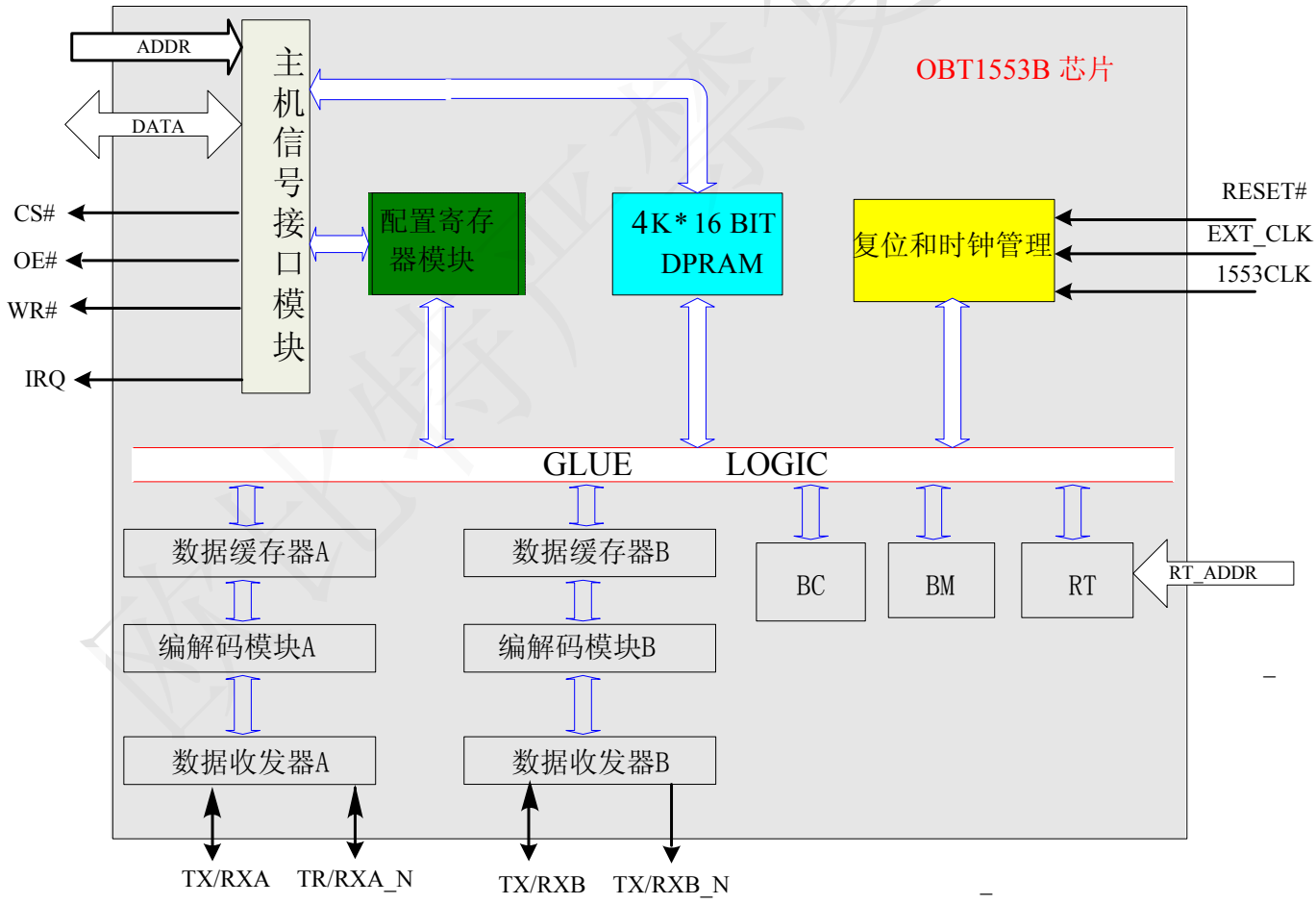

图 1-1 OBT1553B 芯片结构框图

1.3 主要特征

- 通过硬件逻辑方式完全实现 MIL-STD-1553B 标准(国军标 GJB289A-97 标准);
- 操作方式、寄存器设置以及存储器布局等方面同 BU-61580 兼容；
- 支持的通讯类型包括：
  - BC → RT；
  - RT → BC；
  - RT → RT；
  - Broadcast；
  - Mode code；
- 能被配置为 BC、RT、BM 三种类型的控制器；
- 带 4K*16Bit 的集成双口 RAM；
- 与主机接口模式为通用的异步接口；
- 芯片内部集成了总线收发器，不需外部再接收发器;
- 带 A、B 双冗余通道；
- BC 性能：
  - 支持 A/B 区域；
  - 具有自动重发功能；
  - 可编程的消息间隔时间；
  - 帧自动重复发送；
  - 可编程的超时响应时间；
- RT 性能：
  - 可编程的 RT 地址，子地址；
  - 支持单缓冲存储器管理方式；
  - 支持循环缓冲存储器管理方式；
  - 支持双缓冲存储器管理方式；
  - 可编程的非法命令表；
  - 可编程的方式代码中断表；
  - 可编程的子地址忙表；
- BM 性能：
  - 能够实时侦听总线上的数据流，可以将所有的数据流记录下来，也可以有选择地进行数据监听；
  - 支持命令堆栈半满、全满溢出；
  - 支持数据堆栈半满、全满溢出；
  - 命令堆栈与数据堆栈独立；
  - 对每条消息有相应的属性标志；

1.4 产品信息

表 1-1 OBT1553B 芯片产品信息

序号	产品型号	产品描述
1	OBT1553B-C-E	陶封 PGA70，工程样片
2	OBT1553B-C	陶封 PGA70，工业级芯片
3	OBT1553B-ASIC-IP-F	ASIC 版本固核（ASIC 网表）
4	OBT1553B-FPGA-IP-V	FPGA 版本固核（FPGA 网表）
5	OBT1553B-RTL-IP-S	软核（RTL 源码）

2. 主要功能模块概述

2.1 总线控制器（BC）

当主控制器配置为 BC 总线控制器时，则实现 BC 总线控制器功能。

BC 总线控制器控制通讯数据流的传输，是数据发送和接收的发起者和总线网络的管理者。控制计算机将数据写入 BC 总线控制器的内部存储器，并通过 开始/启动（SSR）寄存器来启动 BC 总线控制器进行数据传输。对于每个消息，BC 总线控制器通过 BC 控制字初始化 BC 总线控制器的状态从而发起数据传输（发送或者接收），并通过 BC 命令字通知 RT 响应数据传输（接收或者发送）。BC 总线控制器还可以通过模式命令对 RT 进行控制，包括读取同步和状态字等内容。

BC 总线控制器一方面通过 BC 模块状态字判断接受到的数据是否正确（包括奇偶校验）、响应是否超时等，另一方面通过读回的 RT 状态字，判断 RT 接收的数据是否正确、响应是否超时等。BC 模块状态字和读回的 RT 状态字均正常，说明数据传输正常。

如果传输过程中出现错误（BC 状态字和 RT 状态字异常），BC 总线控制器通过中断通知控制计算机进行处理，如消息重发。如果 BC 总线控制器出现灾难性故障，控制计算机（指 BC 的控制计算机）对 BC 复位。

2.2 远程终端（RT）

当主控制器配置为 RT 远程终端时，则实现 RT 远程终端功能。

RT 能根据协议在规定的时间内响应 BC 总线控制器发出的命令，进行数据接收或发送。RT 对输入信号进行检测，当检测到跟该 RT 地址一致的命令字后，响应数据传输。对于发送命令，RT 在发送数据之前将 RT 状态字通过 1553B 总线发送给 BC；对于接收命令，RT 在接收完数据后将 RT 状态字通过 1553B 总线发送给 BC。BC 通过 RT 状态字判断本次数据传输（发送/接收）是否有效。

当接受到模式命令后，RT 需要对接受到的模式命令进行响应。

2.3 总线监视器（BM）

BM 能够实时侦听总线上的数据流，可以将所有的数据流记录下来，也可以有选择地进行数据监听。并设有命令存储区半满、全满标志和数据存储区半满、全满标志。

3. 寄存器描述

3.1 寄存器地址分配表

表 3-1　寄存器地址分配

地址（HEX）	读/写	有效位宽	默认值(HEX)	寄存器描述
0x00	RD/WR	16	0000	中断屏蔽寄存器(IMR)
0x01	RD/WR	16	0000	配置寄存器 1(CFG1)
0x02	RD/WR	16	0000	配置寄存器 2(CFG2)
0x03	WR	16	0000	启动/复位寄存器(SRR)
	RD	16	0000	BC/RT/BM 命令堆栈指针寄存器(STACK_ADDR)
0x04	RD/WR	16	0000	BM 初始命令堆栈指针寄存器
0x05	RD	16	0000	时间标签寄存器 0(TTR0)
0x06	RD	16	0000	中断状态寄存器(INT_STA)
0x07	RD/WR	16	0000	配置寄存器 3 (CFG3)
0x08	RD/WR	16	0000	配置寄存器 4(CFG4)
0x09	RD/WR	16	0000	配置寄存器 5(CFG5)
0x0A	RD	16	0000	BM 数据堆栈指针寄存器（BM_STACK_ADDR）
0x0B	RD	16	0000	保留
0x0C	RD	16	0000	保留
0x0D	RD	16	0000	RT 上一命令字寄存器(LAST_CMD)
	WR	16	0000	BC 帧时间寄存器
0x0E	RD	16	0000	RT 状态字寄存器（RT_STA）
0x0F	RD	16	0000	RT BIT 字寄存器(RT_BIT_REG)

3.2 中断屏蔽寄存器（IMR）

地址：0x00

表 3-2 中断屏蔽寄存器（IMR）

位（BIT）	描述 (DESCRIPTION)
15	保留
14	保留
13	BC/RT 传输器超时（BC/RT TRANSMITTER TIMEOUT）
12	BC/RT 命令堆栈溢出/ BM 命令堆栈半满溢出
11	BM 命令堆栈溢出（BM COMMAND STACK ROLLOVER）
10	BM 数据堆栈溢出（BM DATA STACK ROLLOVER）
9	保留
8	BC 重发/BM 数据半满溢出
7	RT 地址奇偶校验错误（RT ADDRESS PARITY ERROR）
6	时间标签寄存器溢出（TIME TAG ROLLOVER）
5	RT 循环缓存溢出（RT CIRCULAR BUFFER ROLLOVER）
4	BC 消息/RT 子地址控制字消息结束（BC MSG/RT SUBADDRESS CONTROL WORD EOM）
3	BC 帧结束（BC END OF FRAME）
2	格式错误(FORMAT ERROR)
1	BC 状态置位/RT 方式代码（BC STATUS SET/RT MODE CODE）
0	消息结束（END OF MESSAGE）

中断屏蔽寄存器中位如置 1 表示打开对应中断状态寄存器的中断，如置 0 表示关闭对应的中断状态寄存器的中断。

- **BIT13**： 芯片内置看门狗，当使能（置 1）该位则传输编码时间超过 668us 时产生中断。
- **BIT12**： 芯片命令堆栈大小为 256 字，当使能（置 1）该位则在 BC/RT 类型下命令堆栈指针超出 256 则产生中断；当在 BM 类型下则在命令堆栈半满时产生中断。
- **BIT11**：当使能（置 1）该位则 BM 命令堆栈溢出时产生中断。
- **BIT10**：当使能（置 1）该位则 BM 数据堆栈溢出时产生中断。

- **BIT8**：当使能（置 1）该位则在 BC 类型下 BC 重发消息前将产生中断；如果在 BM 类型下数据堆栈半满溢出将产生中断。
- **BIT7**： 在 RT 模式下当使能（置 1）该位那么如果 RTAD4-RTAD0 与 RTADP 共六位进行奇偶的结果是 0 就产生中断。
- **BIT6**： 当使能（置 1）该位那么当时标寄存器计时到 65535 个单位（由最小计时精度确定）时产生中断。
- **BIT5**： 当使能（置 1）该位那么 RT 循环缓存溢出产生中断。
- **BIT4**： 当使能（置 1）该位那么当作 BC 总线控制器时 BC 控制字中的 BIT4 位为 1 则消息结束产生中断;当作 RT 时当 RT 的子地址控制器的 BIT14 或 BIT9 或 BIT4 中的任意一位为 1 则产生中断。
- **BIT3**： 当使能（置 1）该位那么 BC 帧发送结束产生中断。
- **BIT2**： 当使能（置 1）该位那么格式错误时产生中断。
- **BIT1**： 当使能（置 1）该位那么当作 BC 时，BC 收到的状态字中有置 1 的位，则产生中断；当作 RT 时子地址查找表中方式字对应位为 1 则产生中断。
- **BIT0**： 当使能（置 1）该位那么当 BC/RT 发送/接收消息结束产生中断。

3.3 配置寄存器 1(CFG1- BC)

地址：0x01

表 3-3 配置寄存器 1(BC-CFG1)

位（BIT）	描述 (DESCRIPTION)
15(MSB)	BC/RT/BM 模式设置（RT/BM*-BC*）,(logic 0)
14	BC/RT/BM 模式设置（BM/RT*- BC*）,(logic 0)
13	A*/B 区域设置（CURRENT AREA A*/B）
12	保留
11	保留
10	保留
9	保留
8	帧自动重复发送使能（FRAME AUTO-REPEAT）

位（BIT）	描述 (DESCRIPTION)
7	保留
6	保留
5	消息间隔时间使能（MESSAGE GAP TIMER ENABLED）
4	消息重发使能（RETRY ENABLED）
3	消息重发一次或二次选择（DOUBLE/SINGLE* RETRY）
2	BC 使能（BC ENABLED），*该位只读*
1	BC 帧信息忙指示（BC FRAME IN PROGRESS），*该位只读*
0 (LSB)	BC 消息忙指示（BC MESSAGE IN PROGRESS），*该位只读*

BC 配置寄存器 1 主要用作 1553B 总线控制器工作模式的选择，还有是否使能消息重发、帧重复发送功能，以及 1553B 总线控制器工作状态的指示等。

- **BIT15,BIT14**：此两位组合为 00 设置为 BC 模式，10 设置为 RT 模式,01 设置为 BM 模式，上电默认为 BC 模式。
- **BIT13**：该位为 0 则使用 RAM 的 A 区域，该位为 1 则使用 RAM 的 B 区域。
- **BIT8**：该位为 0 则 BC 发送完一帧数据就停止，若该位为 1 则帧重复发送直到启动/复位寄存器(SSR)中的 BIT0(RESET)、BIT5(STOP_ON_FRAME)，BIT6(STOP_ON_MESSAGE)中任意位为 1 才会停止。
- **BIT5**：该位为 0 则消息之间间隔时间固定为近似 8 到 11us,这位为 1 则消息之间的间隔时间通过 BC 命令堆栈的第三个字指定，其指定范围为最小约 8us 到最大约 65535us, 时间精度为 1us。当为 10Mbps 传输速度时则相应为以前的 0.2 倍。
- **BIT4**：该位为 0，BC 对所有的消息都不重发，该位为 1 且 BC 控制字的 BIT8 也为 1 那么该消息在返回状态字出错，响应时间超时则重发消息。
- **BIT3**：在配置寄存器 1（CFG1）的 BIT4 为 1 的条件下，该位为 0 则该消息在返回状态字出错，响应时间超时则重发一次; 该位为 1 则该消息在返回状态字出错，响应时间超时则重发消息两次。
- **BIT2**：该位为只读位，含义同 BIT1。
- **BIT1**：该位为只读位，在帧的第一个消息启动后到帧的最后一个消息结束一直被设为 1，在帧自动重复发送模式下则一直保持为 1 直到帧重复发送结束。

