6. STM32 串口通信

北京科技大学
计算机与通信工程学院

串行通信的初步认识

- 数据通信时，可以将通信方式分为两种：
  1. 并行通信：通信时数据的各个位同时传送，可以实现字节为单位通信，但是通信线多占用资源多，成本高。如：1110 0100 一次性通过8条线发送
  2. 串行通信：一次只能发送一位，要发送8次才能发送一个字节。

串行数据发送示意图

常用的串行通信协议有：

- 通用异步收发传输器（Universal Asynchronous Receiver and Transmitter，UART）- 即串口通信
- 串行外设接口（Serial Peripheral Interface， SPI）
- I2C总线接口 (Inter - Integrated Circuit，I2C)

通信最核心的问题

- 发送端发送的数据在接收端正确地接收。

- 0 -----〉 0 1 -----〉 1
- 0 --X--〉 1 1 --X--〉 11
- 00 --X--〉 0 11 --X--〉 1

串行同步通信和串行异步通信

- 串行通信分为同步通信和异步通信

（1）串行同步通信

- 同步通信中，所有设备使用一个共同的时钟信号，发送和接收双方严格按照该时钟信号处理数据的发送和接收。同步通信的优点是数据传输速率高，缺点是要求发送时钟和接收时钟保持严格同步。 SPI和I2C属于串行同步通信。

（2）串行异步通信

- 异步通信中，每个设备都有自己的时钟信号，通信双方的时钟频率保持一致。异步通信以字符为单位进行数据传送，每一个字符均按照固定的格式传送，被称为帧，即串行异步通信一次传送一个帧。UART属于串行异步通信。

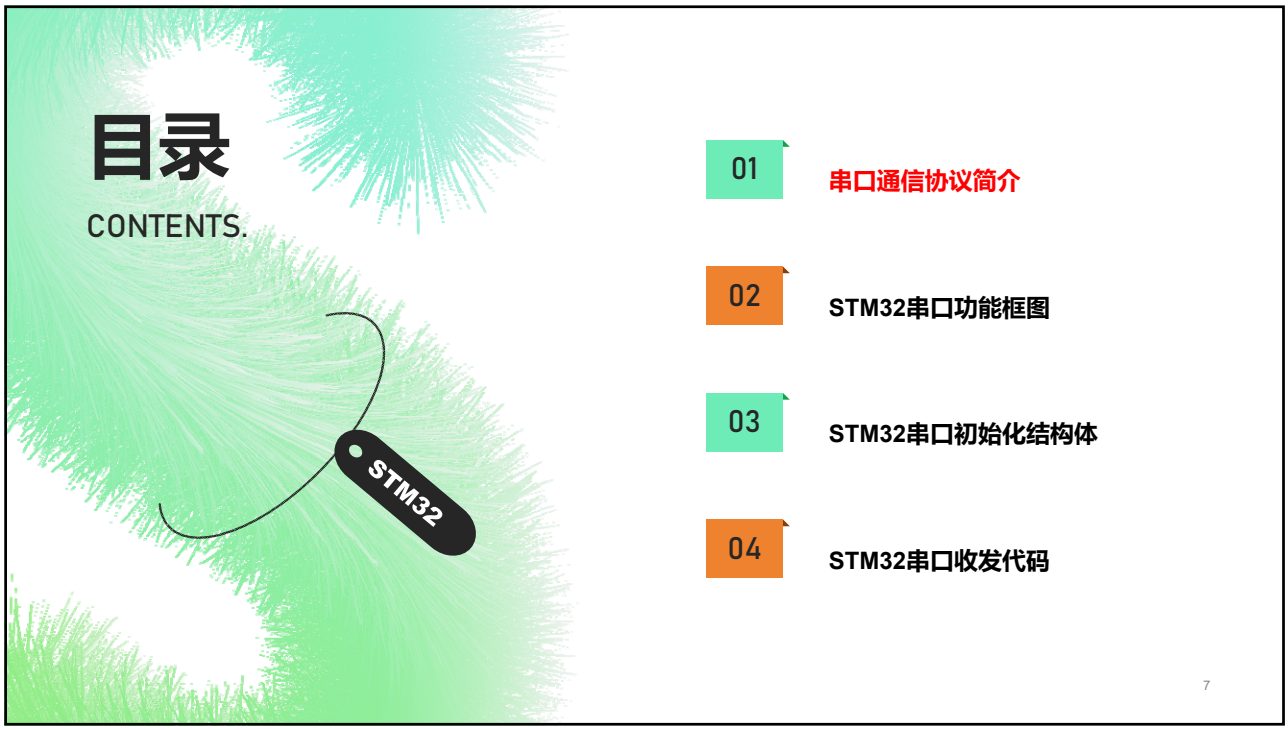
•ASCII码（American Standard Code for Information Interchange，即美国信息互换标准代码）

