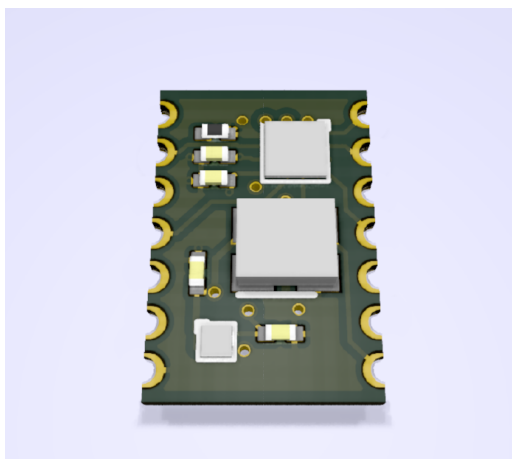


CD485 數據手冊

Duke Fong

June 2, 2017



Contents

1	功能描述	3
1.1	概述	3
1.2	亮點	3
2	CD485 協議	3
3	硬件	4
3.1	電路參考	4
3.2	內部結構	5
3.3	引腳定義	5
3.4	尺寸規格	6
4	寄存器列表	6
5	流程圖	9
5.1	RX	10
5.2	TX	11
6	設備接口	11
6.1	SPI	12
6.2	I2C	12

7	操作示例	12
7.1	初始化	12
7.1.1	兼容模式和傳統模式	13
7.2	TX	13
7.3	RX	13

1 功能描述

1.1 概述

CD485 是一種基於 RS485 總線的通訊協議，它同時也指實現此協議的硬件和使用此協議的總線。

CD485 協議由 Duke Fong 於 2010 年設計，以便捷、多主機、高速率為目標。

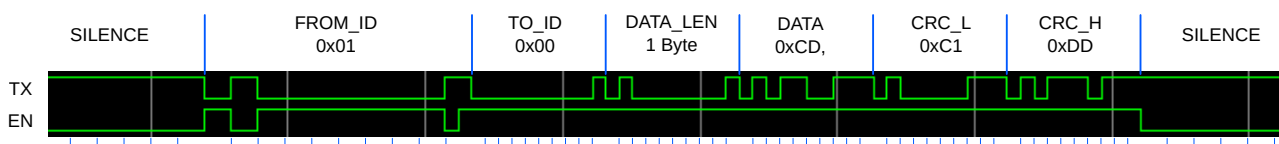
1.2 亮點

當前硬件支持的特性：

- CD485 總線上各節點均可主動發起傳輸，通過節點地址仲裁以避免衝突。
- 總線上每個數據包可以含有 0 ~ 253 字節用戶數據
- 共有 8 個 RX 緩存頁和 2 個 TX 緩存頁，每一頁 256 字節
- 硬件為每個數據包自動完成 16 位 CRC 生成與校驗
- 波特率：103 bps 至 9 Mbps（另有模組支持到 36 Mbps）
- 可分別為仲裁字段和後續數據設置不同的波特率
- 兼容傳統 RS485 總線
- 提供 SPI 和 I2C 接口
- 配置和操作便捷