- **BIT0**： 该位在 BC 总线控制器每个消息开始传输时置为 1，在消息结束传输时清为 0。

3.4 配置寄存器 1 (CFG1-RT)

地址：0x01

表 3-4 配置寄存器 1(CFG1-RT)

位（BIT）	描述 (DESCRIPTION)
15(MSB)	BC/RT/BM 模式设置（RT/BM*-BC*）,(logic 1)
14	BC/RT/BM 模式设置（BM/RT*- BC*）,(logic 0)
13	保留
12	保留
11	动态总线控制接收*（DYNAMIC BUS CONTROL ACCEPTANCE*）
10	忙*（BUSY*）
9	服务请求*（SERVICE REQUEST*）
8	子系统标志*（SUBSYSTEM FLAG*）
7	RT 标志*（RTFLAG*）
6	保留
5	保留
4	保留
3	保留
2	保留
1	保留
0	BC 消息忙指示（BC MESSAGE IN PROGRESS），*该位只读*

- **BIT15,BIT14**：此两位组合为 00 设置为 BC 模式，10 设置为 RT 模式,01 设置为 BM 模式, 上电默认为 BC 模式。
- **BIT11**： 该位为 0 则 RT 状态字寄存器的 BIT1 位为 1。
- **BIT10**： 该位为 0 则 RT 状态字寄存器的 BIT3 位为 1。

- **BIT9**： 该位为 0 则 RT 状态字寄存器的 BIT8 位为 1。
- **BIT8**： 该位为 0 则 RT 状态字寄存器的 BIT2 位为 1。
- **BIT7**： 该位为 0 则 RT 状态字寄存器的 BIT0 位为 1。
- **BIT0**： 该位在 BC 总线控制器每个消息开始传输时置为 1，在消息结束传输时清为 0。

3.5 配置寄存器 1 (CFG1-BM)

地址：0x01

表 3-5 配置寄存器 1(CFG1-BM)

位（BIT）	描述 (DESCRIPTION)
15(MSB)	BC/RT/BM 模式设置（RT/BM*-BC*）,(logic 0)
14	BC/RT/BM 模式设置（BM/RT*- BC*）,(logic1)
13	A*/B 区域设置（CURRENT AREA A*/B）
12	保留
11	保留
10	保留
9	保留
8	保留
7	保留
6	保留
5	保留
4	保留
3	保留
2	保留
1	保留
0	BM 忙指示（BM Busy），*该位只读*

- **BIT15,BIT14**：此两位组合为 00 设置为 BC 模式，10 设置为 RT 模式,01 设置为

BM 模式, 上电默认为 BC 模式。

- **BIT13：**该位为 0 则使用 RAM 的 A 区域，该位为 1 则使用 RAM 的 B 区域。
- BIT0： 该位在 BM 接收每个消息时置为 1，在消息结束时清为 0。

3.6 配置寄存器 2 (CFG2)

地址：0x02

表 3-6 配置寄存器 2(CFG2)

位（BIT）	描述 （DESCRIPTION）
15	保留
14	保留
13	忙查找表使能(BUSY LOOK UP TABLE ENABLE)
12-10	保留
9-7	时间标签最小精度设置（TIME TAG RESOLUTION2，1，0）
6	同步清除时标寄存器使能（CLEAR TIME TAG ON SYNCHRONIZE）
5	同步重载时标寄存器使能（LOAD TIME TAG ON SYNCHRONIZE）
4	中断状态自动清除（INTERRUPT STATUS AUTO CLEAR）
3	电平/脉冲中断(LEVEL/PULSE *INTERRUPT REQUEST)
2	清除服务请求(CLEAR SERVICE REQUEST)
1-0	保留（其中 BIT1 可进行读写，但没有实际意义）

- BIT13： 该位为 1 则使能 RT 的忙位查找表。
- BIT9，BIT8，BIT7：000 则最小精度为 64us，001 则最小精度为 32us，010 则最小精度为 16us，011 则最小精度为 8us，100 则最小精度为 4us，101 则最小精度为 2us，110 则最小精度为 1us，111 则最小精度为 128us。当传输速率为 10Mbps 时则最小精度缩小 5 倍，也就是 000 则最小精度为 12.8us，001 则最小精度为 6.4us 依此类推。
- BIT6： 该位为 1 则当 RT 收到同步方式字(方式代码为 00001)时 RT 的时间标签寄存器清 0。

- **BIT5**： 该位为 1 则当 RT 收到同步方式字(方式代码为 10001)时 RT 的时间标签寄存器重新导入方式代码带的数据值。
- **BIT4**： 该位为 1 则 CPU 读出中断状态寄存器的值后，中断状态字寄存器自动清 0。
- **BIT3**： 该位为 0 产生脉冲中断信号，为 1 则产生电平中断信号，在该芯片中建议用电平中断。
- **BIT2**： 该位为 1 则当 RT 收到方式字(方式代码为 10000)时，将自动将服务请求撤消。也就是将 RT 配置寄存器 1 的 BIT9 置 1， RT 状态寄存器的 BIT8 置 0。

3.7 启动/复位寄存器（SRR)

地址：0x03

表 3-7 启动/复位寄存器（SRR）

位（BIT）	描述 (DESCRIPTION)
15-7	保留
6	BC 停止消息发送（BC STOP-ON-MESSAGE）
5	BC 停止帧发送（BC STOP-ON-FRAME）
4	保留
3	时间标签寄存器清零（TIME TAG RESET）
2	中断状态寄存器清零（INTERRUPT RESET）
1	BC/BM 启动（BC/BM START）
0	系统软复位（RESET）

启动 / 复位寄存器(SSR)用作“命令”类型的功能，能实现软复位，BC 启动，中断状态寄存器复位，时间标签寄存器(TTR)复位，在帧自动重复发送时还可以停止帧的自动复重发送。

- **BIT6**： 置 1 则在一个正在发送的消息发送完毕后即停止 BC 工作，如果没有消息在处理则立即停止 BC 工作。
- **BIT5**： 置 1 则在一个正在发送的帧发送完毕后即停止 BC 工作，如果没有帧在处理则立即停止 BC 工作。
- **BIT3**： 置 1 清时间标签寄存器为 0。
- **BIT2**： 置 1 除了中断状态寄存器 的 BIT 7 位（RT 地址奇偶位错）不被清除其余位均被清除到 0。

- **BIT1**： 置 1 时在 BC 模式下启动帧传输;在 BM 模式下启动 BM 监视。。
- **BIT0**： 置 1 则进行软复位，在 BC/RT 模式时立即停止正在进行的处理。所有的寄存器和内部状态都被复位到上电时的初始态。

3.8 BC/RT/BM 命令堆栈指针寄存器（STACK_ADDR)

地址：0x03

表 3-8 BC/RT 命令堆栈指针寄存器(STACK_ADDR)

位（BIT）	描述 （DESCRIPTION）
15-0	BC/RT/BM 命令堆栈指针

BC/RT/BM 命令堆栈寄存器主要寄存 BC/RT/BM 命令堆栈指针, 当作 BC 时将消息数据读出后该指针递增 4；当作 RT 时 RT 接到新的消息时该指针递增 4； 当作 BM 时 BM 接到新的消息时该指针递增 4。

3.9 BM 初始命令堆栈指针寄存器（INIT_STACK_ADDR)

地址：0x04

表 3-9 BM 初始命令堆栈指针寄存器（INIT_STACK_ADDR)

位（BIT）	描述 (DESCRIPTION)
15-0	BM 命令堆栈指针初始位置

BM 初始命令堆栈指针寄存器主要用于设置最初的命令堆栈指针,默认为 0X0000H 即从 RAM 的第一个单元开始保存接收到的数据。

3.10 时间标签寄存器 0（TTR0）

地址：0x05

表 3-10 时间标签寄存器 0（TTR）

位（BIT）	描述 （DESCRIPTION）
15-0	时间计时标签位

时间标签寄存器 0 用于寄存 OBT1553 计时结果的 BIT15-BIT0 位。

3.11 中断状态寄存器（INT_STA）

地址：0x06

表 3-11 中断状态寄存器（INT_STA）

位（BIT）	描述 (DESCRIPTION)
15	中断请求（MASTER INTERRUPT）
14	保留
13	BC/RT 传输器超时（BC/RT TRANSMITTER TIMEOUT）
12	BC/RT 命令堆栈溢出/BM 命令堆栈半满溢出
11	BM 命令堆栈溢出（BM COMMAND STACK ROLLOVER）
10	BM 数据堆栈溢出（BM DATA STACK ROLLOVER）
9	保留
8	BC 重发（BC RETRY）/BM 数据半满溢出
7	RT 地址奇偶校验错误（RT ADDRESS PARITY ERROR）
6	时间标签寄存器溢出（TIME TAG ROLLOVER）
5	RT 循环缓存溢出（RT CIRCULAR BUFFER ROLLOVER）
4	BC 消息/RT 子地址控制字消息结束（BC MSG/RT SUBADDRESS CONTROL WORD EOM）
3	BC 帧结束（BC END OF FRAME）
1	BC 状态置位/RT 方式代码（BC STATUS SET/RT MODE CODE）
0	消息结束（END OF MESSAGE）

- **BIT15**： BIT14-BIT0 中的任意一位为 1 则该位为 1。
- **BIT13**： 芯片内置看门狗，当传输编码时间超过 668us 时该位置 1。
- **BIT12**： 芯片命令堆栈大小为 256 字，当在 BC/RT 类型下命令堆栈指针超出 256 时该位置 1；当在 BM 类型下则在命令堆栈半满时该位置 1。
- **BIT11**： BM 命令堆栈溢出时则该位为 1。
- **BIT10**： BM 数据堆栈溢出时则该位为 1。
- **BIT8**： 在 BC 类型下 BC 重发消息前将时该位置 1；如果在 BM 类型下数据堆栈半满溢出时该位置 1。
- **BIT7**： 在 RT 模式下那么如果 RTAD4-RTAD0 与 RTADP 共六位进行奇偶的结果是 0 就时该位置 1。

- **BIT6**： 当时标寄存器计时到 65535 个单位（由最小计时精度确定）时该位置 1。
- **BIT5**： RT 循环缓存溢出时该位置 1。
- **BIT4**：当作 BC 总线控制器时 BC 控制字中的 BIT4 位为 1 则消息结束时该位置 1；当作 RT 时当 RT 的子地址控制器的 BIT14 或 BIT9 或 BIT4 中的任意一位为 1 时该位置 1。
- **BIT3**： BC 帧发送结束时该位置 1。
- **BIT2**： 格式错误时该位置 1，格式错误是指响应超时、奇偶校验错、编码错、计数错等。
- **BIT1**： 当作 BC 时，BC 收到的状态字中有置 1 的位，则产生中断；当作 RT 时子地址查找表中方式字对应位为 1 时该位置 1。
- **BIT0**： 当 BC/RT 发送/接收消息结束时该位置 1。

3.12 配置寄存器 3（CFG3）

地址：0x07

表 3-12 配置寄存器 3(CFG3)

位（BIT）	描述 (DESCRIPTION)		
15-13	保留		
12-11	BM 命令堆栈大小设置位 1,0	BIT12,BIT11	全满消息条数
		00	20
		01	40
		10	80
		11	160
10-8	BM 数据堆栈大小设置位 2-0	BIT10,BIT9,BIT8	全满字个数
		000	3328
		001	1664
		010	832
		011	416
		100	208
		101	624

位（BIT）	描述 (DESCRIPTION)		
		110	1248
		111	2496
7	非法命令查找表使能（ILLEGALIZATION DISABLED）		
4	接收非法命令屏蔽（ILLEGAL RX TRANSFER DISABLE）		
3	接收忙屏蔽(BUSY RX TRANSFER ENABLE)		
2-1	保留		
0	增强方式代码功能(ENHANCED MODE CODE HANDLING)		

- **BIT12,BIT11：** 这两位为 BM 命令堆栈大小设置位，“00”时表示收到 20 条消息时全满溢出，半满溢出则是收到 10 条消息；“01”时表示收到 40 条消息时全满溢出，半满溢出则是收到 20 条消息；“10”时表示收到 80 条消息时全满溢出，半满溢出则是收到 40 条消息；“11”时表示收到 160 条消息时全满溢出，半满溢出则是收到 80 条消息。
- **BIT10-BIT8：**这三位为 BM 数据堆栈大小设置位，当全满溢出时接收到的数据字个数如表中所示，半满溢出则少一半。如“000”时表示收到 3328 个数据字时全满溢出，半满溢出则是收到 1664 个数据字。
- **BIT7：** 该位为 1，使能 RT　RAM 中的非法命令查找表。
- **BIT4：** 该位为 0，则在接收到非法命令时将接收到的数据写入 RAM 中，否则在接收到非法命令时不将接收到的数据写入 RAM 中。
- **BIT3：** 该位为 0，则在忙时将接收到的数据写入 RAM 中，否则在忙时不将接收到的数据写入 RAM 中。
- **BIT0：** 该位为 1，使能 RT　RAM 中的方式代码查找表。

3.13 配置寄存器 4(CFG4)

地址：0x08

表 3-13　配置寄存器 4(CFG4)

位（BIT）	描述 (DESCRIPTION)
15-9	保留
8	第一次重发通道选择（FIRST RETRY ALT/SAME* BUS）

位（BIT）	描述 (DESCRIPTION)
7	第二次重发通道选择（SECOND RETRY ALT/SAME* BUS）
6-4	保留
3	RT 地址配置使能(LATCH RT ADDRESS WITH CFG REG #5)
2-0	保留

- **BIT8**： 置 0 则在最初发送的消息失败后第一次重发的消息与最初发送的消息在同一通道上传输。置 1 则在最初发送的消息失败后第一次重发的消息不再最初发送的消息的通道上传输。
- **BIT7**： 置 0 则在第一次重发的消息失败后第二次重发的消息与第一次重发的消息在同一通道上传输。置 1 则在第一次重发的消息失败后第二次重发的消息不再第一次重发的消息的通道上传输。此位只有在配置寄存器 1（CFG1）的 BIT3 位为 1 才有效。
- **BIT3**： 当该位为 1 时配置寄存器 5 的 BIT5-BIT0 才可写。

3.14 配置寄存器 5 (CFG5)

地址：0x09

表 3-14 配置寄存器 5(CFG5)

位（BIT）	描述 (DESCRIPTION)
15-11	保留
10-9	超时响应时间设置(RESPONSE TIMEOUT SELECT1,0)
8-6	保留（其中 BIT6 可进行读写，但没有实际意义）
5-1	RT 地址位 4-0(RT ADDRESS4-ADDRESS0)
0	RT 地址奇偶位(RT ADDRESS PARITY)

- **BIT10.BIT9**： 超时响应时间设置,如为 00 是 19us, 01 是 23us, 10 是 51us, 11 是 130us。当为 10Mbps 的传输速度时则均为 1M 的 0.2 倍，如 00 是 3.8us。
- **BIT5-BIT1**： 配置 RT 地址。
- **BIT0**： 配置 RT 校验位，该位与 BIT5-BIT1 的异或结果要为 1。

3.15 BM 数据堆栈指针寄存器（BM_STACK_ADDR)

地址：0x0A

表 3-15 BM 数据堆栈指针寄存器（BM_STACK_ADDR)