表 11-3　ASCII 码字符表

ASC	控制	ASCII	字符	ASCII	字符	ASCII	字符
000	NUL	032	(space)	064	@	096	'
001	SOH	033	!	065	A	097	a
002	STX	034	"	066	B	098	b
003	ETX	035	#	067	C	099	c
004	EOT	036	$	068	D	100	d
005	END	037	%	069	E	101	e
006	ACK	038	&	070	F	102	f
007	BEL	039	'	071	G	103	g
008	BS	040	(	072	H	104	h
009	HT	041	)	073	I	105	i
010	LF	042	*	074	J	106	j
011	VT	043	+	075	K	107	k
012	FF	044	,	076	L	108	l
013	CR	045	-	077	M	109	m
014	SO	046	.	078	N	110	n
015	SI	047	/	079	O	111	o
016	DLE	048	0	080	P	112	p
017	DC1	049	1	081	Q	113	q
018	DC2	050	2	082	R	114	r
019	DC3	051	3	083	S	115	s
020	DC4	052	4	084	T	116	t
021	NAK	053	5	085	U	117	u
022	SYN	054	6	086	V	118	v
023	ETB	055	7	087	W	119	w
024	CAN	056	8	088	X	120	x
025	EM	057	9	089	Y	121	y
026	SUB	058	:	090	Z	122	z
027	ESC	059	;	091	[	123	{
028	FS	060	<	092	\	124	|
029	GS	061	=	093	]	125	}
030	RS	062	>	094	^	126	~
031	US	063	?	095		127	DEL

逻辑分析仪对串行通信信号的抓取

图 11-7　逻辑分析仪串口数据示意图



串口通信协议简介

物理层：规定通讯系统中具有机械、电子功能部分的特性，确保原始数据在物理媒体的传输。其实就是硬件部分。

协议层：协议层主要规定通讯逻辑，统一收发双方的数据打包、解包标准。其实就是软件部分。

简单来说物理层规定我们用嘴巴还是用肢体来交流，协议层则规定我们用中文还是英文来交流。

串口通信协议简介

1-RS232标准

2-USB转串口

3-原生的串口到串口

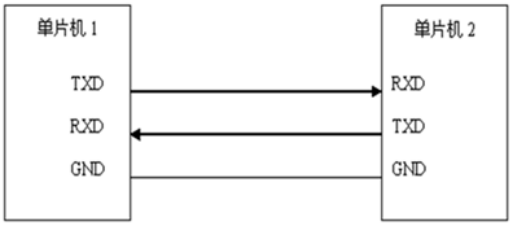
串口通信及其连接方式

串口通信，Universal Asynchronous Receiver/Transmitter
串口通信连接包括两线，发送和接收双方交叉连接：

- 一个RXD（Receive Data， RXD），表示接收数据
- 一个TXD（Transmit Data，TXD），表示发送数据。

另外，如果用到流控制，还需要

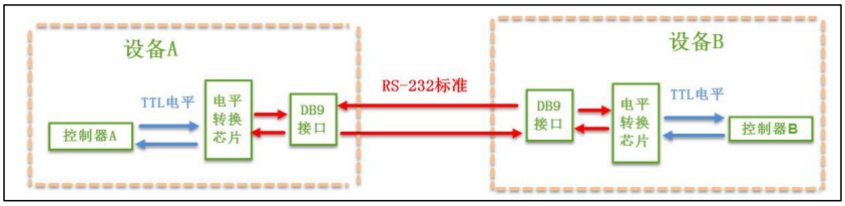
- RTS（Request To Send，RTS）表示请求发送，
- CTS（Clear To Send，CTS）表示清除发送。



