CDCTL01A 数据手册

1 简述

CDCTL01A 是一款 SPI 接口的 UART 控制器，UART 端口固化使用 CDBUS 协议。(或简称 SPI 接口的 CDBUS 控制器。)

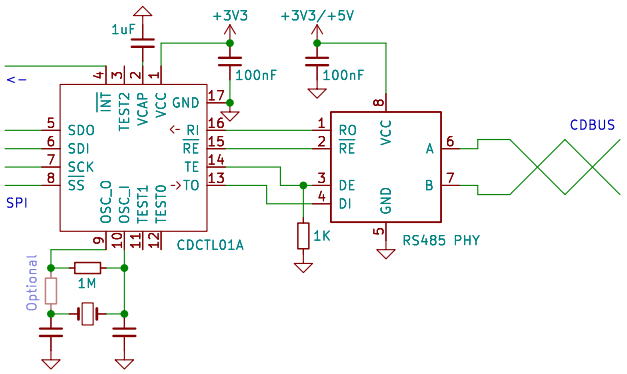
CDBUS 是一种面向串口和串口总线的简单协议，常用于 RS-485 总线，通过硬件仲裁等机制避免数据冲突，各节点可随意收发数据包，突破只能单主机轮询的限制。

2 特性

- 此芯片除了用于 RS-485 通讯，还支持 M-LVDS、单线串口总线、串口扩展等多种用途。
- 支持多主通讯、对等通讯、组播通讯等功能。
- 支持双速率仲裁、单速率高速对等通讯、传统以及全双工等多种模式。
- 最高 50Mbps UART 速率。
- 最高 50MHz SPI 时钟。
- 8 个接收缓存, 2 个发送缓存。(每个缓存大小为 256 bytes.)
- 小封装：QFN16 3x3mm.
- 3.3V (±10%) 单电源供电。
- -40 ~ 125 ℃ 工作范围。
- 支持晶振和外部时钟输入。
- 支持 4 线和 3 线 SPI（SDO 和 SDI 可短接)。
- RI 管脚容忍 5V 输入。
- 超宽主频范围：32KHz ~ 150MHz.
- 环保无铅封装。

3 应用

- 工业自动化
- 机器人
- 汽车电子
- 智慧城市
- 消费电子
- IoT 物联网



参考电路

(使用晶振时 1MΩ 电阻不可省略。串接电阻可调整晶振振幅，可以省略或接 240Ω 左右电阻。)

4 CDBUS 协议

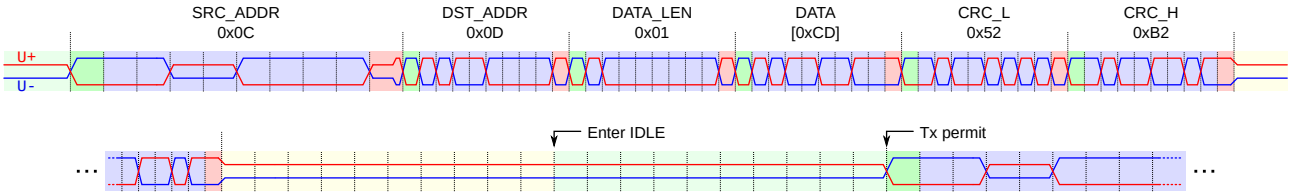
CDBUS 是一种异步串行通信协议，它有一个 3 字节的报头：`[src_addr, dst_addr, data_len]`，然后是用户数据，最后是 2 个字节的 CRC（与 MODBUS CRC 相同）。

字节层面的 CDBUS 协议可以直接用于传统的串口通讯，譬如传统 UART、RS-232、RS-485、USB 虚拟串口。
而 bit 层面的完整 CDBUS 协议需要使用专门的硬件控制器（或者软件模拟），以实现冲突避免、更快的速率和很强的实时性。

4.1 仲裁模式 (CDBUS-A)

- 它引入了一种仲裁机制，能像 CAN 总线一样自动避免冲突。
- 支持双波特率，实现高速通信，高速阶段的波特率最高可达 `sysclk ÷ 3`.（例如 sysclock 为 150MHz 时的波特率为 50Mbps.）
- 支持单播、组播和广播。
- 最大装载用户数据大小为 253 字节。
- 硬件打包、拆包、验证和过滤，节省您的时间和 CPU 占用率。
- 兼容传统 RS-485 硬件（仲裁功能仍然有效）。