位（BIT）	描述 (DESCRIPTION)
15-0	BM 数据堆栈指针

BM 数据堆栈寄存器主要寄存 BM 数据堆栈指针, BM 接到新的数据时该指针递增 1。

3.16 BC 帧时间/RT 上一命令字寄存器(LAST_CMD)

地址：0x0D

表 3-16 BC 帧时间/RT 上一命令字寄存器（LAST_CMD)

位（BIT）	描述 (DESCRIPTION)
15-0	BC 帧时间/RT 上一命令字寄存器

该寄存器用作 BC 帧时间设置寄存器时为可写，RT 上一命令字时为可读。

3.17 RT 状态字寄存器(RT_STA)

地址：0x0E

表 3-17 RT 状态字寄存器(RT_STA)

位（BIT）	描述 (DESCRIPTION)
15-11	均为 0
10	消息错误(MESSAGE ERROR)
9	测试手段(INSTRUMENTATION)
8	服务请求(SERVICE REQUEST)
7-5	保留
4	广播指令接收(BROADCAST COMMAND RECEIVED)
3	忙(BUSY)
2	子系统标志(SYBSYSTEM FLAG)
1	动态总线控制接收(DYNAMIC BUS CONTROL ACCEPT)
0	终端标志(TERMINAL FLAG)

- **BIT10**： 如该位为 1 则表明有消息错误。消息错误是指响应超时、奇偶校验错、

编码错、计数错，非法命令等。

- **BIT9**： 该位在 OBT1553B IP 中一直为 0。
- **BIT8**： 如该位为 1 则表明 RT 有服务请求。
- **BIT4**： 如该位为 1 则表明通讯方式是广播通讯方式。
- **BIT3**： 如该位为 1 则表明系统正忙。
- **BIT2**： 子系统标志位。
- **BIT1**： 动态总线控制接收标志位。
- **BIT0**： 终端标志位。

3.18 RT BIT 字寄存器(RT_BIT_REG)

地址：0x0F

表 3-18 RT BIT 字寄存器(RT_BIT_REG)

位（BIT）	描述 (DESCRIPTION)
15	传输器超时（TRANSMITTER TIMEOUT）
14-12	保留
11	通道 B 发送器关闭（TRANSMITTER SHUTDOWN B）
10	通道 A 发送器关闭（TRANSMITTER SHUTDOWN A）
9	终端标志禁止(TERMINAL FLAG INHIBITED)
8	总线传输通道 B/A*(CHANNEL B/A*)
7	保留
6	字计数错（WORD COUNT ERROR）
5	错误数据同步头（INCORRECT SYNC RECEIVED）
4	奇偶/位计数错(PARITY/BIT COUNT ERROR)
3	RT-RT 同步头/地址错(RT-RT SYNC/ADDRESS ERROR)
2	RT-RT 响应超时（RT-RT NO RESPONSE ERROR)
1	RT-RT 第二个命令字错(RT-RT 2ND COMMAND WORD ERROR)
0	命令字内容错(COMMAND WORD CONTENTS ERROR)

- **BIT15**： 传输器超时该位置 1。
- **BIT11**： 通道 B 发送器关闭该位置 1。

- **BIT10**： 通道 A 发送器关闭该位置 1。
- **BIT9**： 终端标志禁止该位置 1。
- **BIT8**： 消息传输在 A 通道进行为 0，消息传输在 B 通道进行为 1。
- **BIT6**： 字计数错该位置 1。
- **BIT5**： 错误数据同步头该位置 1。
- **BIT4**： 奇偶/位计数错该位置 1。
- **BIT3**： RT-RT 同步头/地址错则该位置 1。
- **BIT2**： RT-RT 响应超时则该位置 1。
- **BIT1**： RT-RT 第二个命令字错如奇偶错则该位置 1。
- **BIT0**： RT-RT 第二个命令字内容出错如地址错则该位置 1。

说明：从表 3-19 开始后面的表不是寄存器，它们占用 RAM 空间。

3.19 BC 控制字（BC_CTRL）

表 3-19 BC 控制字（BC_CTRL）

位（BIT）	描述 (DESCRIPTION)
15	保留
14	消息格式错误屏蔽（M.E.MASK）
13	服务请求位屏蔽(SERVICE REQUEST BIT MASK)
12	忙位屏蔽(SUBSYS BUSY BIT MASK)
11	子系统标志位屏蔽(SUBSYS FLAG BIT MASK)
10	终端标志位屏蔽(TERMINAL FLAG BIT MASK)
9	保留
8	重试使能(RETRY ENABLED)
7	总线通道选择 A/B*(bus channel a/b*)
6	保留
5	保留
4	EOM 中断使能(EOM INTERRUPT ENABLE)

位（BIT）	描述 (DESCRIPTION)
3	保留
2	模式命令(MODE CODE FORMAT)
1	广播命令(BROADCAST FORMAT)
0	RT2RT(RT-TO-RT FORMAT)

- **BIT14-BIT10**： 这 5 位均是为 1 则屏蔽相应的 RT 状态字位，如果 RT 状态字位某位被屏蔽则该位不影响中断状态寄存器的（INT_STA）的 BIT1 位，但影响 BC 块状态字的 BIT7 位。
- **BIT8**： 置 1 且配置寄存器 1（CFG1）的 bit4 为 1 则在响应超时和格式错误时消息重发。
- **BIT7**： 置 1 消息传输通过 A 通道，置 0 消息传输通过 B 通道。
- **BIT4**： 该位置 1 且中断屏蔽寄存器（IMR）的 BIT4 也置 1 那么则消息结束中断状态寄存器(INT_STA)的 BIT4 为 1。
- **BIT2.BIT1.BIT0**：置为 000 且命令字的 T/R*位是 0 则是 BC-> RT 通讯方式，置为 000 且命令字的 T/R*位是 1 则是 RT->BC 的通讯方式； 置为 001 是 RT-RT 通讯方式；置为 010 是 Broadcast 通讯方式；置为 100 是 Mode Code 通讯方式； 置为 110 是 Broadcast Mode Code 通讯方式。 其它两种组合没用到。

3.20 BC 命令字（BC_CMD）

表 3-20 BC 命令字(BC_CMD)

位（BIT）	描述 (DESCRIPTION)
15-11	远程终端地址（REMOTE TERMINAL ADDRESS）
10	发送/接收* (T/R*)
9-5	子地址/方式(SUBADDRESS/MODE)
4-0	数据字计数/方式代码(DATA WORD COUNT/MODE CODE)

3.21 BC 块状态字(BC_BLK)

表 3-21 BC 块状态字(BC_BLK)

位（BIT）	描述 (DESCRIPTION)

位（BIT）	描述 (DESCRIPTION)
15	消息传输结束标志位（EOM）
14	保留
13	传输通道指示(CHANNEL B/A*)
12	出错标志(ERROR FLAG)
11	状态设置（STATUS SET）
10	格式错误(FORMAT ERROR)
9	响应超时(NO RESPONSE TIMEOUT)
8	保留
7	屏蔽状态设置（MASKED STATUS SET）
6-5	重发消息次数(RETRY COUNT 1,0)
4	数据传输正常（GOOD DATA BLOCK TRANSFER）
3	错误的状态地址（WRONG STATUS ADDRESS）
2	字计数错误（WORD COUNT ERROR）
1	同步头出错(INCORRECT SYNC TYPE)
0	无效字（INVALID WORD）

该字主要用来说明消息发送/接收后的状态。该存储器字如果有重发消息，只表明的是最后一个消息的状态。

- **BIT15**： 消息转输结束该位置 1。
- **BIT13**： 消息传输在 A 通道进行为 0，消息传输在 B 通道进行为 1。
- **BIT12**： 有格式错误或响应超时产生该位为 1。
- **BIT11**： 接受的 RT 返回字中 BIT10-BIT0 中只要有一位为 1 则该位为 1。
- **BIT10**： 格式错误如计数个数、同步头、奇偶等错误则该位为 1。
- **BIT9**： 响应超时则该位置 1。
- **BIT6,5**： 记录 BC 重发消息的个数，00 表示没有重发，01 表示重发了一次，11 表示重发了两次。
- **BIT4**： 当 RT2BC,RT2RT,或者传输带数据方式代码，没有消息格式错则为 1，有消息格式错为 0；当 BC2RT，带数据方式代码收和不带数据方式代码，为 0。

- **BIT3**： RT 返回的状态字中 RT 地址有错则该位为 1。
- **BIT2**： 当 RT2BC,RT2RT,或者传输带数据方式代码，字计数有错则为 1，没错为 0；当 BC2RT，带数据方式代码收和不带数据方式代码，为 0。
- **BIT1**： 同步头出错该位置 1。
- **BIT0**： 当数据传输中有奇偶错、位计数错和同步头错则该位置 1。

3.22 RT 子地址控制字(RT_SUB_CTRL)

表 3-22 RT 子地址控制字(RT_SUB_CTRL)

位（BIT）	描述 (DESCRIPTION)
15	RX: 接收双缓存使能（DOUBLE BUFFER ENABLE）
14	TX: 发送消息结束中断使能（EOM INT）
13	TX: 发送循环缓存溢出使能(CIRC BUF INT)
12-10	TX: 发送存储器管理设置（MEMORY MANAGEMENT2,1,0）
9	RX: 接收消息结束中断使能（EOM INT）
8	RX: 接收循环缓存溢出使能(CIRC BUF INT)
7-5	RX: 接收存储器管理设置（MEMORY MANAGEMENT2,1,0）
4	BCST: 广播消息结束中断使能（EOM INT）
3	BCST: 广播循环缓存溢出使能(CIRC BUF INT)
2-0	BCST: 广播存储器管理设置（MEMORY MANAGEMENT2,1,0）

- **BIT15**： 该位为 1 则选择 RT 为双缓冲存储器管理方式，为 0 则为单缓冲或循环缓冲存储器管理方式。
- **BIT14**： 置 1 发送消息结束中断使能。
- **BIT13**： 置 1 发送循环缓存溢出使能。
- **BIT12-BIT10**：发送存储器管理设置,000 是单消息模式，001 是在循环缓存中大小为 128 字，010 是 256 字依此类推。
- **BIT9**： 置 1 接收消息结束中断使能。
- **BIT8**： 置 1 接收循环缓存溢出使能。
- **BIT7-BIT5**： 接收存储器管理设置，000 是单消息模式，001 是在循环缓存中大小为 128 字，010 是 256 字依此类推。

- **BIT4**： 置 1 广播消息结束中断使能。
- **BIT3**： 置 1 广播循环缓存溢出使能。
- **BIT2-BIT0**： 广播存储器管理设置，000 是单消息模式，001 是在循环缓存中大小为 128 字，010 是 256 字依此类推。

3.23 RT/BM 块状态字(RT/BM_BLK)

表 3-23 RT/BM 块状态字(RT/BM_BLK)

位（BIT）	描述 (DESCRIPTION)
15	消息传输结束标志（EOM）
14	消息传输开始标志（SOM）
13	传输通道指示（CHANNEL B/A*）
12	出错标志(ERROR FLAG)
11	RT-RT 通讯方式标志(RT-RT FORMAT)
10	格式错误标志(FORMAT ERROR)
9	响应超时标志(NO RESPONSE TIMEOUT)
8	保留
7	循环缓存溢出（DATA STACK ROLLOVER）
6-5	非法命令字(ILLEGAL COMMAND WORD)
4	字计数个数错(WORD COUNT ERROR)
3	数据同步头出错(INCORRECT DATA SYNC)
2	无效字（INVALID WORD）
1	RT-RT 同步头/地址错（RT-RT SYNCH/ADDRESS ERROR）
0	RT-RT 第二个命令字错（RT-RT 2ND COMMAND ERROR）

- **BIT15**： 消息传输结束该位置 1。
- **BIT14**： 消息传输开始该位置 1。
- **BIT13**： 消息传输在 A 通道进行为 0，消息传输在 B 通道进行为 1。
- **BIT12**： 有格式错误或响应超时产生该位为 1。
- **BIT11**： 在 RT-RT 通讯时接收 RT 时被设置为 1， 发送 RT 时与该位无关。
- **BIT10**： 格式错误如计数个数，同步头，奇偶等错误则该位为 1。

- **BIT9**：RT-RT 通讯时响应超时则该位置 1。
- **BIT7**：循环缓存溢出则该位置 1。
- **BIT6**：非法命令字则该位置 1。
- **BIT5**：RT 接收的字个数出错时该位置 1。
- **BIT4**：数据同步头出错出错时该位置 1。
- **BIT3**：当数据传输中有奇偶错、位计数错则该位置 1。
- **BIT2**：RT-RT 同步头/地址错则该位置 1。
- **BIT1**：RT-RT 第二个命令字错如奇偶错则该位置 1。
- **BIT0**：RT-RT 第二个命令字内容出错如地址错则该位置 1。

4. 功能模块描述

4.1 BC 总线控制器工作方式

4.1.1 BC 存储器地址分配

表 4-1 BC 存储器地址分配(4K 双口 RAM)

地址（HEX）	描述
0000-00FF	堆栈 A（STACK A）
0100	堆栈指针 A (STACK POINTER A)
0101	消息个数设置 A (MESSAGE COUNT A)
0102-0103	保留
0104	堆栈指针 B (STACK POINTER B)
0105	消息个数设置 B (MESSAGE COUNT B)
0106-0107	保留
0108-012D	消息块 0(MESSAGE BLOCK0)
012E-0153	消息块 1(MESSAGE BLOCK1)
…	…
0ED6-0EFB	消息块 93(MESSAGE BLOCK 93)
…	…
0F00-0FFF	堆栈 B（STACK B）

4.1.2 BC 存储器管理

BC 存储器管理如图 BC 存储器管理图所示。该图说明了命令堆栈区包含四个描述符，即块状态字，时间标签字，消息间隔时间字和消息块地址字。块状态字包括消息状态、完成、有效性及总线通道信息；时间标签字寄存了当前消息结束时时间标签寄存器的值；消息间隔时间字存储的是设定的消息间隔时间；消息块地址字寄存的是指向消息块第一个字的地址。程序通过 RAM 的 0X0100H 地址取命令堆栈指针，0X0101 地址取消息个数值。

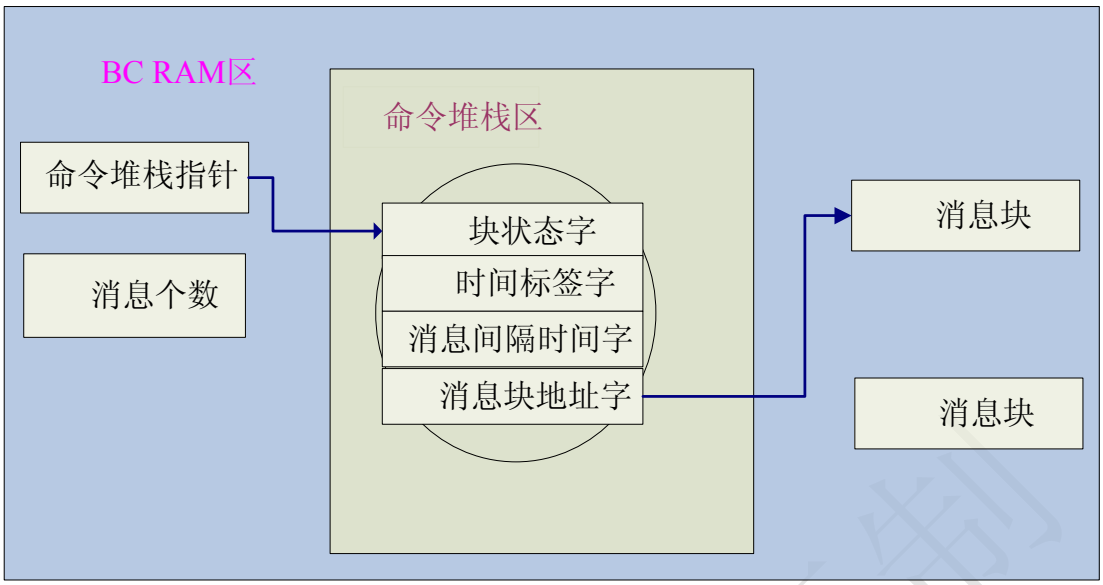

图 4-1 BC 存储器管理

4.1.3 BC 消息格式

表 4-2 BC 消息格式

BC 到 RT 的传输	RT 到 BC 的传输	RT 到 RT 的传输	不带字的方式命令
控制字	控制字	控制字	控制字
接收命令字	发送命令字	接收命令字	方式命令字
数据字 1	发送命令字的回应字	发送命令字	方式回应字
数据字 2	状态字	发送命令字的回应字	状态字
…	数据字 1	发送终端的状态字	
	数据字 2	数据字 1	
最后一个数据字	…	…	
最后数据字的回应字		最后一个数据字	
状态字	最后一个数据字	接收终端的状态字	