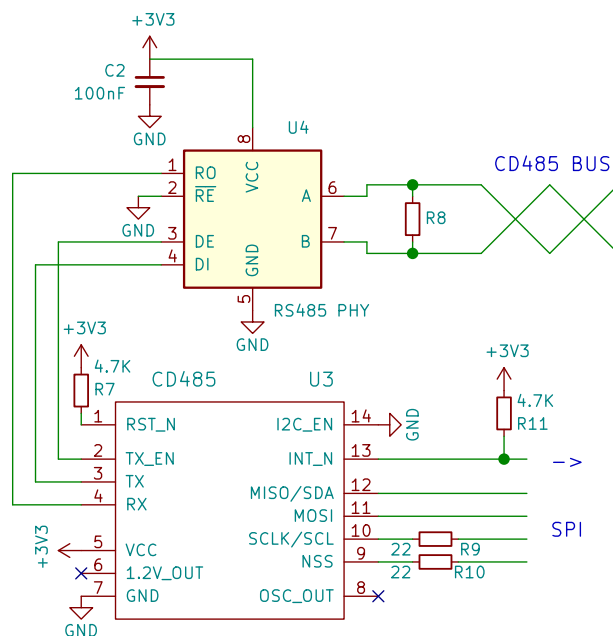
2 CD485 協議

參考時序：

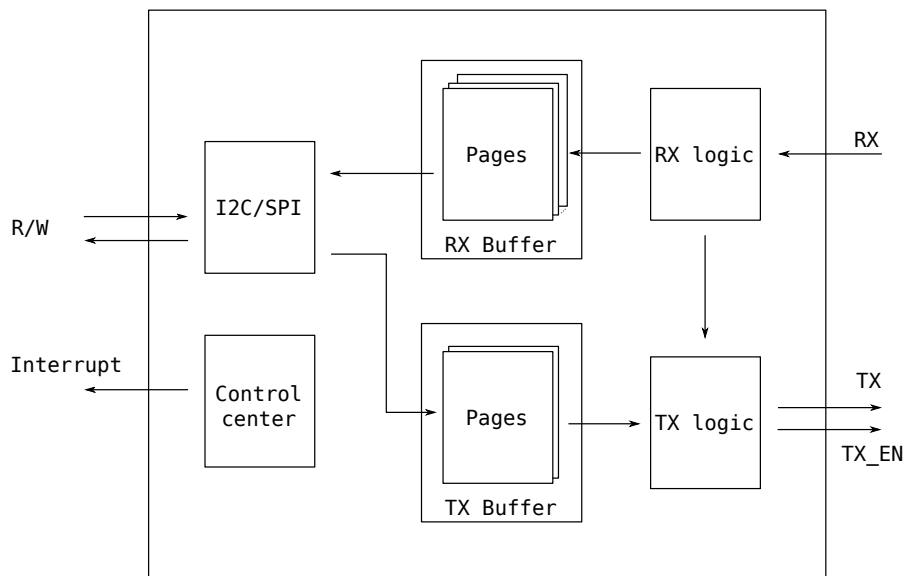


字段	長度 (bytes)	用途
SILENCE	0~25.5 Default: 2 (20 bits)	分隔數據包 總線在數據包結束後繼續保持 SILENCE 位長的時間為 1, 總線便進入空閒模式。

3 硬件

[illegible]