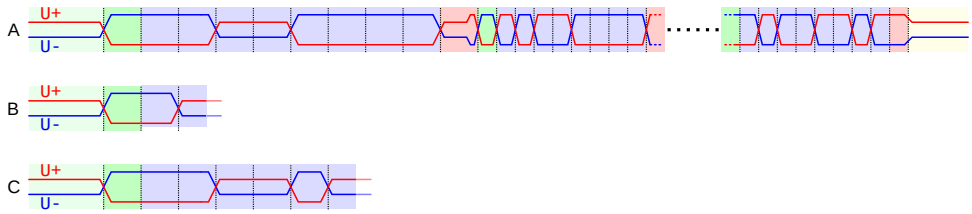
协议时序示例，只包含一个字节用户数据：
（您可以设置进入空闲和等待发送的时间长度。）



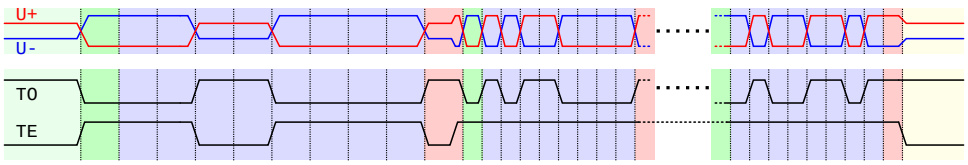
备注：

- 当高优先级节点需要发送不重要的数据时，可以动态改大发送等待时间 (TX_PERMIT_LEN)。

仲裁示例：



TO 和 TE 引脚的示例波形：

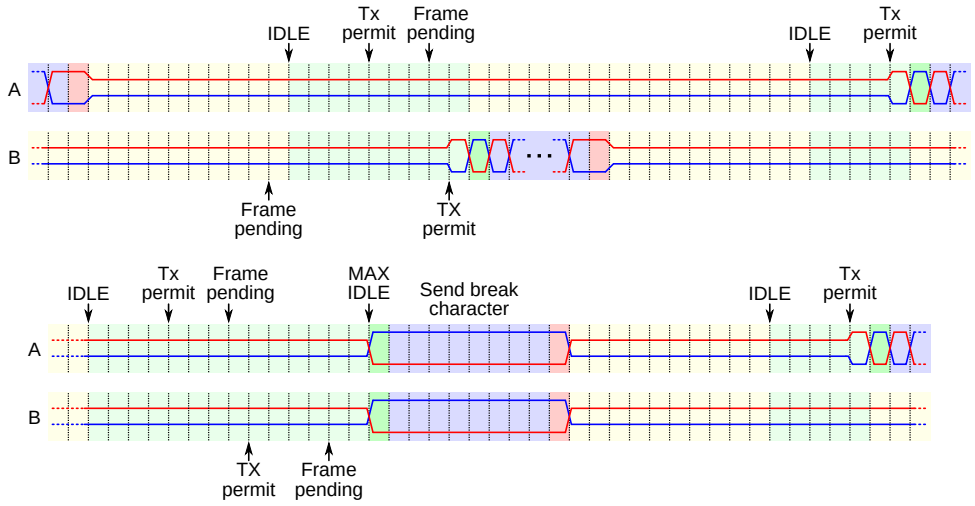


RX 接收数据采样点位于 1/2 bit; TX 回读数据采样点位于 3/4 bit.

4.2 Break 字符同步模式 (CDBUS-BS)

在 CDBUS-A 模式下，如果低速部分耗时较长，就会成为通信效率的瓶颈。
若要解决这个问题，可通过 CDBUS-BS 模式实现单速率点对点总线通信：

- 不同节点的 TX_PERMIT_LEN 参数配置不同，需要足够大的差异以避免冲突。
- 如果任何节点在发送许可时刻之前有等待发送的数据帧，则从发送许可时刻开始发送。
- 否则要等到空闲时间超过 MAX_IDLE_LEN 后，且当有数据帧等待发送时，先发送一个中断字符使总线退出空闲状态。