表 4-3 BC 消息格式（接上表）

带字的发送方式命令	带字的接收方式命令	广播命令	不带字的广播方式命令	带字的广播方式命令
控制字	控制字	控制字	控制字	控制字
发送方式命令字	接收方式命令字	广播命令字	广播方式命令字	广播方式命令字
方式命令回应字	数据字	数据字 1	广播方式命令字的回应字	数据字
接收状态字	接收命令字的回应字	数据字 2 …		广播方式命令字的回应字
数据字	接收状态字	最后的数据字		

4.2 RT 远程终端工作方式

4.2.1 RT 存储器地址分配

表 4-4 RT 存储器地址分配(4K 双口 RAM)

地址（HEX）	描述
0000-00FF	堆栈 (STACK)
0100	堆栈指针(STACK POINTER)
0101-0107	保留
0108-010F	方式代码选择中断表（MODE CODE SELECTIVE INTERRUPT TABLE）
0110-013F	方式代码数据(MODE CODE DATA)
0140-01BF	查找表(LOOKUP TABLE)
01C0-023F	保留
0240-0247	忙位查找表（BUSY BIT LOOKUP TABLE）
0248-025F	（没有使用）
0260-027F	数据块 0 (DATA BLOCK 0)

地址（HEX）	描述
0280-02FF	数据块 1-4　(DATA BLOCK 1-4)
0300-03FF	非法命令表 (CINNABD UKKEGAKUZUBG TABKE)
0400-041F	数据块 5 (DATA BLOCK 5)
0420-043F	数据块 6 (DATA BLOCK 6)
…	…
0FE0-0FFF	数据块 100 (DATA BLOCK 100)

4.2.2 RT 存储器查找表

表 4-5　RT 存储器查找表(LOOK_UP TABLE)

地址（HEX）	对应子地址	描述
0140 … 015F	Rx_SA0 … Rx _SA31	接收查找表
0160 … 017F	Tx_SA0 … Tx _SA31	发送查找表
0180 … 19F	Bcst_SA0 … Bcst _SA31	广播查找表
01A0 … 01BF	SACW_SA0 … SACW_SA31	子地址控制字查找表

4.2.3 RT 存储器非法命令表地址分配

表 4-6 RT 存储器非法命令地址分配表(COMMAND ILLEGALIZING TABLE)

位（BIT）	描述 (DESCRIPTION)
15-10	均为逻辑 0
9-8	均为逻辑 1
7	广播*/本身地址（BROADCAST*/OWN ADDRESS）
6	发送/接收*（T/R*）
5-1	子地址 4－子地址 0（SA4-SA0）
0	子计数 4/方式字 4（WC4/MC4）

RT 非法命令表，在 RT 中占用 0x300～0x3FF 的地址空间。当 RT 接收到命令字后，如果使能（为 1）非法化命令检测。通过广播/RT 地址、T/R*、SA4~SA0 和 WC4/MC4 共 8 位在 0x300～0x3FF 中，查找 WC3~WC0（MC3~MC0）收到的给 RT 某一子地址、某些个数的命令字是否非法。

4.2.4 RT 存储器忙位查找表地址分配

表 4-7 RT 存储器忙位查找表地址分配表(BUSY BIT LOOKUP TABLE)

位（BIT）	描述 (DESCRIPTION)
15-10	均为逻辑 0
9	逻辑 1
8	逻辑 0
7	逻辑 0
6	逻辑 1
5-3	均为逻辑 0
2	广播/本身地址*（BROADCAST/OWN ADDRESS*）
1	发送/接收*（T/R*）
0	子地址 4（SA4）

RT 忙位查找表，在 RT 中占用 0x240～0x247 的地址空间。当 RT 接收到命令字后，如果使能（为 1）忙位查找表检测。通过广播/RT 地址、T/R*、SA4 共 3 位在 0x240～0x247 中，查找 SA3~SA0 收到的给 RT 某一子地址的命令字是否忙。

4.2.5 RT 存储器方式代码选择中断表

表 4-8 RT 存储器方式代码选择中断表（MODE CODE SELECTIVE INTERRUPT TABLE）

位（BIT）	描述 (DESCRIPTION)
0108	接收方式命令 0-15(undefined)
0109	发送方式命令 16-31(WITH DATA)
010A	发送方式命令 0-15(WITHOUT DATA)
010B	发送方式命令 16-31(WITH DATA)
010C	广播接收方式命令 0-15(undefined)
010D	广播接收方式命令 16-31(WITH DATA)
010E	广播发送方式命令 0-15(WITHOUT DATA)
010F	广播发送方式命令 16-31(UNDEFINED)

4.2.6 RT 存储器方式代码选择中断地址分配

表 4-9 RT 存储器方式代码选择中断表地址分配表(MODE CODE SELECTIVE INTERRUPT TABLE)

位（BIT）	描述 (DESCRIPTION)
15-9	均为逻辑 0
8	逻辑 1
7	逻辑 0
6	逻辑 0
5	逻辑 0
4	逻辑 0

位（BIT）	描述 (DESCRIPTION)
3	逻辑 1
2	广播/本身地址*（BROADCAST/OWN ADDRESS*）
1	发送/接收*（T/R*）
0	方式代码 4（MC4）

RT 方式代码选择中断表，在 RT 中占用 0x108～0x10F 的地址空间。当 RT 接收到命令字后，如果使能（为 1）式代码选择中断表检测。通过广播/RT 地址、T/R*、MC4 共 3 位在 0x108～0x10F 中，查找 MC3~MC0 收到的给 RT 某一方式代码是否有中断。

4.2.7 RT 方式代码数据表

表 4-10 RT 存储器方式代码数据表（MODE CODE DATA）

地址（HEX）	方式代码（MODE CODE）
0110	没有定义
0111	同步带数据
0112-11F	没有定义
0120	发送矢量字
0121-13F	没有定义

4.2.8 芯片实现的方式代码

表 4-11 芯片实现的方式代码

发/收*	方式代码	功能	是否带数据字	是否允许广播指令
1	00000	动态总线控制	否	否
1	00001	同步	否	是
1	00010	发送上一状态字	否	否
1	00011	启动自测试[1]	否	是
1	00100	发送器关闭	否	是
1	00101	取消发送器关闭	否	是

发/收*	方式代码	功能	是否带数据字	是否允许广播指令
1	00110	禁止终端标志位	否	是
1	00111	取消禁止终端标志位	否	是
1	01000	复位远程终端[2]	否	是
1	01001	备用	否	待定
…	…	…	…	…
1	01111	备用	否	待定
1	10000	发送矢量字	是	否
0	10001	同步[3]	是	是
1	10010	发送上一指令字	是	否
1	10011	发送自检测字	是	否

4.2.9 RT 单缓冲存储器管理

RT 单缓冲存储器管理如图 RT 单缓冲存储器管理图所示。该图说明了命令堆栈区包含四个描述符，即块状态字，时间标签字，数据块指针字和接收命令字。块状态字包括消息状态、完成、有效性及总线通道信息；时间标签字寄存了当前消息结束时时间标签寄存器的值；数据块指针字存储指向数据块的起始地址；接收命令字存储 RT 接收到的命令字。程序通过 RAM 的 0X0100H 地址取命令堆栈指针。

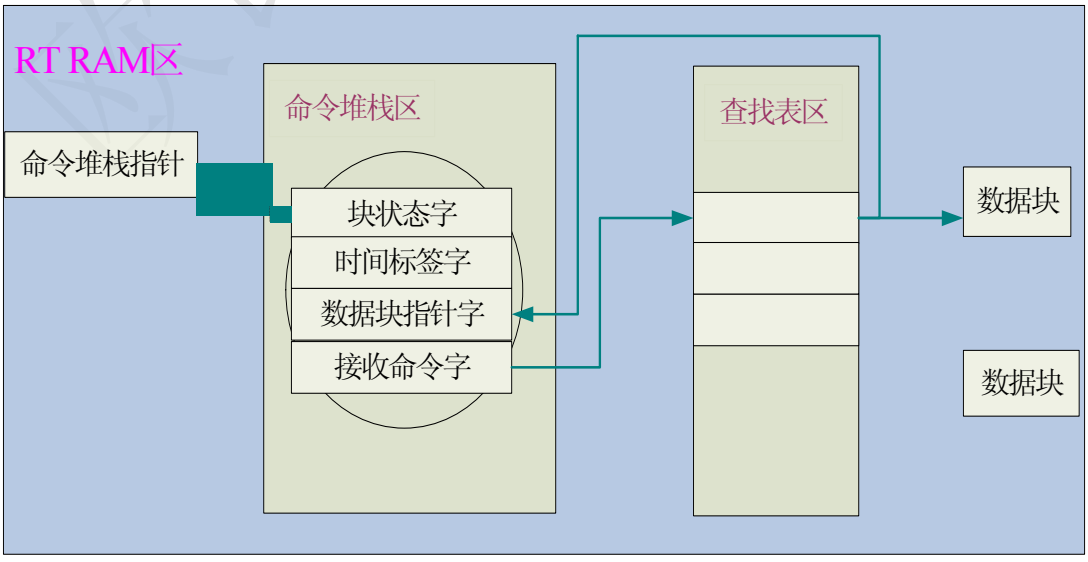

图 4-2 RT 单缓冲存储器管理

4.2.10 RT 循环缓冲存储器管理

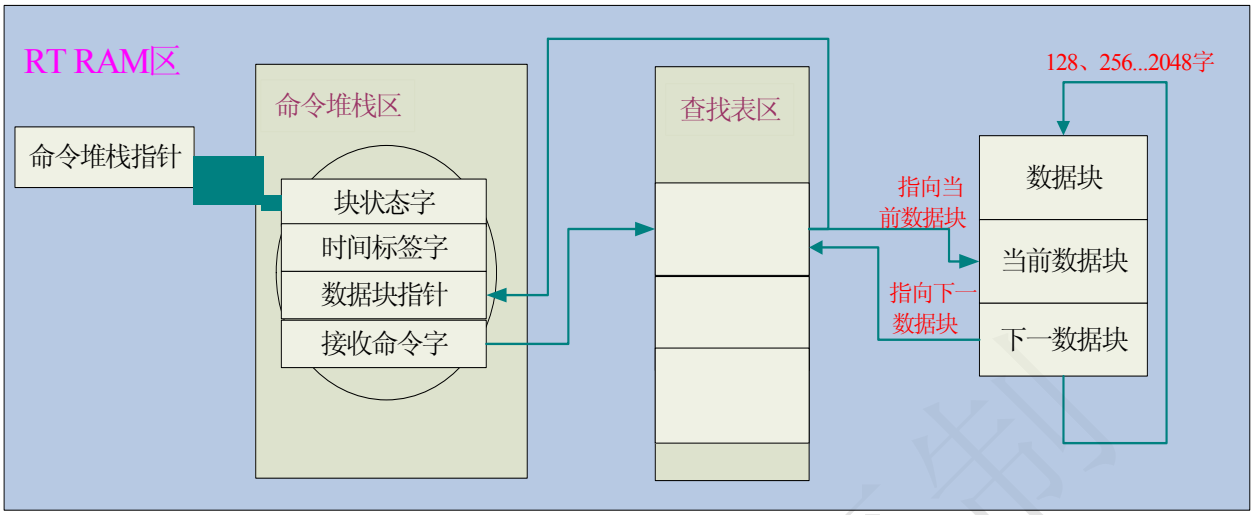

图 4-3 RT 循环缓冲存储器管理

4.2.11 RT 双缓冲存储器管理

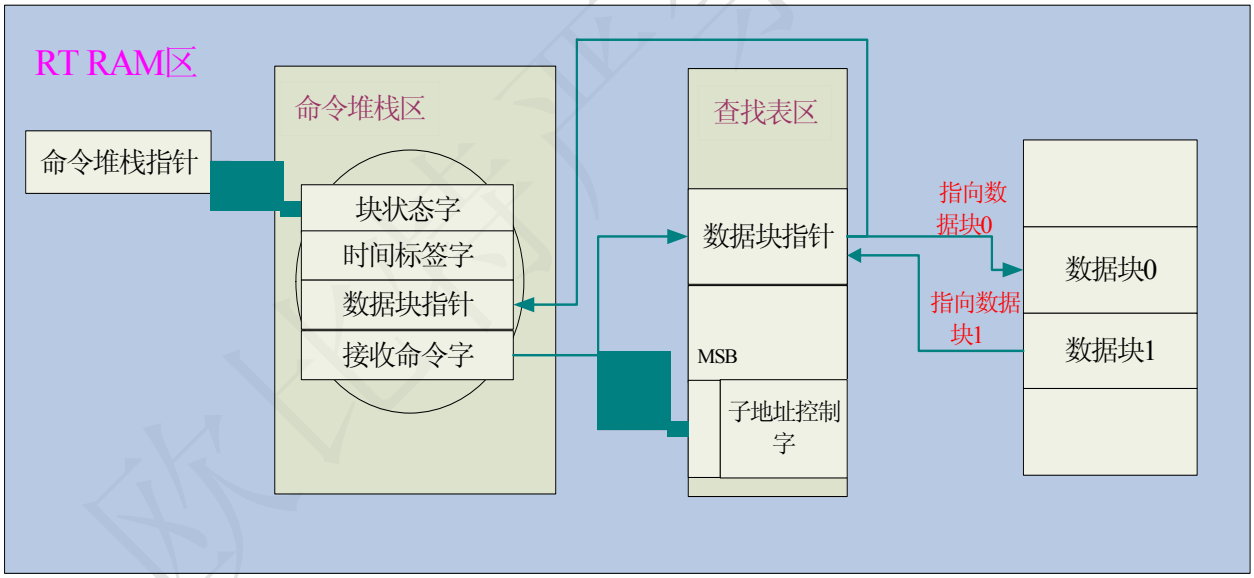

图 4-4 RT 双缓冲存储器管理

4.3 BM 总线监视器工作方式

4.3.1 BM 存储器地址分配

表 4-12 BM 存储器地址分配

地址（HEX）	描述

地址（HEX）	描述
0000-027F	命令堆栈区域 (STACK AREA)
0280-02FF	子地址选择设置区域(SUBADDRESS SELECT AREA)
0300-0FFF	数据块区域(DATA BLOCK AREA)