3.2 內部結構



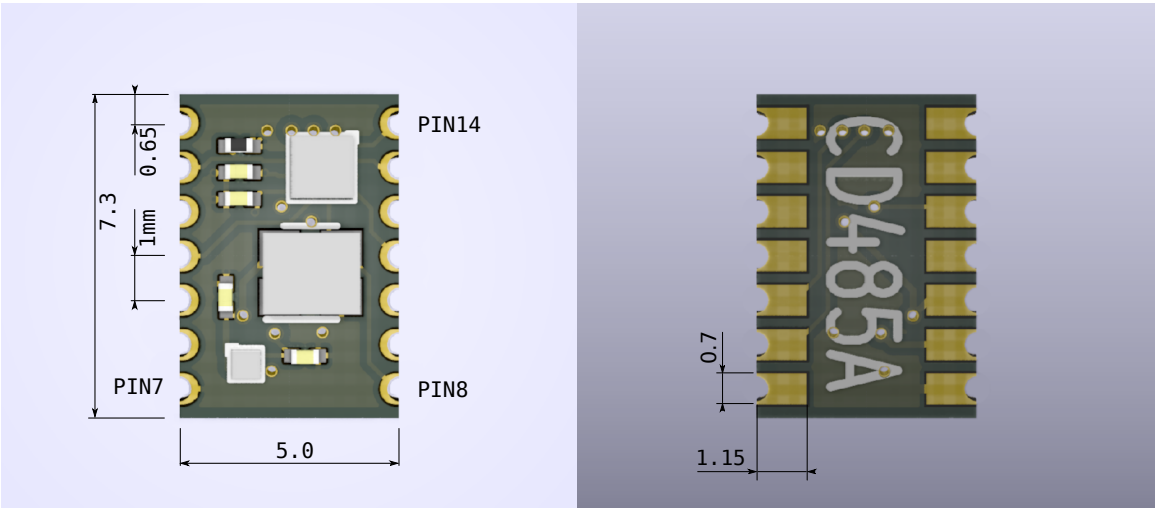
3.3 引腳定義

No	Name	I/O	Pull	Description
1	RST_N	I	-	復位，低有效，需要加外部上拉
2	TX_EN	O	D	使能腳，連接 RS485 PHY（內部 2 K Ω 下拉電阻）
3	TX	O	U	發送腳，連接 RS485 PHY
4	RX	I	-	接收腳，連接 RS485 PHY
5	VCC			電源 3.3V \pm 5%, \leq 5mA
6	1.2V_OUT			內部 1.2V LDO 輸出
7	GND			地
8	OSC_OUT	O		內部 27MHz 時鐘輸出
9	NSS	I	U	SPI NSS（上電或復位時請勿拉低）
10	SCLK/SCL	I	U	SPI/I2C CLOCK
11	MOSI	I	-	SPI MOSI
12	MISO/SDA	I/O	U	SPI MOSI / I2C SDA
13	INT_N	O	-	中斷，低有效，開漏輸出
14	I2C_SEL	I	-	輸入高選擇 I2C 模式；低為 SPI 模式

上拉均為 50 K Ω ，且僅在復位結束後有效。

注意：在上電或復位時的一小段時間，TX 腳會輸出低、NSS 輸出高、SCLK/SCL 輸出一段脈衝，在這段時間裏請不要驅動這些引腳，最好在模塊和 CPU 間為 NSS 和 SCLK/SCL 串電阻做保護。