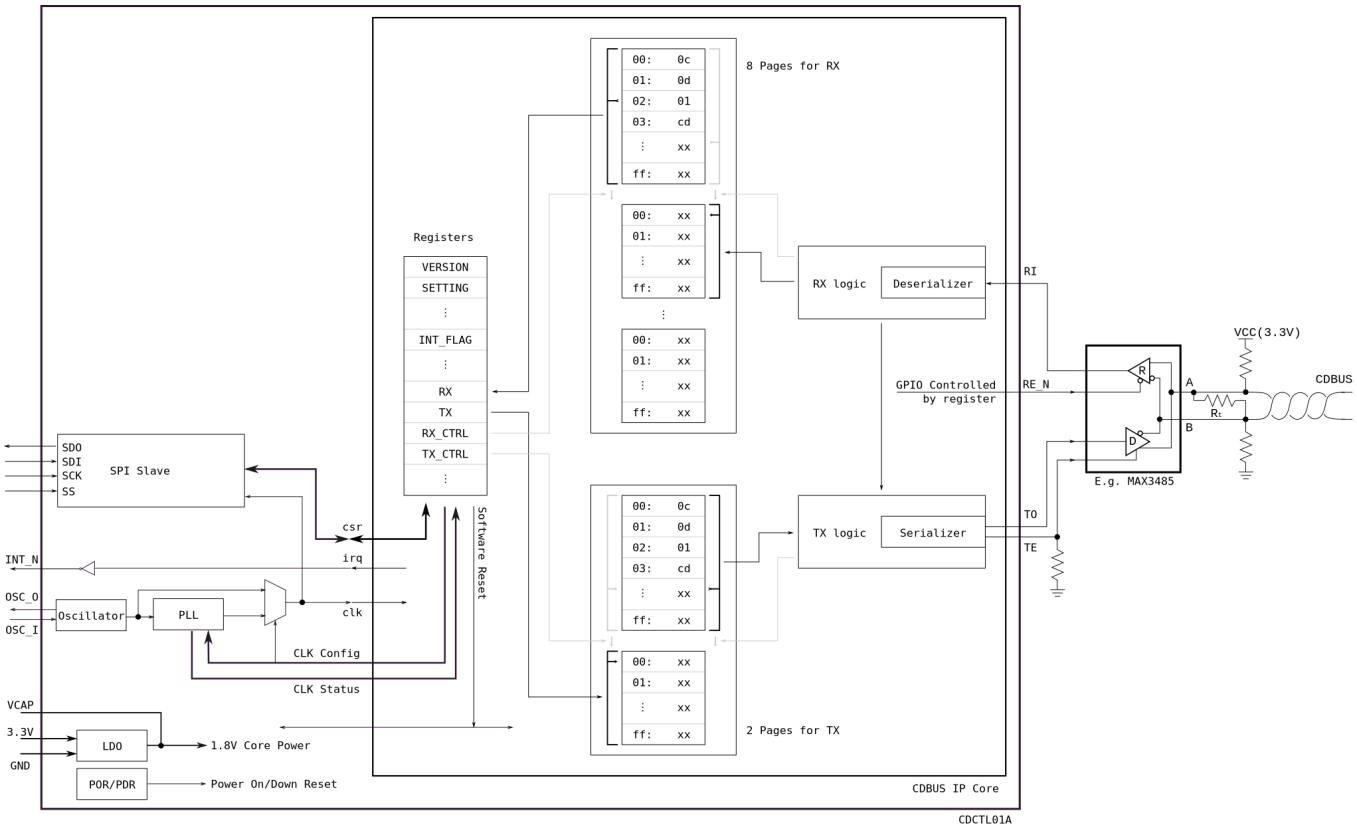


CDBUS-BS 模式适用于节点较少的高速应用，也适用于软件模拟实现。

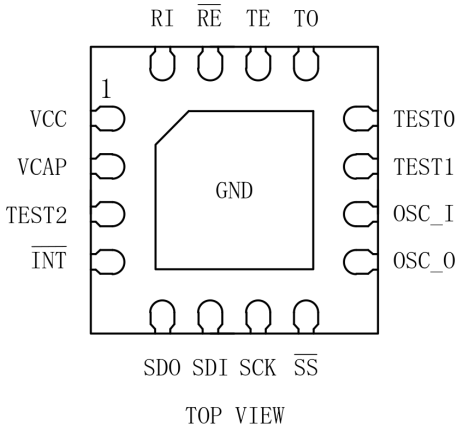
5 CDBUS IP 核

芯片源码和更多资料：https://cdbus.org

5.1 原理框图



6 管脚定义

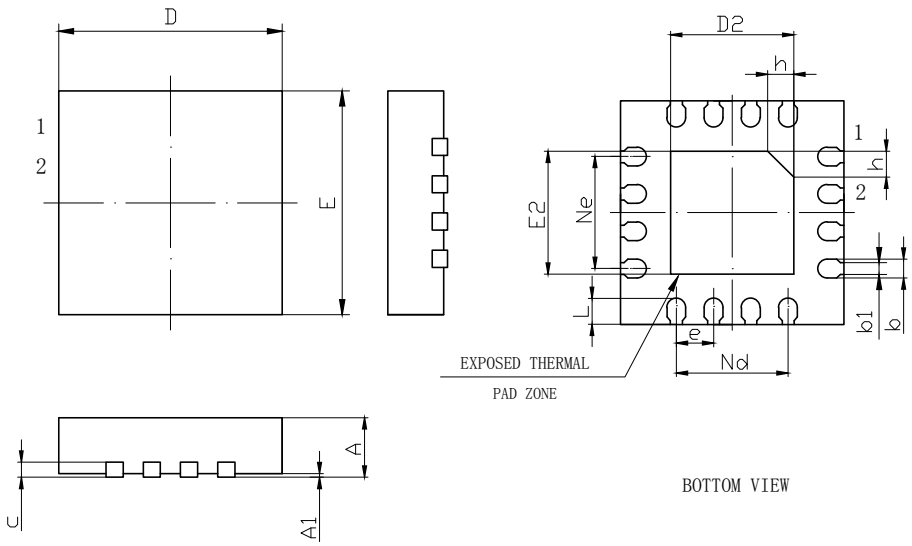


No	名称	I/O	内部上下拉	描述
17	GND			地
1	VCC			3.3V 电源 (±10%), 外接 100nF 陶瓷电容旁路到地
2	VCAP			内部 1.8V (±10%) 100mA LDO 输出, 外接 1uF 陶瓷电容旁路到地
3, 11, 12	TESTx	I	下拉	留空或接地

4	$\overline{INT}$	O	上拉	中断引脚，开漏（默认）或推挽输出
5	SDO	O	-	SPI MISO
6	SDI	I	-	SPI MOSI
7	SCK	I	-	SPI 时钟
8	$\overline{SS}$	I	-	SPI 片选
9	OSC_O	O	-	OSC 输出（使用外部时钟输入时留空）
10	OSC_I	I	-	OSC 输入或外部时钟输入
13	TO	O	-	TX 输出，连接至 RS-485 PHY 的发送引脚
14	TE	O	-	TX 使能，连接至 RS-485 PHY 的发送使能引脚