4.3.2 BM 存储器管理

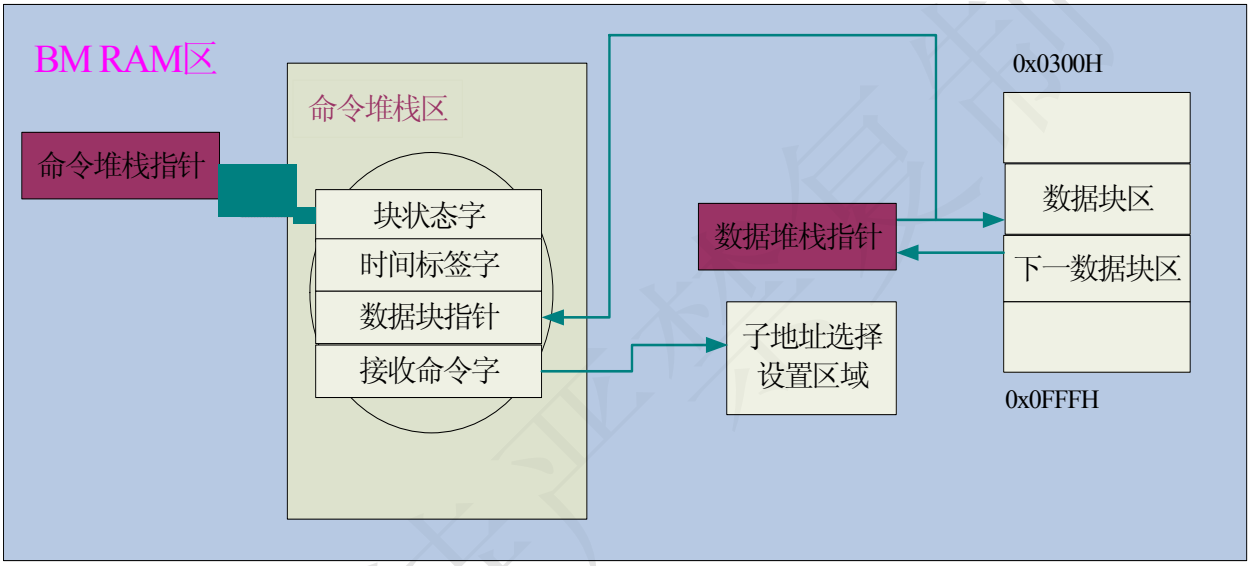

图 4-5 BM 存储器管理

4.3.3 BM 子地址选择设置区地址分配

表 4-13 BM 子地址分配

位（BIT）	描述 (DESCRIPTION)
15-11	逻辑 0
10	逻辑 0
9	逻辑 1
8	逻辑 0
7	逻辑 1
6	RT 地址 4（RT ADDRESS 4）

位（BIT）	描述 (DESCRIPTION)
5	RT 地址 3（RT　ADDRESS 3）
4	RT 地址 2（RT　ADDRESS 2）
3	RT 地址 1（RT　ADDRESS 1）
2	RT 地址 0（RT　ADDRESS 0）
1	发送/接收*（T/R*）
0	子地址 4（SA4）

BM 子地址选择设置区地址分配表，在 BM 中占用 0x280～0x2FF 的地址空间。当 BM 接收到命令字后，如果使能（为 1）子地址选择设置检测。通过 RT 地址、T/R*、SA4 共 7 位在 0x280～0x2FF 中，查找 SA3~SA0 收到的给 BM 某一子地址的内容接收，否则不做接收。

5. 时序图

OBT1553B 芯片与主机的接口支持 INTEL 模式 E 和 MOTOROLA 模式两种方式，图 5-1 所示为 INTEL 模式的存取时序图，图 5-2 所示为 MOTOROLA 模式的存取时序图，表 5-1 为时间参数要求：

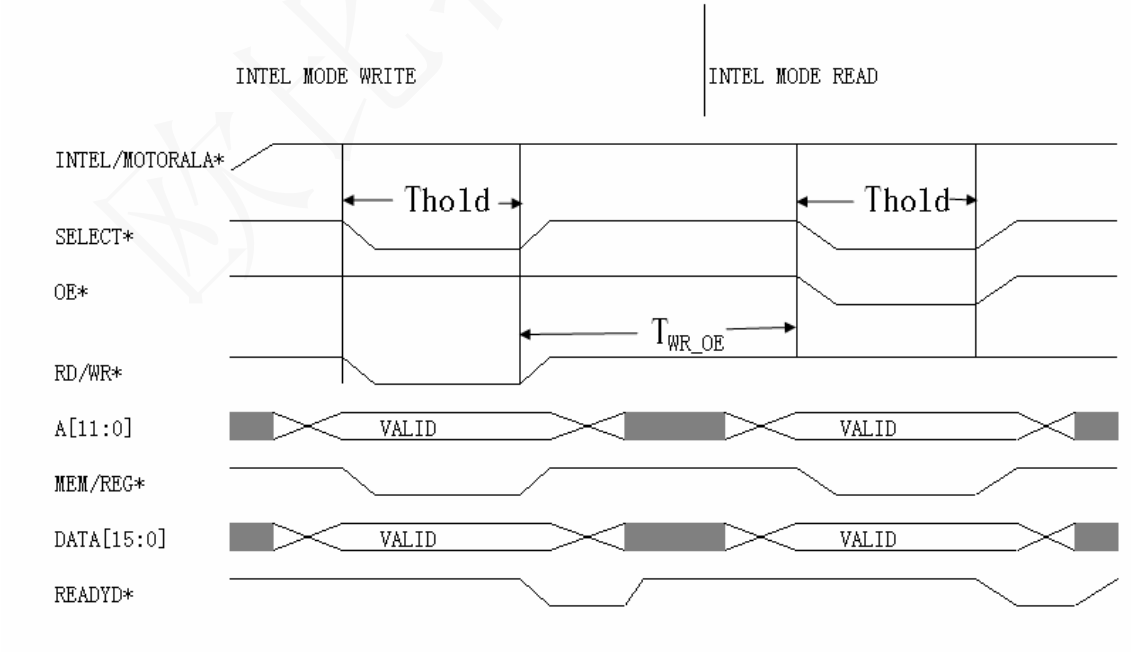

图 5-1　INTEL 模式存取时序图（以存取寄存器为例）

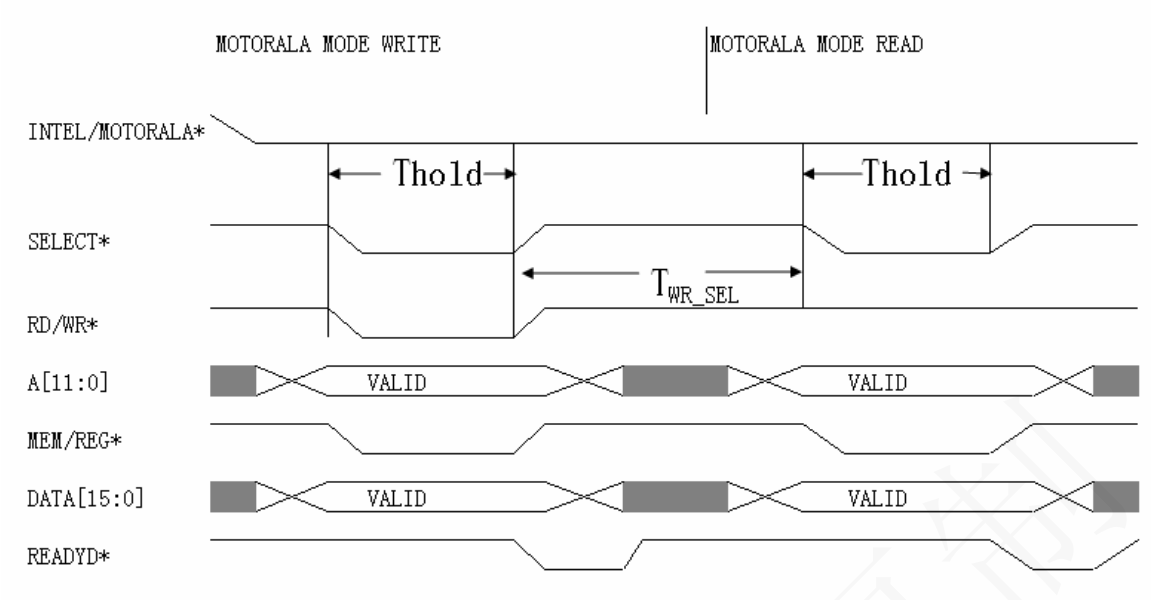

图 5-2 MOTOROLA 模式存取时序图（以存取寄存器为例）

表 5-1 时间参数

SYMBOL	PARAMETER	MIN	TYP	MAX	UNIT
Thold	READ OR WRITE OR CS PULSE WIDTH	70			NS
T_{WR_OE}	OE ON AFTER WRITE OFF	70			NS
T_{WR_SEL}	CS ON AFTER WRITE OFF	70			NS

6. 电气特性

6.1 极限电气参数

表 6-1 极限电气参数

Absolute Maximum Ratings	
Specification	**Maximum Rating**
Storage Temperature	-65 to +150 ^{O}C
Ambient Temprerature with Power Applied	-55 to +125 ^{O}C
Supply Voltage to Ground	-0.5 to +7.0V
Input Voltage(Vin)	Vss-0.5V V_{DD} +0.5V
Maximum Power Dissipation	

6.2 直流电气特性

测试条件: Vcc=5V±0.3V, Vgnd = 0v, Tamb = 25 ^{o}C ;

表 6-2 直流电气特性

Electrical Characteristics over Operating Range(VDD=5V)						
Parameter	**Description**	**Test Conditions**		**Min**	**Max**	**Units**
V_{OH}	Output High Voltage	VDD = Min V_{IN}=V_{IH} or V_{IL}	I_{OH}= -1.5 mA	3.0	4.9	V
V_{OL}	Output Low Voltage		I_{OL}=+1.5mA	-	4.9	V
V_{IH}	Input High Level	-	-	2.4		V
V_{IL}	Input Low Level	-	-		0	V
I_{LI}	Input Leakage Current	Vss ≤ V_{IN} ≤ V_{DD}, V_{DD}=MAX			80	uA
I_{OZ}	Three-State Output Leakage Current	Vss ≤ V_{IN} ≤ V_{DD}, V_{DD}=MAX			0.000001	uA
I_{CC}	Power Supply Current	V_{DD}=5.3V, CLKsample=24MHz CLK1553 = 16MHz			55	mA

7. 封装描述

7.1 PGA70 封装尺寸描述

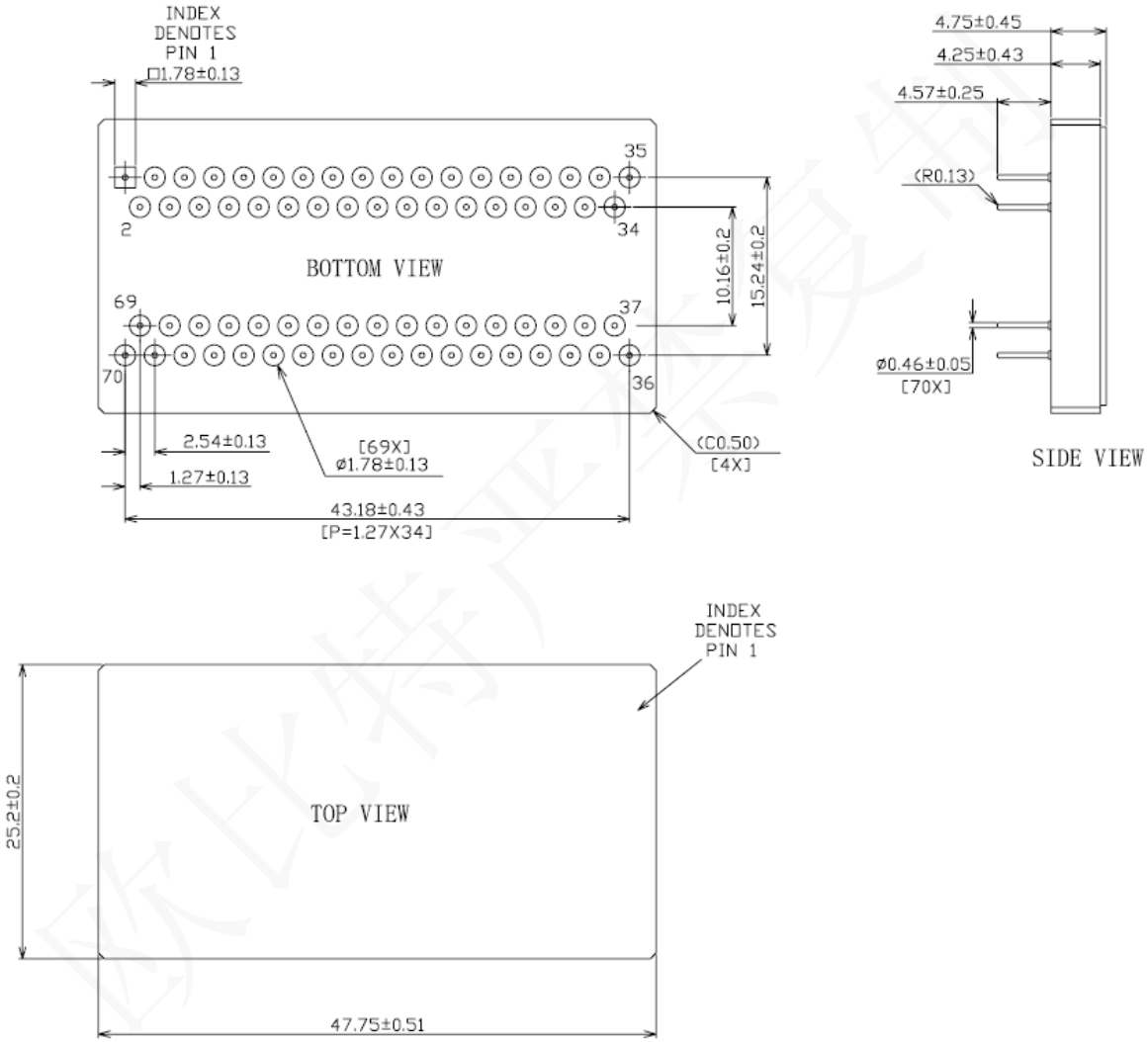

图 7-1　封装尺寸图

7.2 PGA70 引脚功能定义

表 7-1 引脚功能定义表

引脚号	信号名称	方向	有效电平	信号说明
1	TX/RX-A	I/O	-	A 通道正向模拟发送/接收信号，外部直接接 1553 变压器
2	TX/RX-A*	I/O	-	A 通道反向模拟发送/接收信号，外部直接接 1553 变压器
3	SELECT#	I	低	芯片片选信号
4	STRBD#	I	低	芯片输出使能信号，等效于 RAM 的 OEN 信号
5	MEM/REG#	I	-	访问内部 RAM 或寄存器选择信号，为低时选择寄存器
6	RD/WR#	I	-	读写信号
7	MSTCLR#	I	-	芯片复位信号
8	A15	-	-	保留引脚
9	A14	-	-	保留引脚
10	A13	-	-	保留引脚
11	A12	-	-	保留引脚
12	A11	I	-	输入地址线 A11
13	A10	I	-	输入地址线 A10
14	A9	I	-	输入地址线 A9
15	A8	I	-	输入地址线 A8
16	A7	I	-	输入地址线 A7
17	A6	I	-	输入地址线 A6
18	GND		-	电源地