3.4 尺寸规格



4 寄存器列表

地址	名稱	R/W	說明
0x00	VERSION	R	硬件版本，當前為：0xC0

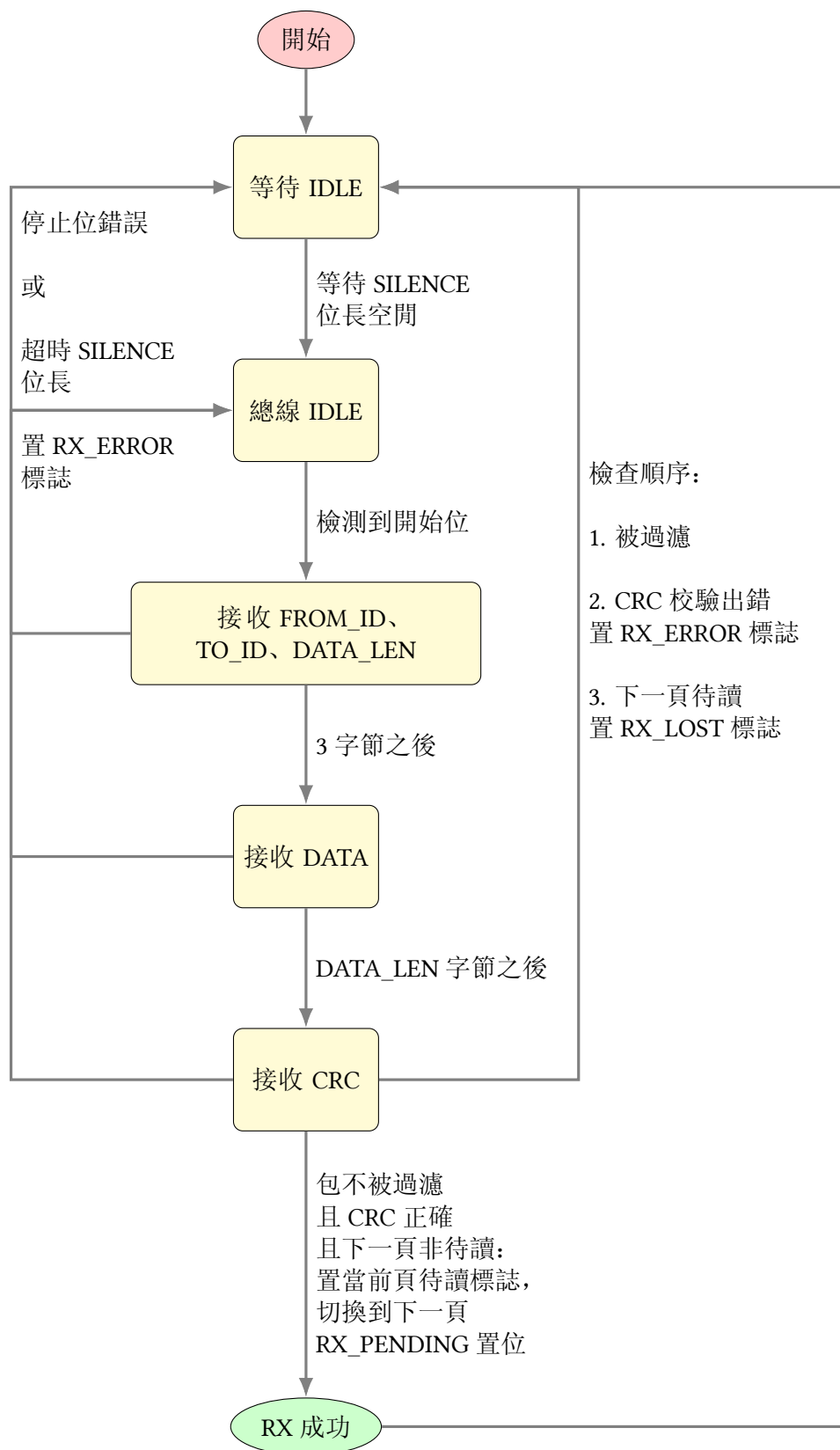
0x01	SETTING	RW	Bits:																								
			<div>bit0</div> <div>OUTPUT_EN</div> <div>如未置位，TX 和 TX_EN 輸出高阻。</div>																								
			<div>bit1</div> <div>NO_ARBITRATE</div> <div>關閉仲裁功能，輸出時 TX_EN 一直有效。</div>																								
			<div>bit2</div> <div>USER_CRC</div> <div>關閉硬件 CRC</div> <div>如果關閉，用戶需要自行計算兩字節 CRC 並追寫在數據之後，且在讀完數據之後再多讀兩字節 CRC 數據以便自行校驗。用戶數據最大長度將會降至 251 字節。</div>																								
			<div>bit3</div> <div>NO_DROP</div> <div>如果設置，當接收出錯置位 RX_ERROR 標誌時會同時保留出錯的數據包。</div> <div>通過 RX_PAGE_FLAG 判斷當前 RX 緩存頁中的數據包是否有錯。</div>																								
			<div>bit[5:4]</div> <div>G_CLOCK_CONF</div> <div>系統時鐘選擇：</div> <div> <div>x0: 6.75 MHz</div> <div>x1: 27 MHz</div> </div>																								
			<div>bit[7:6]</div> <div>TX_EN_ADVANCE（僅 NO_ARBITRATE 置位時有效）</div> <div>TX_EN 提前 TX 使能的位長（額外加上 1 個系統時鐘週期）。</div>																								
			默認：0101x000 (x: 不關心，寫 0)																								
0x02	SILENCE_LEN	RW	總線在數據包結束後繼續保持 SILENCE 位長的時間為 1，總線便進入空閒模式，默認 20 (bits)																								