15	$\overline{RE}$	O	-	GPIO 推挽输出，通常用于控制 RS-485 PHY 的接收使能
16	RI	I	-	RX 输入，连接至 RS-485 PHY 的接收引脚（5V 容差）

7 规格

7.1 尺寸规格



SYMBOL	MILLIMETER		
	MIN	NOM	MAX
A	0.80	0.85	0.90
A1	—	0.02	0.05
b	0.18	0.25	0.30
b1	0.11	0.16	0.21
c	0.10	0.15	0.20
D	2.90	3.00	3.10
D2	1.55	1.65	1.75
e	0.50BSC		
Ne	1.50BSC		
Nd	1.50BSC		
E	2.90	3.00	3.10
E2	1.55	1.65	1.75
L	0.30	0.35	0.40
h	0.30	0.35	0.40

7.2 极限参数

参数	Min.	Max.
存储温度	−55 ℃	150 ℃
静电放电 人体模型 (ESD-HBM) [1]	-	±2000 V
静电放电 电器件模型 (ESD-CDM) [2]	-	±750 V
高温 Latch-Up [3]	-	±200 mA / +1.5 VccMax

1. 参考规范 AEC-Q100-002. 间隔 0.3 秒击发 1 次脉冲。
2. 参考规范 AEC-Q100-011.
3. 参考规范 JESD78F:2022, Ta=+125℃.