引脚号	信号名称	方向	有效电平	信号说明
19	1553clk	I	-	16MHz 晶振输入
20	A5	I	-	输入地址线 A5
21	A4	I	-	输入地址线 A4
22	A3	I	-	输入地址线 A3
23	A2	I	-	输入地址线 A2
24	A1	I	-	输入地址线 A1
25	A0	I	-	输入地址线 A0
26	DTREQ#/MSB/LSB	-	-	保留引脚
27	Ssflag#/ext_trig	-	-	保留引脚
28	MEMENA_OUT#	-	-	保留引脚
29	MEMOE#/ADDR_LAT	-	-	保留引脚
30	Memwr#/zero_wait#	I	-	用作选择 MOTOROLA 模式或 INTEL 模式，为 1 是 MOTOROLA 模式
31	DTREQ#/16/8*	I	-	接+5V，只支持 16BIT 模式
32	POLARITY_SEL	I	-	接+5V，只支持 RD/WR*信号为高时是 RD，为低时是 WR
33	TRIGGER_SEL	I	-	悬空
34	TX/RX-B	I/O	-	B 通道正向模拟发送/接收信号，外部直接接 1553 变压器
35	TX/RX-B*	I/O	-	B 通道反向模拟发送/接收信号，外部直接接 1553 变压器
36	VB	I	-	悬空
37	GNDB		-	接地
38	+5VB		-	接+5V 电源
39	RTAD0	I	-	RT 终端地址输入 0

引脚号	信号名称	方向	有效电平	信号说明
40	RTAD1	I	-	RT 终端地址输入 1
41	RTAD2	I	-	RT 终端地址输入 2
42	RTAD3	I	-	RT 终端地址输入 3
43	RTAD4	I	-	RT 终端地址输入 4
44	RDADP	I	-	用作 RTAD0-RTAD4 的奇校验位输入，RDADP 同 RTAD0-RTAD4 进行异或的结果要为 1
45	INCMD	I	-	保留引脚
46	D00	I/O	-	数据线 0
47	D01	I/O	-	数据线 1
48	D02	I/O	-	数据线 2
49	D03	I/O	-	数据线 3
50	D04	I/O	-	数据线 4
51	D05	I/O	-	数据线 5
52	D06	I/O	-	数据线 6
53	D07	I/O	-	数据线 7
54	+5V		-	电源+5V 输入
55	D08	I/O	-	数据线 8
56	D09	I/O	-	数据线 9
57	D10	I/O	-	数据线 10
58	D11	I/O	-	数据线 11
59	D12	I/O	-	数据线 12
60	D13	I/O	-	数据线 13

引脚号	信号名称	方向	有效电平	信号说明
61	D14	I/O	-	数据线 14
62	D15	I/O	-	数据线 15
63	TAG_CLK	-	-	保留引脚
64	TRANSPARENT/BUFFERED*	I	-	接地，只支持 BUFFERED 模式
65	INT#	O	低	中断输出引脚
66	READYD#	O	低	与主处理器握手信号，为低表示 1553 处理数据完毕，等待主机发新的指令
67	IOEN#	-	-	保留引脚
68	+5VB	-	-	接+5V 电源
69	GNDB	-	-	接地
70	VA#	-	-	保留引脚

注： 表中保留引脚是指该脚可以接电源正、接电源地或悬空。

8. 应用案例

8.1 典型外围接口图

OBT1553B 芯片能提供很灵活的主机接口，能支持 MOTOROLA 和 INTEL 两种接口的访问时序。OBT1553B 芯片内部带有 4KByes 的 RAM，支持缓冲（BUFFER）模式。下面的接线图为应用的典型接线图,其中图 8-1 为 16BIT INTEL 模式的接线图,图 8-2 为 16BIT MOTOROLA 模式的接线图，图 8-3 为芯片与变压器的接线图。

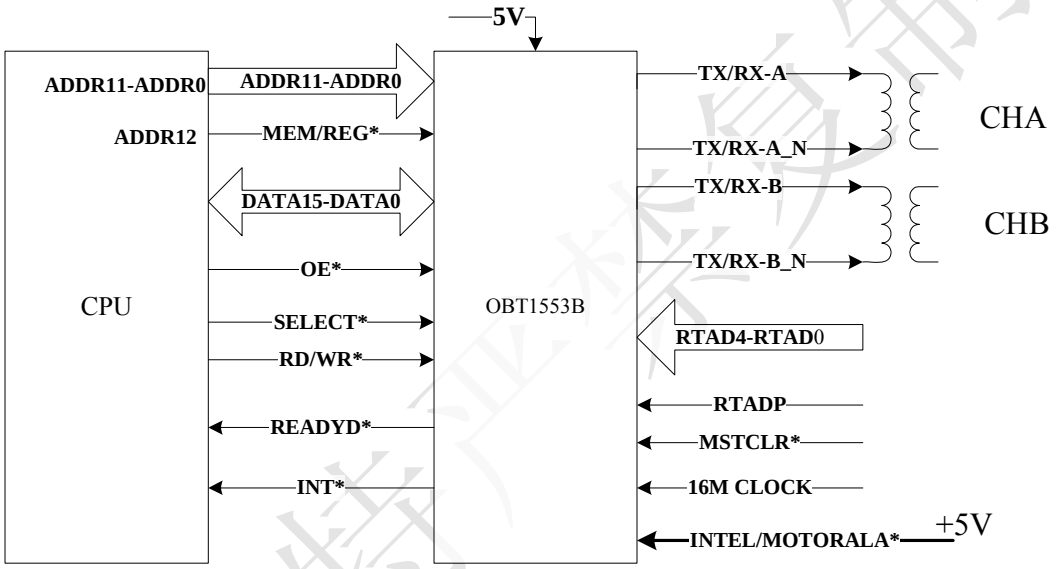

图 8-1 16BIT INTEL 模式接线图

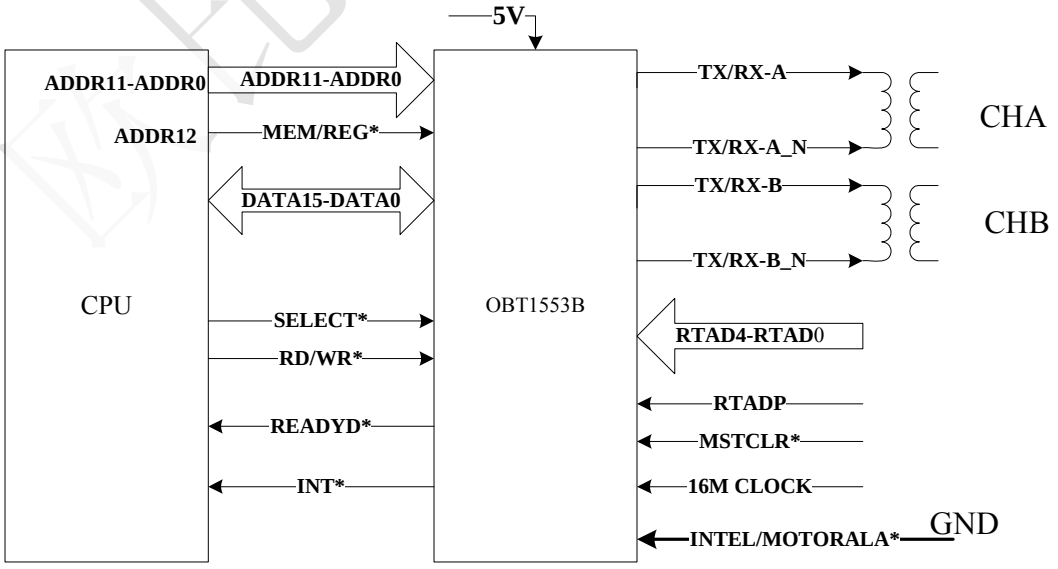

图 8-2 16BIT MOTOROLA 模式接线图

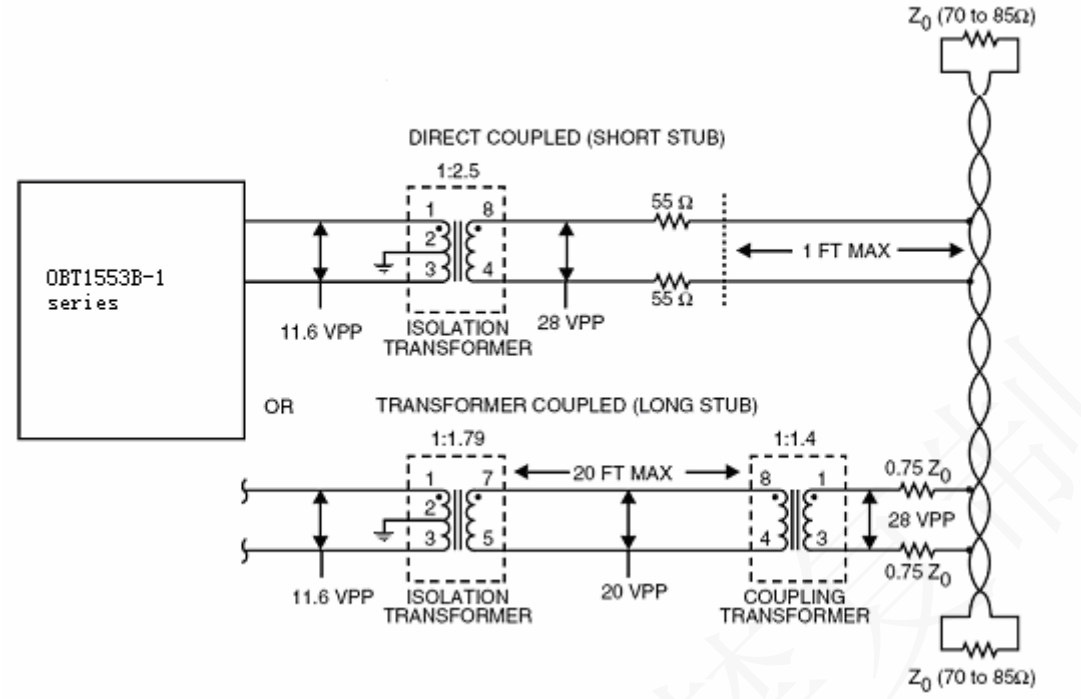

图 8-3　芯片与外部变压器接线图

8.2 BC 总线控制器应用案例

对于 BC 编程，首先要初始化相应的寄存器以及堆栈指针、消息计数器；然后定义消息的控制字、命令字等；最后启动 BC。需要注意的是 BC 控制字不会在 1553B 总线上传输。BC 的消息格式通过编程 BC 控制字的最低 3 位来控制。

BC 消息帧可以通过查询和中断来进行处理。如果采用查询模式，那么可以查询配置寄存器 1、中断状态寄存器、堆栈指针和消息计数器寄存器。另外，每一条 BC 消息结束后堆栈指针加 4。在嵌入式系统软件处理中，我们应尽量采用中断方式。

流程图如下图所示。程序把 OBT1553B 设置成 BC 总线控制器类型，并且设置消息类型为 BC-RT，RT-BC；一帧两条消息，第一条消息为 BC 往地址为 0，子地址为 5 发送 32 个数据；第二条消息为地址 0，子地址 4 往 BC 发送 32 个数据。

BC 示例程序见附录一。

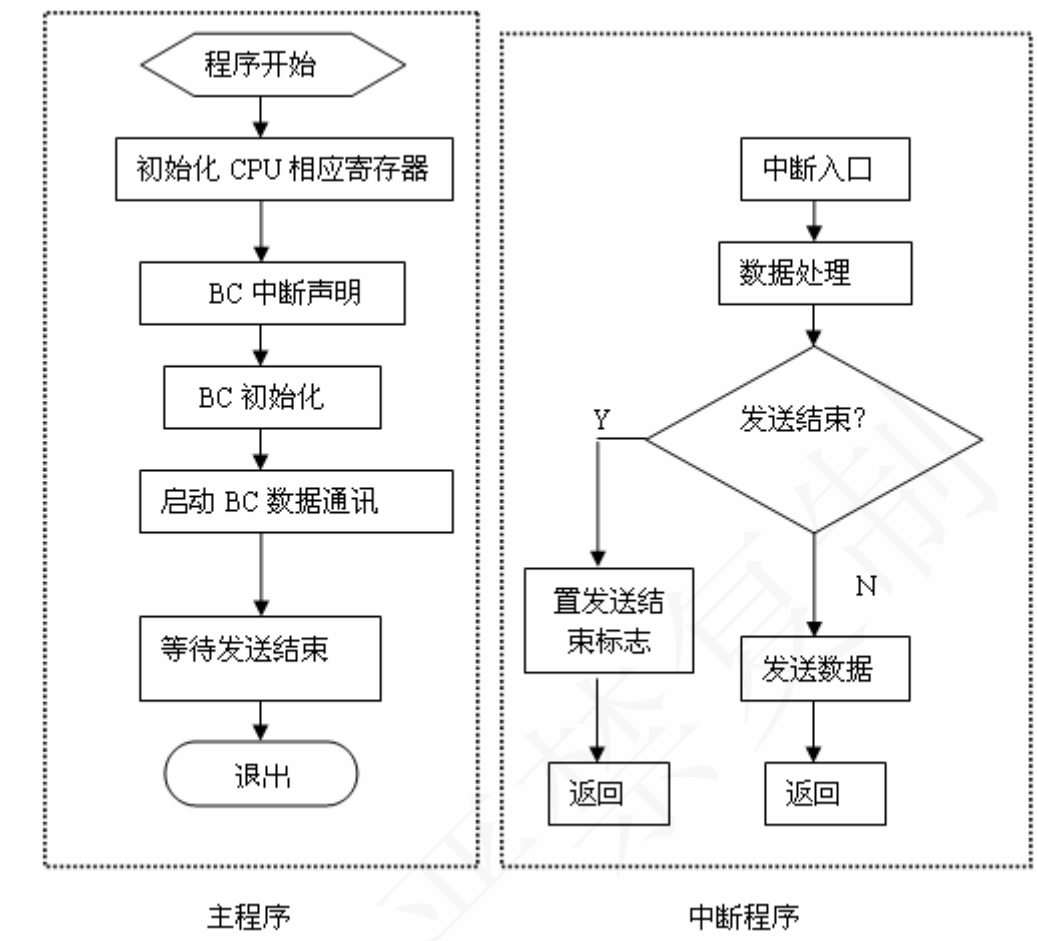

图 8-4 OBT1553B BC 程序流程图

8.3 RT 远程终端应用案例

对于 RT 编程，首先初始化相应的寄存器；然后设置非法区、初始化相应子地址的查询表及子地址控制字；最后设置配置寄存器 1 使设备处于 RT 模式。此后该设备就处于在线，只要 BC 发送一条消息命令与该设备相关，那么该设备就会做出反映。处理 RT 消息时，这里也有四个字的块描述符，即块状态字、时间标志字、数据块起始地址指针和接收到的 16 位命令字。与 BC 模式一样，要读取接收到的消息，我们应该首先从堆栈指针中读取当前消息的堆栈指针，来分别读出块状态字、时间标志字、上一条消息的块地址和命令字。

流程图如下图所示。程序把 OBT1553B 设置成 RT 远程终端类型，并且设置消息类型为 BC-RT；BC 往 RT 地址 0，子地址为 1, MC32 发送消息，消息的数据长度为 1 个，数据为互为反码的一组数----0x0000 和 0xffff。