0x03	TX_DELAY	RW	當總線進入空閒並保持此段時間，才允許發送，默認 10 (bits) 可以為越高優先級節點設置越低的值，但至少要保留 1 bit, 以確保所有節點都有足夠時間檢測到總線 IDLE 狀態。																								
0x04	SELF_ID	RW	僅用做接收過濾：（由上至下進行匹配）																								
			<table> <tr> <th>FROM_ID</th><th>TO_ID</th><th>SELF_ID</th><th>接收或丟棄</th></tr> <tr> <td>not care</td><td>not care</td><td>255</td><td>接收（嗅探模式）</td></tr> <tr> <td>= SELF_ID</td><td>not care</td><td>!= 255</td><td>丟棄（避免環路）</td></tr> <tr> <td>!= SELF_ID</td><td>255</td><td>not care</td><td>接收（廣播）</td></tr> <tr> <td>!= SELF_ID</td><td>!= 255</td><td>= TO_ID</td><td>接收（點對點）</td></tr> <tr> <td>not care</td><td>!= 255</td><td>!= TO_ID</td><td>丟棄</td></tr> </table>	FROM_ID	TO_ID	SELF_ID	接收或丟棄	not care	not care	255	接收（嗅探模式）	= SELF_ID	not care	!= 255	丟棄（避免環路）	!= SELF_ID	255	not care	接收（廣播）	!= SELF_ID	!= 255	= TO_ID	接收（點對點）	not care	!= 255	!= TO_ID	丟棄
FROM_ID	TO_ID	SELF_ID	接收或丟棄																								
not care	not care	255	接收（嗅探模式）																								
= SELF_ID	not care	!= 255	丟棄（避免環路）																								
!= SELF_ID	255	not care	接收（廣播）																								
!= SELF_ID	!= 255	= TO_ID	接收（點對點）																								
not care	!= 255	!= TO_ID	丟棄																								
			Default: 255																								