7.3 使用条件

参数	Min.	Max.
使用温度	−40 ℃	125 ℃

7.4 直流电气特性

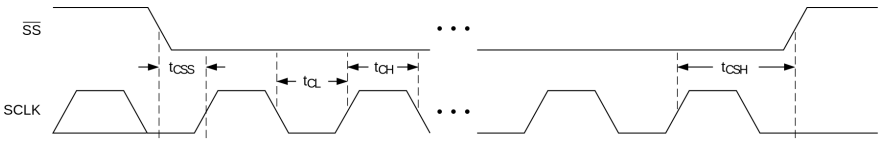
参数	Min.	Typ.	Max.
VCC 供电	-10%	3.3 V	+10%
V_{IL}	-0.3 V	-	0.8 V
V_{IH}	2.0 V	-	3.6 V
V_{IH} (RI 脚)	2.0 V	-	5.5 V
V_{OL}	-	-	0.4 V
V_{OH}	2.4 V	-	-
I_{OL}, I_{OH}	-	12 mA	-
I_{OL}, I_{OH} (OSC_O 脚)	-	2 mA	-
Input 或 I/O 漏电流	-	-	+/-10 uA
I_{VCC} ($F_{SYS} = 32KHz$)	-	140 uA	-
I_{VCC} ($F_{SYS} = 12MHz$, pll off)	-	2.8 mA	-
I_{VCC} ($F_{SYS} = 12MHz$, pll on)	-	8.7 mA	-
I_{VCC} ($F_{SYS} = 60MHz$)	-	10.6 mA	-
I_{VCC} ($F_{SYS} = 150MHz$)	-	14.3 mA	-

7.5 时序规格

符号	参数	Min.	Max.
$F_{CRYSTAL}$	晶振频率	4 MHz	32 MHz
F_{EXT}	外部时钟输入	32 KHz	32 MHz
F_{SYS}	系统主时钟频率	32 KHz	150 MHz
F_{SCK}	SPI 时钟频率	-	50 MHz
F_{UART}	波特率	-	50 Mbps

对 RX 和 TX 寄存器读写的额外限制：$F_{SCK} \leq F_{SYS} \times 80\%$

向任何寄存器写入都需要满足：$t_{CSH} \geq \frac{1}{F_{SYS} \times 80\%}$



8 寄存器列表

寄存器名称	地址	可读写	默认值	描述 （未说明默认 8-bit 位宽）
VERSION	0x00	RD	0x10	硬件版本号
CLK_CTRL	0x01	RD/WR	0x00	时钟控制
SETTING	0x02	RD/WR	0x10	配置
IDLE_WAIT_LEN	0x04	RD/WR	0x0a	进入空闲的等待时间
TX_PERMIT_LEN_L	0x05	RD/WR	0x14	允许发送的等待时间 (10 bits)
TX_PERMIT_LEN_H	0x06	RD/WR	0x00	
MAX_IDLE_LEN_L	0x07	RD/WR	0xc8	BS 模式下的最大空闲等待时间 (10 bits)
MAX_IDLE_LEN_H	0x08	RD/WR	0x00	

TX_PRE_LEN	0x09	RD/WR	0x01	在 TO 输出之前多久使能 TE (2 bits)
FILTER	0x0b	RD/WR	0xff	本机地址
DIV_LS_L	0x0c	RD/WR	0x67	低速波特率设置 (16 bits)
DIV_LS_H	0x0d	RD/WR	0x00	
DIV_HS_L	0x0e	RD/WR	0x67	高速波特率设置 (16 bits)
DIV_HS_H	0x0f	RD/WR	0x00	
INT_FLAG	0x10	RD	n/a	状态寄存器
INT_MASK	0x11	RD/WR	0x00	中断掩码寄存器
RX	0x14	RD	n/a	读 RX 页
TX	0x15	WR	n/a	写 TX 页
RX_CTRL	0x16	WR	n/a	RX 控制
TX_CTRL	0x17	WR	n/a	TX 控制
RX_ADDR	0x18	RD/WR	0x00	当前 RX 页读指针（极少用）
RX_PAGE_FLAG	0x19	RD	n/a	当前 RX 页标记
FILTER_M0	0x1a	RD/WR	0xff	组播地址过滤器 0
FILTER_M1	0x1b	RD/WR	0xff	组播地址过滤器 1
PLL_ML	0x30	RD/WR	0x12	PLL M[7:0] (M: 9 bits)
PLL_OD_MH	0x31	RD/WR	0x20	PLL OD 和 M[8]
PLL_N	0x32	RD/WR	0x00	PLL N (5 bits)
PLL_CTRL	0x33	RD/WR	0x01	PLL 控制
PIN_INT_CTRL	0x34	RD/WR	0x00	$\overline{\text{INT}}$ 脚控制
PIN_RE_CTRL	0x35	RD/WR	0x00	$\overline{\text{RE}}$ 脚控制
CLK_STATUS	0x36	RD	0x01	时钟状态寄存器