RT 示例程序见附录二。

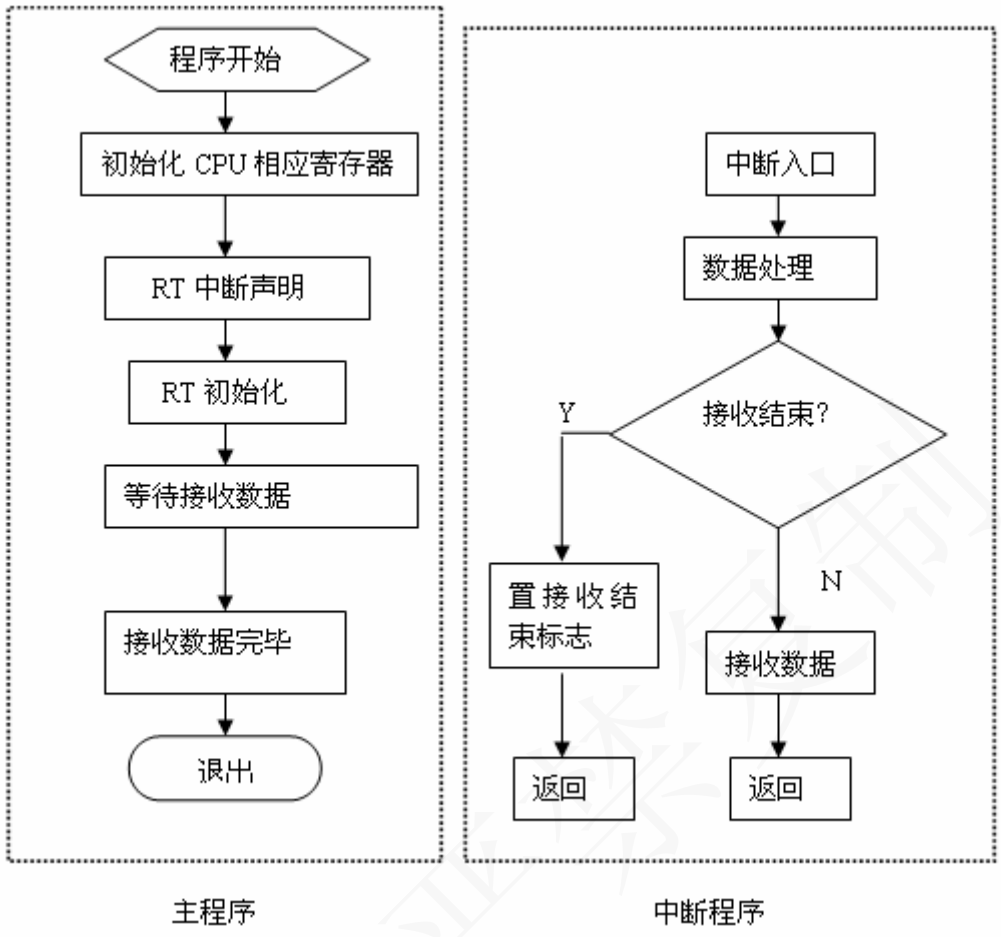

图 8-5 OBT1553B RT 程序流程图

8.4 BM 总线监视器应用案例

对于 BM 编程，首先初始化相应的寄存器；然后设置子地址选择设置区；最后设置启动/复位寄存器启动 BM。此后该设备就处于在线。接收到的数据里也有四个字的块描述符，即块状态字、时间标志字、数据块起始地址指针和接收到的 16 位命令字。与 RT 模式一样，要读取接收到的消息，我们应该首先从堆栈指针中读取当前消息的堆栈指针，来分别读出块状态字、时间标志字、上一条消息的块地址和命令字; 从数据堆栈指针中读取接收到的数据。

流程图如下图所示。程序把 OBT1553B 设置成 BM 类型，接收消息类型 BC→RT0 SA1 MC32 ,判断 BM 接收的数据是否是 BC 发送的数据。

BM 示例程序见附录三。

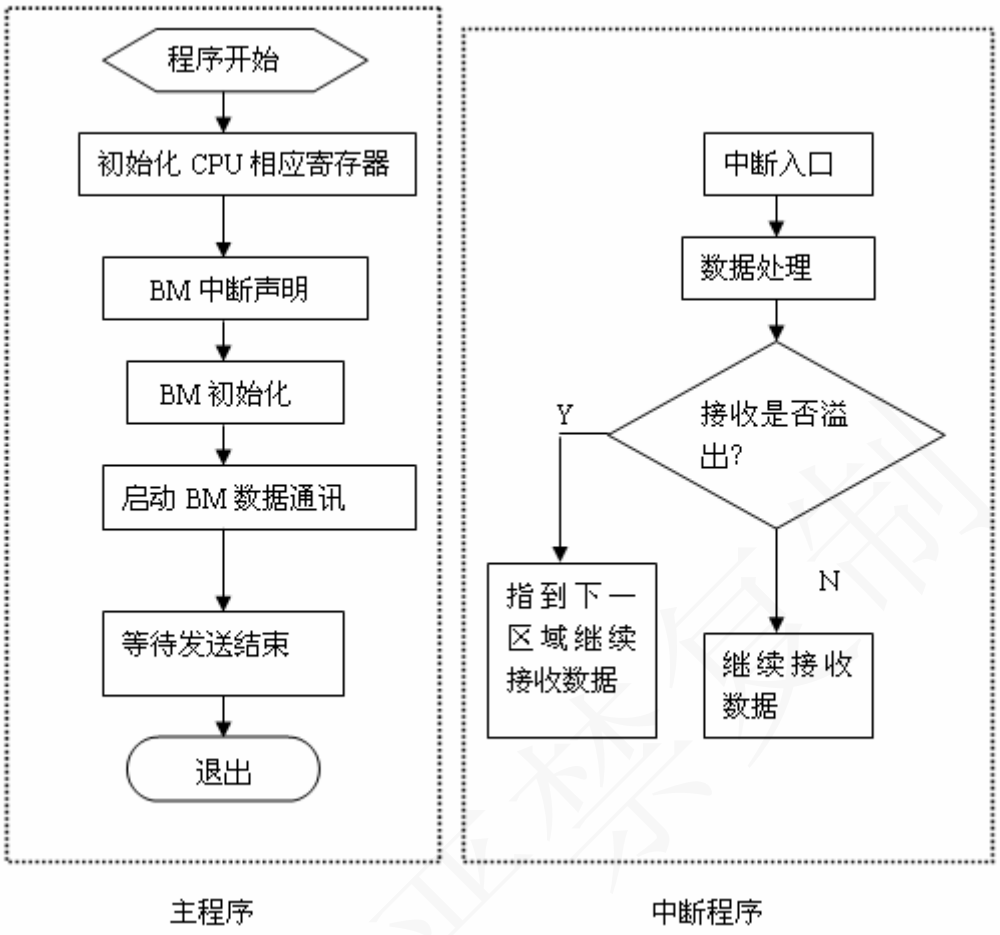

图 8-6 OBT1553B BM 程序流程图

9. 附录

9.1 附录一：BC 示例程序

Bc.h

```
#include<stdio.h>
#define IRQMP_BASE                0x80000200
#define   INTMASK       *(volatile unsigned int *)(IRQMP_BASE+0x40)   //R/W   FIRST INT MASK REGISTER
#define   INTPEND       *(volatile unsigned int *)(IRQMP_BASE+0x4)   //R   FIRST INT PEND REGISTER
#define   INTFORCE       *(volatile unsigned int *)(IRQMP_BASE+0x8)   //W   FIRST INT FORCE REGISTER
#define   INTCLEAR       *(volatile unsigned int *)(IRQMP_BASE+0xc)   //W   FIRST INT CLEAR REGISTER
#define   CTRL_RAM_BASE     0x80008000
#define   CTRL_REG_BASE     0x8000c000
#define   INT_MASK          0X0
#define   CFG_REG1          0x1
#define   CFG_REG2          0x2
#define   CFG_REG3          0x7
#define   CFG_REG4          0x8
#define   CFG_REG5          0x9
#define   START_RESET_REG   0x3
#define   MN_MN_POINTER       0x4
#define   INT_STATUS            0x6
#define   RT_STATUS             0xc
#define   RT_STATUS_RD        0xe
#define   RT_STACK_POINTER 0x3
#define control_word     0x180
#define   command_word ((0<<11)+(0<<10)+(5<<5)+0)
#define   command_word1   ((0<<11)+(1<<10)+(4<<5)+0)
#define   TR_BIT        (1<<10)
#define   RT_BUSY_BIT (1<<3)
#define   RT_ADDRESS   0xe //rt   address is 7
unsigned int rt_recv_pt,rt_send_pt;
unsigned int stack_data0[4]={0x0,0x0,0x320,0x108};
unsigned int stack_data[12]={0x0,0x0,0x320,0x108,
0x0,0x0,0x320,0x12e,
0x0,0x0,0x320,0x154,
};
unsigned int send_pt[32]=
{
0x0000,0xffff,0x0000,0xffff,
0x0000,0xffff,0x0000,0xffff,
0x0000,0xffff,0x0000,0xffff,
0x0000,0xffff,0x0000,0xffff,
```

```
0x0000,0xffff,0x0000,0xffff,
0x0000,0xffff,0x0000,0xffff,
0x0000,0xffff,0x0000,0xffff,
0x0000,0xffff,0x0000,0xffff
};
void M1553birq();
void bc_init();
void bc_send_message();
void bc_recv_message();
void delay();
unsigned int send_end_flag,send_num;
unsigned int rt_address_count,sub_address_count,data_count,data_count1;
```

MAIN.C

```
#include"bc.h"
main()
{
 printf("/************************************/");

 printf("\n       bc 控制器 BC 模式测试！");

 printf("\n       BC->RT0    SA5    MC32,RT0 SA4    MC 32");

 printf("\n/************************************/");
 INTMASK=0;
 INTCLEAR=(1<<10);
 catch_interrupt(M1553birq,10);
 delay();
 send_end_flag=1;

 bc_init();
 INTMASK=(1<<10);
 delay();
 *(volatile unsigned int *)(CTRL_REG_BASE+START_RESET_REG*4)=0x02;
 delay();
 while(send_end_flag)
 {
    M1553birq();

 }

////////////////////////////////////////////////////////////////////////////////
```

```
void M1553birq()
{
 unsigned int i,intsta,pending,message_block_address,message_block_address2,temp1[32],temp2[32],temp;
 //       printf("\n    coming ");
 pending=INTPEND;
 //              temp=0x5555;

 if(pending&1<<10)
 {

    intsta=*(volatile unsigned int *)(CTRL_REG_BASE + INT_STATUS*4);
    *(volatile unsigned int *)( CTRL_REG_BASE + START_RESET_REG*4 )=0x04;
    INTCLEAR = ( 1<<10 );
    for(i=0;i<4;i++);
    bc_send_message();
    delay();
    *(volatile unsigned int *)(CTRL_REG_BASE+START_RESET_REG*4)=0x02;
    delay();
    if(*(volatile unsigned int *)(0x80008420+34*4)!=*(volatile unsigned int *)(0x80008420+33*4))
       printf("\n last data word looped back is wrong!last data word looped back is %x actually!",*(volatile unsigned int *)(0x80008420+33*4));
    if(((*(volatile unsigned int *)(0x80008420+35*4)>>11)&0x1f)!=((*(volatile unsigned int *)(0x80008424)>>11)&0x1f))
       printf("\n status received is wrong !status received value is %x",*(volatile unsigned int *)(0x80008420+35*4));
    for(i=0;i<32;i++)
    {
       temp1[i] = *(volatile unsigned int *)(0x80008420+(i+2)*4);
    }
    if(*(volatile unsigned int*)0x80008000!=0x8000)
       printf("\n  块状态字有错，实际的块状态字是  %x ",*(volatile unsigned int*)0x80008000);
    delay();
    if(*(volatile unsigned int *)(0x800084bc)!=*(volatile unsigned int *)(0x800084c0))
       printf("\n transmit word looped back is wrong !transmit word value is %x actually!",*(volatile unsigned int *)(0x80008428));
    if(((*(volatile unsigned int *)(0x800084bc)>>11)&0x1f)!=((*(volatile unsigned int *)(0x800084c0)>>11)&0x1f))
       printf("\n status received is wrong!status received value is %x actually!",*(volatile unsigned int *)(0x80008428));
    for(i=0;i<32;i++)
    {
       temp2[i]=*(volatile unsigned int *)(0x800084b8+(i+4)*4);
    }
    for(i=0;i<32;i++)
    {
       if(temp1[i]!=temp2[i])
          printf("\n  期望收到的数是%x,实际收到的数是%x",temp1[i],temp2[i]);
    }
```

```
        if(*(volatile unsigned int*)0x80008010!=0x8010)
            printf("\n  块状态字有错，实际的块状态字是  %x ",*(volatile unsigned int*)0x80008010);
     }
    }


    void    bc_init()
    {
     unsigned int i,word_count,send_data, control_word_init,command_word_init;
      send_data=0x5555;

    //configure register

     *(volatile unsigned int *)(CTRL_REG_BASE+START_RESET_REG*4) = 0x1; //复位 1553B 寄存器
     *(volatile unsigned int *)(CTRL_REG_BASE+CFG_REG1*4) = 0x38;      //设置 BC 模式，时间标签使能，重试使能，
重试两次
     *(volatile unsigned int *)(CTRL_REG_BASE+CFG_REG2*4) = 0x388; // --time tag resolution 2 us
     *(volatile unsigned int *)(CTRL_REG_BASE+CFG_REG3*4) = 0x000; // --no override mode T/R* error
     *(volatile unsigned int *)(CTRL_REG_BASE+CFG_REG4*4) = 0x180; // --1ST retry ALT bus,2ND retry ALT bus
     *(volatile unsigned int *)(CTRL_REG_BASE+CFG_REG5*4) = 0x000; //response timeout select=00
     *(volatile unsigned int *)(CTRL_REG_BASE+INT_MASK*4) = 0x08;      //EOF 消息中断使能

     delay();

     for(i=0;i<8;i++)
     {
        *(volatile unsigned int *)(CTRL_RAM_BASE + i*4) = stack_data[i];//配置 RAM 堆栈
     }

     *(volatile unsigned int *)(CTRL_RAM_BASE + (0x100 + 0) *4)=0x0;//program the stack pointer

     *(volatile unsigned int *)(CTRL_RAM_BASE + (0x101 + 0) *4)=0xfffd;   //发送消息长度是 0xffff 的反码


     *(volatile unsigned int *)(CTRL_RAM_BASE+ 0x108*4) = control_word;
     *(volatile unsigned int *)(CTRL_RAM_BASE+ 0x109*4) = command_word;
     for(i=0;i<4;i++);

    //transmit data

     for(i=2;i<34;i++)
     {
        *(volatile unsigned int *)(CTRL_RAM_BASE+( 0x108+i)*4)=send_data++;
```

```
 }
 delay();
 *(volatile unsigned int *)(CTRL_RAM_BASE+ 0x12e*4) = control_word;
 *(volatile unsigned int *)(CTRL_RAM_BASE+ 0x12f*4) = command_word1;
 for(i=0;i<4;i++);
}

void    bc_send_message()
{

 unsigned int i,word_count,send_data, control_word_init,command_word_init;
 unsigned int message_block_address,block_state,send_data2,message_block_address1;
 // delay();
 message_block_address = *(volatile unsigned int *)((CTRL_RAM_BASE + (4*0 + 3)*4))*4 + CTRL_RAM_BASE;
 // temp1=message_block_address;
 // printf("\n is %x",(volatile unsigned int *)message_block_address);
 delay();
 *(volatile unsigned int *)(message_block_address)=control_word;
 *(volatile unsigned int *)(message_block_address+4)=command_word;
 for(i=2;i<34;i++)
 {
    *(volatile unsigned int *)(message_block_address+i*4)=send_pt[i-2];
 }
 for(i=0;i<4;i++);
 message_block_address1 = *(volatile unsigned int *)((CTRL_RAM_BASE + (4*1 + 3)*4))*4 + CTRL_RAM_BASE;
 *(volatile unsigned int *)(message_block_address1)=control_word;
 *(volatile unsigned int *)(message_block_address1+4)=command_word1;
 for(i=0;i<4;i++);
}

void delay()
{
 unsigned int i;
 for(i=0;i<0xfff;i++);
```