0x05	PERIOD_LS_L	RW	<p>PERIOD_LS 低 8 位 (LS: Low Speed) 為 SILENCE、TX_DELAY 和 FROM_ID 字段設置波特率（也包括 EN_ADVANCE）。</p> <p>計算公式 $factor = sysclock \div bond_rate - 1$ 舉例：如果系統時鐘是 27 MHz，設置 233 最接近 115200 bps（默認 233）</p>
0x06	PERIOD_LS_H	RW	PERIOD_LS 高 8 位，共 16 位（默認 0）
0x07	PERIOD_HS_L	RW	<p>PERIOD_HS 低 8 位 (LS: Low Speed, 默認 233) 為 TO_ID、DATA_LEN、DATA 和 CRC_L/H 字段設置波特率。</p>
0x08	PERIOD_HS_H	RW	PERIOD_HS 高 8 位，共 16 位（默認 0）
0x09	INT_FLAG	R	<p>中斷標誌：</p> <hr/> <p>bit0 BUS_IDLE 指示總線是否進入 IDLE 模式。</p> <hr/> <p>bit1 RX_PENDING 指示 RX 緩存是否有頁待讀。 寫 1 到 RX_CTRL[CLR_RX_PENDING] 清除當前頁待讀標誌。</p> <hr/> <p>bit2 RX_LOST 當一個包正確抵達且不被過濾，但卻因為沒有更多頁用做下一次接收而被丟棄，此標誌置位。 寫 1 到 RX_CTRL[CLR_RX_LOST] 清此標誌。</p> <hr/> <p>bit3 RX_ERROR 當一個不被過濾的包停止位錯誤、超時或校驗錯誤，此標誌置位。 寫 1 到 RX_CTRL[CLR_RX_ERROR] 清此標誌。</p> <hr/> <p>bit4 TX_BUF_CLEAN 指示是否所有 TX 緩存頁都未標誌為待發送。</p> <hr/> <p>bit5 TX_CD 當檢測到有更高優先級節點時推後發送並置此標誌。 寫 1 到 TX_CTRL[CLR_TX_CD] 清此標誌。 此位用作調試使用。</p>
0x0A	INT_MASK	RW	<p>中斷允許 當 INT_FLAG & INT_MASK != 0 時 INT_N 輸出低，否則輸出高阻（默認 0x00）</p>