8.1 CLK_CTRL:

字段	描述
[7]	软件复位：写 1 复位器件
[0]	时钟选择：0: OSC 输入，1: PLL 输出

8.2 SETTING:

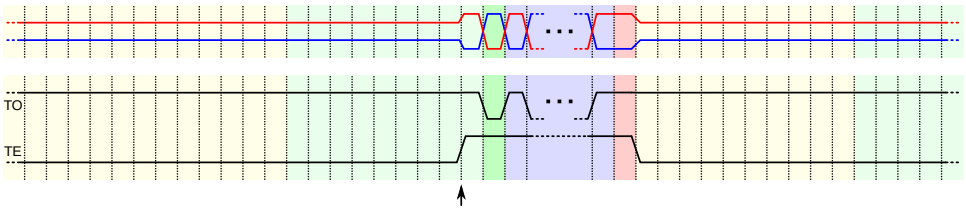
字段	描述
[6]	全双工模式
[5]	Break 字符同步 (BS) 模式
[4]	仲裁模式 (A)
[3]	接收保存受损的数据帧
[2]	由用户填充和检验 CRC
[1]	反转 TO 脚输出
[0]	启用 TO 和 TE 引脚的推挽输出

TO 脚默认是开漏输出，TE 脚默认是高阻输出。

[6]	[5]	[4]	DESCRIPTION
0	0	0	传统半双工模式
0	0	1	CDBUS-A 仲裁模式（默认）
0	1	0	CDBUS-BS 模式
1	0	0	全双工模式

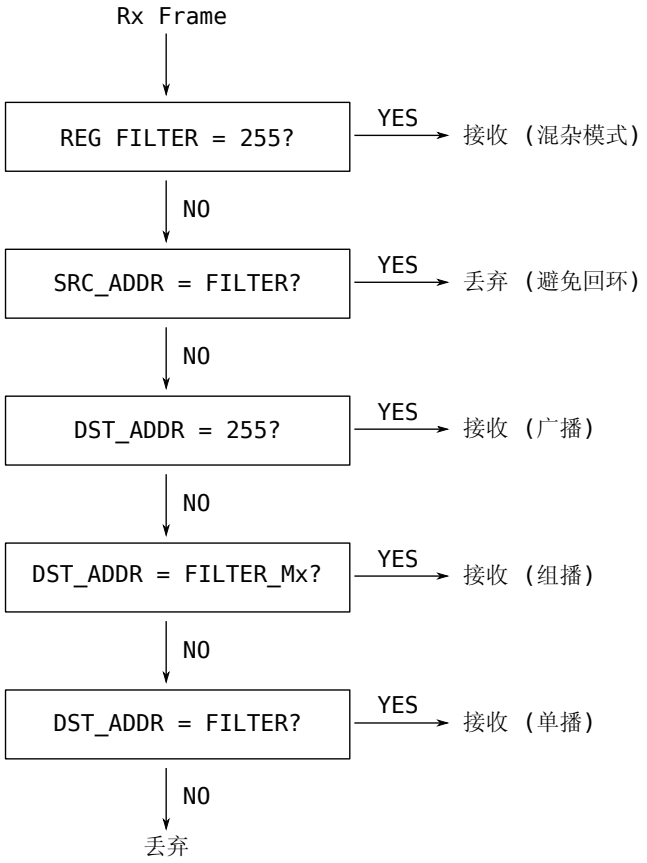
8.3 TX_PRE_LEN:

TO 和 TE 引脚波形示例 (TX_PRE_LEN = 1 bit):



未用于仲裁模式和 BS 模式自动生成的 break 字符。

8.4 FILTERS:



FILTER_Mx 的默认值为 0xff，表示未启用。0xff 以外的任意地址都可以选择做为组播地址。

8.5 DIV_xx_x:

波特率分频器值：

$$div\_xx[15:0] = \frac{sysclk}{baudrate} - 1$$

最小值为 2.
使用单速率时，DIV_HS 需要和 DIV_LS 设置相同的值。

8.6 INT_FLAG:

字段	描述
[7]	1: 发送错误：发送为 0, 但接收采样为 1
[6]	1: 检测到发送碰撞
[5]	1: TX 页被硬件释放
[4]	1: RX 错误：帧损坏
[3]	1: RX 丢失：无空页用于接收
[2]	1: 接收到 break 字符
[1]	1: 已有 RX 页可读取
[0]	1: 总线处于空闲模式

读取该寄存器将自动清除 bit7、bit6、bit4、bit3 和 bit2.