9.2 附录二：RT 示例程序

RT.H

```
#include<stdio.h>


#define IRQMP_BASE                0x80000200
#define    INTMASK       *(volatile unsigned int *)(IRQMP_BASE+0x40)    //R/W    FIRST INT MASK REGISTER
```

```
#define   INTPEND       *(volatile unsigned int *)(IRQMP_BASE+0x4)   //R   FIRST INT PEND REGISTER
#define   INTFORCE      *(volatile unsigned int *)(IRQMP_BASE+0x8)   //W   FIRST INT FORCE REGISTER
#define   INTCLEAR      *(volatile unsigned int *)(IRQMP_BASE+0xc)   //W   FIRST INT CLEAR REGISTER
#define   CTRL_RAM_BASE     0x80008000
#define   CTRL_REG_BASE     0x8000c000
#define   INT_MASK          0X0
#define   CFG_REG1          0x1
#define   CFG_REG2          0x2
#define   CFG_REG3          0x7
#define   CFG_REG4          0x8
#define   CFG_REG5          0x9
#define   START_RESET_REG   0x3
#define   MN_MN_POINTER      0x4
#define   INT_STATUS           0x6
#define   RT_STATUS            0xc
#define   RT_STATUS_RD        0xe
#define   RT_STACK_POINTER 0x3
#define RT_ADDRESS 0xe   //地址 7

unsigned int add_data[] = {
0x100,0x141,0x142,0x143,0x144,0x145,0x146,0x147,0x148,0x149,
0x14a,0x14b,0x14c,0x14d,0x14e,0x14f,0x150,0x151,0x152,0x153,
0x154,0x155,0x156,0x157,0x158,0x159,0x15a,0x15b,0x15c,0x15d,
0x15e,0x161,0x162,0x163,0x164,0x165,0x166,0x167,0x168,0x169,
0x16a,0x16b,0x16c,0x16d,0x16e,0x16f,0x170,0x171,0x172,0x173,
0x174,0x175,0x176,0x177,0x178,0x179,0x17a,0x17b,0x17c,0x17d,
0x17e,0x187
};
unsigned int dat_data[] = {
0x000,0x260,0x260,0x260,0x260,0x260,0x260,0x260,0x260,0x260,
0x260,0x260,0x260,0x260,0x260,0x260,0x260,0x260,0x260,0x260,
0x260,0x260,0x260,0x260,0x260,0x260,0x260,0x260,0x260,0x260,
0x260,0x260,0x260,0x260,0x260,0x260,0x260,0x260,0x260,0x260,
0x260,0x260,0x260,0x260,0x260,0x260,0x260,0x260,0x260,0x260,
0x260,0x260
};
unsigned int send_pt[32]=
{
0x0000,0xffff,0x0000,0xffff,
0x0000,0xffff,0x0000,0xffff,
0x0000,0xffff,0x0000,0xffff,
0x0000,0xffff,0x0000,0xffff,
0x0000,0xffff,0x0000,0xffff,
0x0000,0xffff,0x0000,0xffff,
```

```
0x0000,0xffff,0x0000,0xffff,
0x0000,0xffff,0x0000,0xffff
};
void rt_init();
void M1553birq();
void rt_recv_message();
void rt_send_message();
void delay();

unsigned int rt_recv_num,rt_address,rt_recv,rt_right_recv,rt_error_recv,recv_end_flag,command_word,word_count,num_count;
unsigned int send_end_flag,recv_end_flag,rt_address_count,sub_address_count,stack_offset,stack_offset1,count,temp1[32],count;
```

MAIN.C

```
#include"rt.h"
main()
{
printf("/************************************/");
printf("\n       BC 控制器 RT 模式测试！ ");
printf("\n    BC-->RT0    SA1 MC32 RT0 SA1 MC32-->BC");
printf("\n/************************************/");
INTMASK=0;
INTCLEAR=(1<<10);
catch_interrupt(M1553birq,10);
delay();
send_end_flag=1;
rt_recv_num=0;
rt_address_count=0;
sub_address_count=1;
num_count=1;
count=0;
INTMASK=(1<<10);
rt_init();
delay();
while(send_end_flag);
void M1553birq()
 {
unsigned int i=0,intsta,pending,message_block_address,message_block_address2,send_dat=0x5555,ww,temp2[32];
pending=INTPEND;
 if(pending&1<<10)
{
intsta=*(volatile unsigned int *)(CTRL_REG_BASE+INT_STATUS*4);
*(volatile unsigned int *)(CTRL_REG_BASE+START_RESET_REG*4)=0x04;
```

```
INTCLEAR=(1<<10);
if(intsta&0x01)
{
rt_recv_num++;

if(rt_recv_num<123008)
{
rt_recv_message();
delay();//
for(i=0;i<count;i++)
{
if(temp1[i]!=*(volatile unsigned int *)(CTRL_RAM_BASE+(0x260+i)*4))
{
 printf("\n 期望收到的数是%x, 实际收到的数是%x",temp1[i],*(volatile unsigned int *)(CTRL_RAM_BASE+(0x260+i)*4));
}
 }
 if(*(volatile unsigned int *)0x80008000 !=0x8000)
{
 printf("\n RT 模式块状态字有错");
printf("\n 实际的块状态字是%x",*(volatile unsigned int *)0x80008000);
                }
}
}
 }
void  rt_init()
{
  unsigned int i,send_data;
  send_data=0x5555;
      //configure register
     *(volatile unsigned int *)(  CTRL_REG_BASE+START_RESET_REG*4)=0x1;    //复位 1553B 寄存器
      *(volatile unsigned int *)(  CTRL_REG_BASE+CFG_REG1*4)=0x8780;    //设置 RT 模式，时间标签使能，重试使能，重试两次
      *(volatile unsigned int *)(  CTRL_REG_BASE+CFG_REG2*4)  = 0x388; // --time tag resolution 2 us
      *(volatile unsigned int *)(CTRL_REG_BASE+  CFG_REG3*4)  = 0x000; // --no override mode T/R* error
      *(volatile unsigned int *)(CTRL_REG_BASE+  CFG_REG4*4)  = 0x0188; // --1ST retry ALT bus,2ND retry ALT bus
      *(volatile unsigned int *)(CTRL_REG_BASE+CFG_REG5*4)  = 0x640|0x0; //set RT ADDRESS 0,TIME OVERRIDE ANSWER 130us
      *(volatile unsigned int *)(  CTRL_REG_BASE+INT_MASK*4)=0x0001;    //EOM 消息中断使能
      delay();
      printf("\n 配置堆栈");
          for(i=0;i<62;i++)
          {
```

```
                    *(volatile unsigned int *)(CTRL_RAM_BASE + add_data[i]*4) = dat_data[i];//配置 RT 工作模式
                }
                for (i=0;i<32;i++)      //fill the receive buffer
 {
        *(volatile unsigned int *)(CTRL_RAM_BASE + (0x260 + i)*4)= send_data++;
 }
   printf("\n  初始化结束！ ");
 }
void    rt_recv_message()
{
 unsigned int i,k,block_state,data_address,temp,tr_bit,send_data;
 //get block    stack    message
   delay();
   stack_offset1=*(volatile unsigned int *)(CTRL_REG_BASE+RT_STACK_POINTER*4)-4;
   data_address=*(volatile unsigned int *)(CTRL_RAM_BASE+(stack_offset1+2)*4);
   command_word=*(volatile unsigned int *)(CTRL_RAM_BASE+(stack_offset1+3)*4);
   send_data=0x6666;
   count=0;
   word_count=command_word&0x1f;
   tr_bit=command_word>>10;
   sub_address_count=(command_word>>5)&0x1f;
   rt_address_count=(command_word>>11)&0x1f;
   if(word_count==0)
        word_count=32;
        if(word_count==32)
        num_count=32;
}
void delay()
{
 unsigned int i;
 for(i=0;i<0xfff;i++);
}
```

9.3 附录三：BM 示例程序

BM.H

```
#include<stdio.h>

#define   INTMASK      *(volatile unsigned int *)(0x80000090)   //R/W   FIRST INT MASK REGISTER
#define   INTPEND      *(volatile unsigned int *)(0x80000094)   //R   FIRST INT PEND REGISTER
#define   INTFORCE     *(volatile unsigned int *)(0x80000098)   //W   FIRST INT FORCE REGISTER
#define   INTCLEAR     *(volatile unsigned int *)(0x8000009C)   //W   FIRST INT CLEAR REGISTER
```

```
#define    CTRL_RAM_BASE      0X80054000
#define    CTRL_REG_BASE      0x80050000



#define    INT_MASK           0X0
#define    CFG_REG1           0x1
#define    CFG_REG2           0x2
#define    CFG_REG3           0x7
#define    CFG_REG4           0x8
#define    CFG_REG5           0x9
#define    START_RESET_REG    0x3
#define    MN_MN_POINTER      0x4
#define    INT_STATUS         0x6
#define    RT_STATUS          0xc
#define    RT_STATUS_RD       0xe
#define    RT_STACK_POINTER 0x3
#define RT_ADDRESS 0xe   //地址 7
#define BM_ADDRESS_ENBLE 0x280


unsigned int add_data[] = {
                            0x100,0x141,0x142,0x143,0x144,0x145,0x146,0x147,0x148,0x149,
                            0x14a,0x14b,0x14c,0x14d,0x14e,0x14f,0x150,0x151,0x152,0x153,
                            0x154,0x155,0x156,0x157,0x158,0x159,0x15a,0x15b,0x15c,0x15d,
                            0x15e,0x161,0x162,0x163,0x164,0x165,0x166,0x167,0x168,0x169,
                            0x16a,0x16b,0x16c,0x16d,0x16e,0x16f,0x170,0x171,0x172,0x173,
                            0x174,0x175,0x176,0x177,0x178,0x179,0x17a,0x17b,0x17c,0x17d,
                            0x17e,0x187
                           };
unsigned int dat_data[] = {
                            0x000,0x260,0x260,0x260,0x260,0x260,0x260,0x260,0x260,0x260,
                            0x260,0x260,0x260,0x260,0x260,0x260,0x260,0x260,0x260,0x260,
                            0x260,0x260,0x260,0x260,0x260,0x260,0x260,0x260,0x260,0x260,
                            0x260,0x260,0x260,0x260,0x260,0x260,0x260,0x260,0x260,0x260,
                            0x260,0x260,0x260,0x260,0x260,0x260,0x260,0x260,0x260,0x260,
                            0x260,0x260
                           };
unsigned int send_pt[32]=
                            {
                                0x0000,0xffff,0x0000,0xffff,
                                0x0000,0xffff,0x0000,0xffff,
                                0x0000,0xffff,0x0000,0xffff,
                                0x0000,0xffff,0x0000,0xffff,
```

```
                                        0x0000,0xffff,0x0000,0xffff,
                                        0x0000,0xffff,0x0000,0xffff,
                                        0x0000,0xffff,0x0000,0xffff,
                                        0x0000,0xffff,0x0000,0xffff
                                    };



    void bm_init();
    void M1553birq();
    void bm_recv_message();
    void delay();

    unsigned                                                                                    int
rt_recv_num,rt_address,rt_recv,rt_right_recv,rt_error_recv,recv_end_flag,command_word,word_count,num_count;
    unsigned int send_end_flag,recv_end_flag,rt_address_count,sub_address_count,stack_offset,stack_offset1,count;
```

MAIN.C

```
    #include"bm.h"
    main()
    {
            printf("/**********************************/");
            printf("\n      BC 控制器 bm 模式测试！ ");
            printf("\n      BC-->RT0    ");
            printf("\n/**********************************/");
            INTMASK=0;
            INTCLEAR=(1<<12);
            catch_interrupt(M1553birq,12);
            delay();
            send_end_flag=1;
            rt_recv_num=0;
            rt_address_count=0;
            sub_address_count=1;
            num_count=0;
            count=0;
            INTMASK=(1<<12);
            rt_init();
            *(volatile unsigned int *)(CTRL_REG_BASE+ START_RESET_REG *4)        = 0x2;
            while(send_end_flag);
    void M1553birq()
    {
        unsigned int i=0,intsta,pending,message_block_address,message_block_address2,send_dat=0x5555,ww;

        pending=INTPEND;
```

```
        intsta=*(volatile unsigned int *)(CTRL_REG_BASE+INT_STATUS*4);
        *(volatile unsigned int *)(CTRL_REG_BASE+START_RESET_REG*4)=0x06;
        INTCLEAR=(1<<12);
        bm_recv_message();
    }
    void bm_init()
    {
      unsigned int i,send_data;
      send_data=0x5555;
    //configure register
            *(volatile unsigned int *)(     CTRL_REG_BASE+START_RESET_REG*4)=0x1;     //复位 1553B 寄存器
     *(volatile unsigned int *)(   CTRL_REG_BASE+CFG_REG1*4)=0x4000;      //设置 bm 模式
     *(volatile unsigned int *)(   CTRL_REG_BASE+CFG_REG2*4)    = 0x388; // --time tag resolution 2 us
     *(volatile unsigned int *)(CTRL_REG_BASE+     CFG_REG3*4)  = 0x000; // --no override mode T/R* error
     *(volatile unsigned int *)(CTRL_REG_BASE+     CFG_REG4*4)  = 0x0188; // --1ST retry ALT bus,2ND retry ALT bus
    //      *(volatile unsigned int *)(CTRL_REG_BASE+CFG_REG5*4)       =  0x640|0x0;  //set  RT  ADDRESS  0,TIME
OVERRIDE   ANSWER   130us
     *(volatile unsigned int *)(   CTRL_REG_BASE+INT_MASK*4)=0x0001;      //EOM 消息中断使能
     delay();
     printf("\n 配置堆栈");
     *(volatile unsigned int *)(   CTRL_RAM_BASE+0x100*4)=0x0;

     for(i=0;i<0x80;i++)
     {

       *(volatile unsigned int *)(CTRL_RAM_BASE +( BM_ADDRESS_ENBLE + i)*4) = 0xffff;//配置 RT 工作模式
     }
     printf("\n bm 初始化结束！ ");
     for(i=0;i<0xff;i++);
     }
    void   bm_recv_message()
    {
     unsigned int i,k,block_state,data_address,temp,tr_bit;
    //get block   stack   message
     delay();
     stack_offset1=*(volatile unsigned int *)(CTRL_REG_BASE+RT_STACK_POINTER*4)-4;
     data_address=*(volatile unsigned int *)(CTRL_RAM_BASE+(stack_offset1+2)*4);
     command_word=*(volatile unsigned int *)(CTRL_RAM_BASE+(stack_offset1+3)*4);

     word_count=command_word&0x1f;
     tr_bit=command_word>>10;
     sub_address_count=(command_word>>5)&0x1f;
     rt_address_count=(command_word>>11)&0x1f;
```

```
//      printf("\n %x",*(volatile unsigned int *)0x80050004);
}
}
void delay()
{
unsigned int    i;
 for(i=0;i<0xfff;i++);
}
```