0x0B	RX	R	<p>讀 RX 緩存頁數據，地址自動增加 共有 8 個 RX 緩存頁，每一頁 256 字節。 當硬件端成功接收到不被過濾的包：如果下一頁未標誌為待讀（可用作下一次接收），將當前頁標誌為待讀並切換到下一頁；否則丟棄該包並置位 RX_LOST。 RX_PENDING 位指示用戶端當前頁待讀，寫 1 到 CLR_RX_PENDING 清除當前頁的待讀標誌並切換到下一頁。寫 1 到 RST_RX 清除所有頁的待讀標誌，並復位接收邏輯。</p>														
0x0C	TX	W	<p>寫 TX 緩存頁數據，地址自動增加 共有 2 個 TX 緩存頁，每一頁 256 字節。 當用戶寫完數據，需要等待 TX_BUF_CLEAN 置位，然後才可以通過 START_TX 置位當前頁的待發送標誌，並自動切換到下一頁（否則什麼都不會發生）。 當頁的待發送標誌被置上，硬件將會啓動發送，當讀完發送所需數據時，硬件端清頁的待發送標誌並切換到下一頁。</p>														
0x0D	RX_CTRL	W	<p>RX 控制:</p> <table><tr><td>bit0</td><td>RST_RX_POINTER</td></tr><tr><td></td><td>寫 1 歸零當前 RX 緩存頁的讀指針</td></tr><tr><td>bit1</td><td>CLR_RX_PENDING (自動包含 bit0)</td></tr><tr><td>bit2</td><td>CLR_RX_LOST</td></tr><tr><td>bit3</td><td>CLR_RX_ERROR</td></tr><tr><td>bit4</td><td>RST_RX (自動包含 bit0, 2, 3)</td></tr></table>	bit0	RST_RX_POINTER		寫 1 歸零當前 RX 緩存頁的讀指針	bit1	CLR_RX_PENDING (自動包含 bit0)	bit2	CLR_RX_LOST	bit3	CLR_RX_ERROR	bit4	RST_RX (自動包含 bit0, 2, 3)		
bit0	RST_RX_POINTER																
	寫 1 歸零當前 RX 緩存頁的讀指針																
bit1	CLR_RX_PENDING (自動包含 bit0)																
bit2	CLR_RX_LOST																
bit3	CLR_RX_ERROR																
bit4	RST_RX (自動包含 bit0, 2, 3)																
0x0E	TX_CTRL	W	<p>TX 控制:</p> <table><tr><td>bit0</td><td>RST_TX_POINTER</td></tr><tr><td></td><td>寫 1 歸零當前 TX 緩存頁的寫指針</td></tr><tr><td>bit1</td><td>START_TX (自動包含 bit0)</td></tr><tr><td>bit2</td><td>SET_TX_BUF_CLEAN_MASK</td></tr><tr><td></td><td>Set INT_MASK[TX_BUF_CLEAN] bit</td></tr><tr><td></td><td>你可以在置位 START_TX 時同時置此位，當頁的待發送標誌被清除時，便會來 TX_BUF_CLEAN 中斷。</td></tr><tr><td>bit3</td><td>CLR_TX_CD</td></tr></table>	bit0	RST_TX_POINTER		寫 1 歸零當前 TX 緩存頁的寫指針	bit1	START_TX (自動包含 bit0)	bit2	SET_TX_BUF_CLEAN_MASK		Set INT_MASK[TX_BUF_CLEAN] bit		你可以在置位 START_TX 時同時置此位，當頁的待發送標誌被清除時，便會來 TX_BUF_CLEAN 中斷。	bit3	CLR_TX_CD
bit0	RST_TX_POINTER																
	寫 1 歸零當前 TX 緩存頁的寫指針																
bit1	START_TX (自動包含 bit0)																
bit2	SET_TX_BUF_CLEAN_MASK																
	Set INT_MASK[TX_BUF_CLEAN] bit																
	你可以在置位 START_TX 時同時置此位，當頁的待發送標誌被清除時，便會來 TX_BUF_CLEAN 中斷。																
bit3	CLR_TX_CD																
0x0F	RX_ADDR	RW	讀寫當前 RX 緩存頁的讀指針														
0x10	RX_PAGE_FLAG	R	<p>（僅 NO_DROP 置位時使用） 0 代表當前 RX 緩存頁中的數據包正確； 非 0 表示數據包錯誤，其值指示最後接收到的字節地址，包含 CRC 字段。</p>														