8.7 INT_MASK:

```
irq = ((INT_FLAG & INT_MASK) != 0)
int_n = !irq
```

8.8 RX_CTRL:

字段	描述
[4]	复位 RX 模块
[1]	切换 RX 页
[0]	重置 RX 页读指针

8.9 TX_CTRL:

字段	描述
[5]	发送 break 字符
[4]	终止当前发送
[1]	切换 TX 页
[0]	重置 TX 页写指针

8.10 RX_PAGE_FLAG:

值为 0 表示当前 RX 页中的帧正确；
非 0 表示当前 RX 页中的帧存在错误，并指向最后接收到的字节，包含 CRC.
如果未启用 '保存破损帧'，则始终为 0。

8.11 PLL_OD_MH:

字段	描述
[5:4]	PLL OD
[0]	PLL M[8]

8.12 PLL_CTRL:

字段	描述
[4]	启用 PLL：0: 禁用 PLL，1: 开启 PLL
[0]	PLL 休眠：1: 休眠，0: 运行

该寄存器中的保留位必须保持为 0。

PLL 输出时钟：

$$pll\_output = \frac{osc\_input}{pll\_n + 2} \times (pll\_m + 2) \times \frac{1}{2^{(pll\_od[1]+pll\_od[0])}}$$

当 OSC 输入频率等于 12MHz 时，默认 PLL 参数的输出频率为 60MHz.