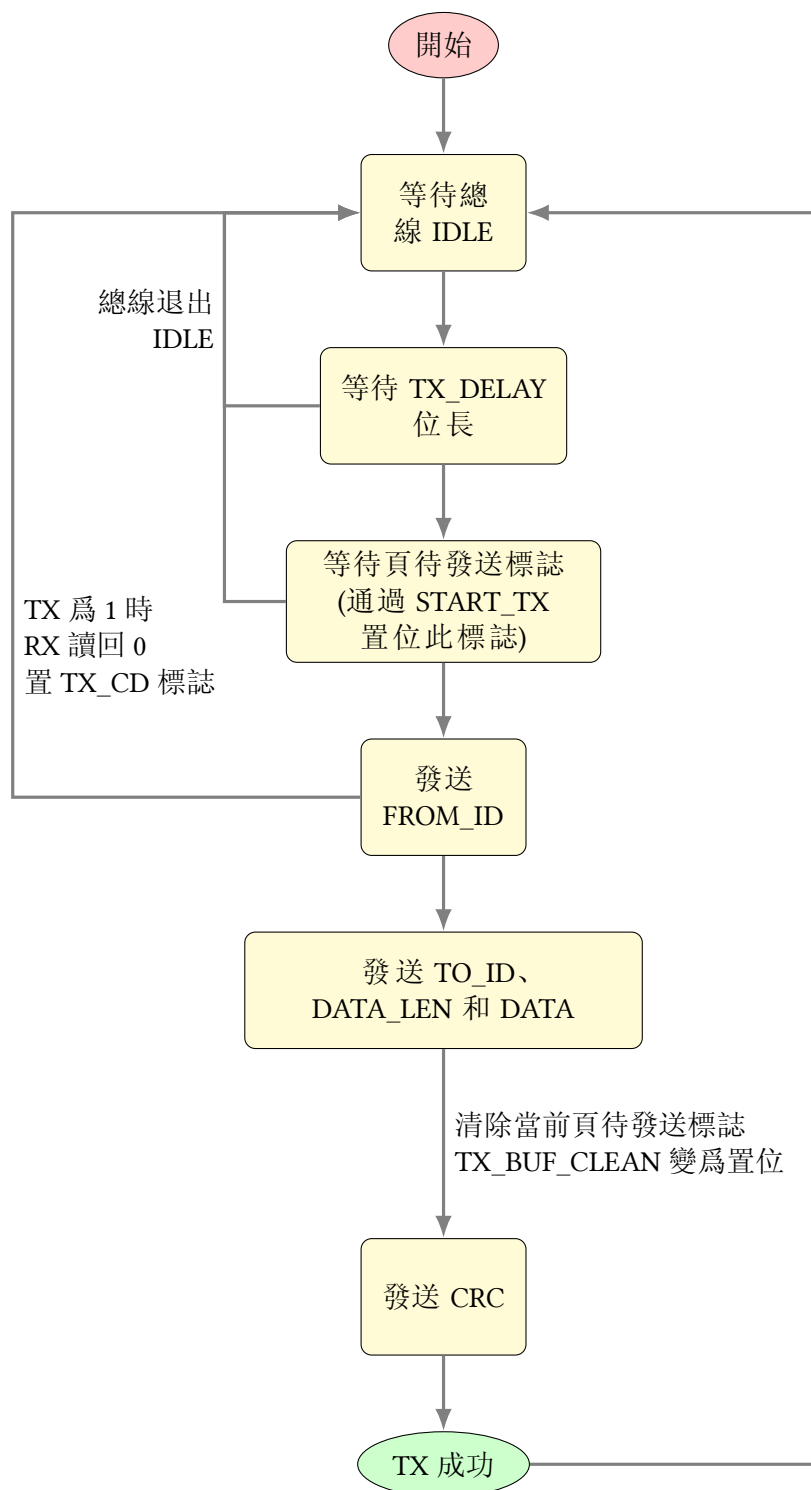
5 流程圖

5.1 RX



如果當前包接收不夠兩個字節，或者將會被丟棄則不會設置 RX_ERROR 標誌。

5.2 TX



6 設備接口

SPI 和 I2C 頻率最好低於 $\text{sysclock} \div 10$.

除了 RX 和 TX, 其餘寄存器通常只讀寫 1 字節。

6.1 SPI

讀寫：

```
start (NSS = 0)
Write reg address with bit7: 0: read, 1: write
Read or write arbitrary length of data
stop (NSS = 1)
```

6.2 I2C

寫地址: 0xc0

讀地址: 0xc1

寫：

```
start
write the write address
write 1 byte reg address
write arbitrary length of data
stop
```

讀：

```
start
write the write address
write 1 byte reg address
restart (or stop + start)
write the read address
read arbitrary length of data, ACK all bytes except last byte
stop
```

7 操作示例

7.1 初始化

```
// select system clock, enable OUTPUT
CD485_write(REG_SETTING, F27M | OUTPUT_EN);

// set SELF_ID
CD485_write(REG_SELF_ID, 0xcd);

// set bondrate, PERIOD_XX_H default 0
CD485_write(REG_PERIOD_LS_L, 35); // 750000 bps
CD485_write(REG_PERIOD_HS_L, 2); // 9 Mbps

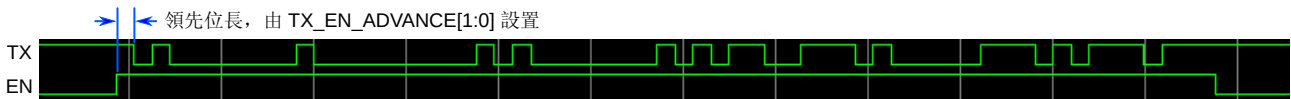
// clean RX buffer
CD485_write(REG_RX_CTRL, RST_RX);
```

```
// enable interrupt
CD485_write(REG_INT_MASK, RX_ERROR | RX_LOST | RX_PENDING);
```

7.1.1 兼容模式和傳統模式

PERIOD_LS 和 PERIOD_HS 設置相同為兼容模式。

進一步置位 NO_ARBITRATE 進入傳統模式：



7.2 TX

```
header_buf[0] = 0xcd; // FROM_ID
header_buf[1] = 0x02; // TO_ID
header_buf[2] = 12;   // DATA_LEN

CD485_write_chunk(REG_TX, header_buf, 3);           // write HEADER
CD485_write_chunk(REG_TX, data_buf, header_buf[2]); // write DATA

while (CD485_read(REG_INT_FLAG) & TX_BUF_CLEAN == 0); // make sure TX_BUF_CLEAN is set
// sent packet, and enable TX_BUF_CLEAN interrupt
CD485_write(REG_TX_CTRL, SET_TX_BUF_CLEAN_MASK | START_TX);

// write next packet
// send next packet when the TX_BUF_CLEAN interrupt occur

// if no further packet need send, disable TX_BUF_CLEAN interrupt:
CD485_write(REG_INT_MASK, CD485_read(REG_INT_MASK) & ~TX_BUF_CLEAN);
```

7.3 RX

```
// when RX_PENDING interrupt occur:

CD485_read_chunk(REG_RX, header_buf, 3);           // read HEADER
CD485_read_chunk(REG_RX, data_buf, header_buf[2]); // read DATA

CD485_write(REG_RX_CTRL, CLR_RX_PENDING);          // release page
```