PLL 参数必须满足以下条件：

$$1MHz \leq \frac{osc\_input}{pll\_n + 2} \leq 15MHz$$

$$100MHz \leq \frac{osc\_input}{pll\_n + 2} \times (pll\_m + 2) \leq 500MHz$$

切换至使用 PLL 时钟：

- 如有需要，更改 PLL 的 N、M 和 OD 值
- 写 0x10 到 PLL_CTRL 寄存器
- 写 0x01 到 CLK_CTRL 寄存器

8.13 PIN_INT_CTRL:

字段	描述
[4]	输出模式：0: 开漏，1: 推挽

8.14 PIN_RE_CTRL:

字段	描述
[4]	输出模式：0: 禁用（高阻抗），1: 推挽
[0]	输出值：0: 低，1: 高

8.15 CLK_STATUS:

字段	描述
[2]	时钟切换状态：1: 已完成，0: 进行中
[1]	系统时钟为 PLL 输出
[0]	系统时钟为 OSC 输入

9 SPI 接口

多字节读写通常用于访问 REG_RX 和 REG_TX 寄存器。

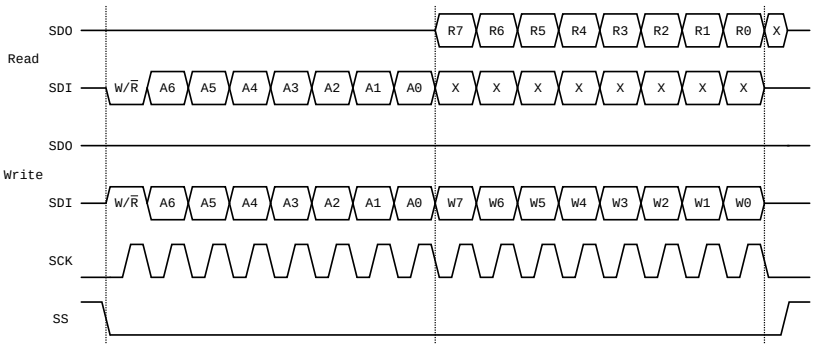
读取或写入取决于 W/$\overline{R}$ 位：

0: 读
1: 写

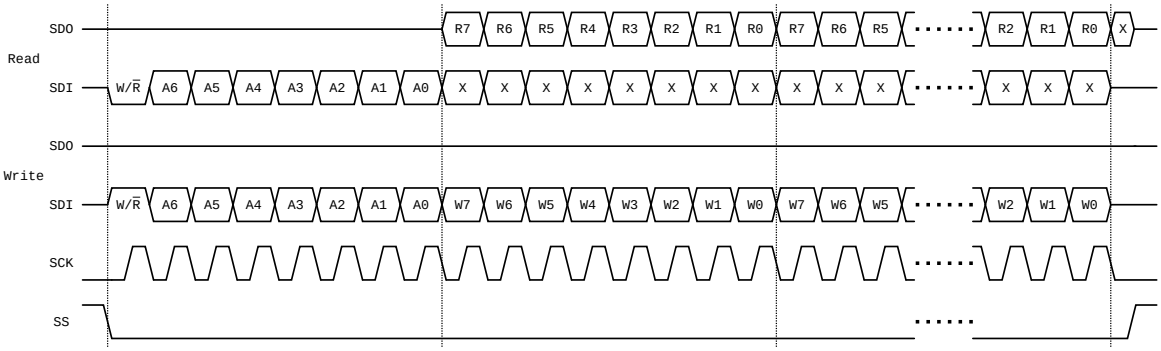
字段	描述

Ax	寄存器地址
Wx	写数据
Rx	读数据
X	忽略

单字节读写：



多字节读写：



10 操作示范

10.1 初始化

```
cd_write(REG_CLK_CTRL, 0x80);                   // 软件复位器件
cd_write(REG_PIN_RE_CTRL, 0x10);                // 使能 RE_N 脚，输出低电平
cd_write(REG_SETTING, 0x11);                    // 使能 TX 和 TX_EN 脚
cd_write(REG_FILTER, 0x0c);                     // 设置 FILTER 过滤寄存器

// 设置波特率，单速率时 LS 和 HS 要设置相同的值
cd_write(REG_DIV_LS_L, 11);                     // 1 Mbps @ 12MHz sysclk
cd_write(REG_DIV_LS_H, 0);
cd_write(REG_DIV_HS_L, 2);                      // 4 Mbps @ 12MHz sysclk
cd_write(REG_DIV_HS_H, 0);

// 复位接收模块，忽略波特率配好前收到的一些错误状态 (可选)
// cd_write(REG_RX_CTRL, 0x11);

// 使能中断 (可选，主控也可以定期查询 INT_FLAG)
// cd_write(REG_INT_MASK, BIT_FLAG_TX_ERROR | BIT_FLAG_RX_ERROR \
            | BIT_FLAG_RX_LOST | BIT_FLAG_RX_PENDING);
```

芯片上电需要时间，建议等待 50 ms 或不停尝试直到读到正确 VERSION 版本号再进行初始化操作。

10.2 发送

```
uint8_t tx_buf[] = {
    0x0c, 0x0d, 0x02,                          // 原地址、目标地址、数据长度
    0x01, 0x00                                 // 2 字节示范数据，无 CRC
};

cd_write_chunk(REG_TX, tx_buf, tx_buf[2] + 3);  // 写入一个 CDBUS 数据包，不含 CRC
while (!(cd_read(REG_INT_FLAG) & 0x20));        // 等待之前的发送完成
cd_write(REG_TX_CTRL, 0x03);                    // 提交数据包，启动发送
```

FILTER 寄存器仅用于接收过滤，不影响发送。发送时，发送方应自觉使用自身地址作为数据帧的 src_addr, 非自身地址通常用于环路测试，避免数据包被过滤掉。

10.3 接收

```
while (!(cd_read(REG_INT_FLAG) & 0x02));        // 等待接收数据包
cd_read_chunk(REG_RX, rx_buf, 3);               // 读数据包头部
cd_read_chunk(REG_RX, rx_buf + 3, rx_buf[2]);   // 读数据包内容，不含 CRC
cd_write(REG_RX_CTRL, 0x03);                    // 释放数据包
```

读数据包头部和内容也可以在一次 $\overline{\text{SS}}$ 拉低期间完成。

11 版权声明

12 联络信息

- 销售和客户支持：sales@dukelec.com
- 技术支持：support@dukelec.com
- 商业合作：info@dukelec.com
- 公司网站：https://dukelec.com

13 文档修订记录

- 20260123 (v1.3): 改用流程图说明过滤流程；示范代码增加说明。
- 20240304 (v1.2): 修改简介。
- 20231125 (v1.1): 增加 ESD 和 Latch-Up 相关数据。
- 20230802 (v1.0): 初建文档。