单片机之间 UART 通信示意图

串口通信协议简介

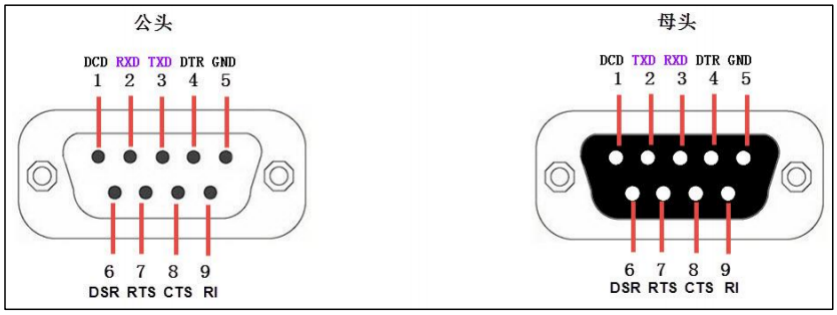
RS232标准串口通讯结构图



1、RS232标准串口主要用于工业设备直接通信

2、电平转换芯片一般有MAX3232，SP3232

串口通信协议简介



DB9 标准的公头及母头接法

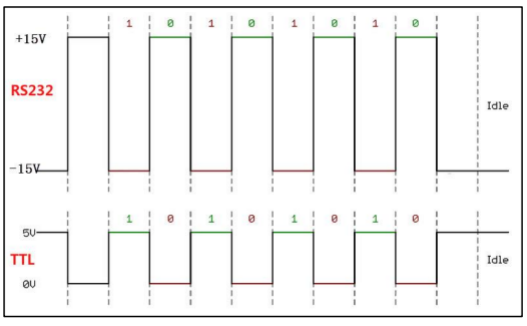
串口通信协议简介

串口线

DB9串口线

USB转串口线

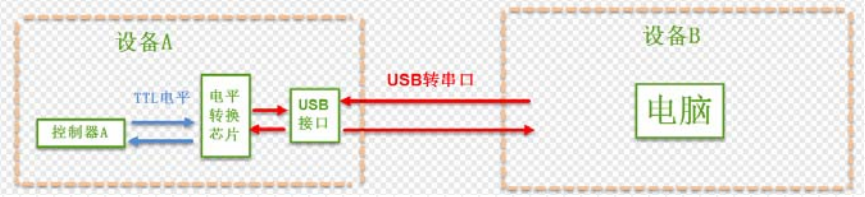
串口通信协议简介



RS-232 与 TTL 电平区别

串口通信协议简介

USB转串口通讯结构图



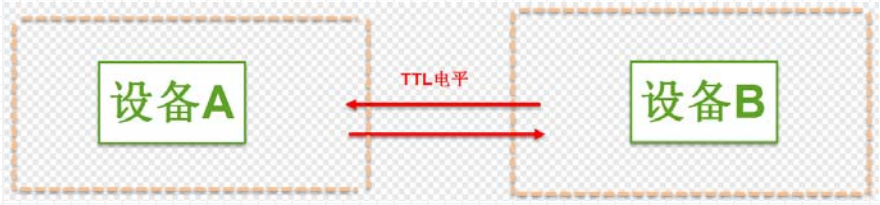
1、USB转串口主要用于设备跟电脑通信

2、电平转换芯片一般有CH340、PL2303、CP2102、FT232

3、使用的时候电脑端需要安装电平转换芯片的驱动

串口通信协议简介

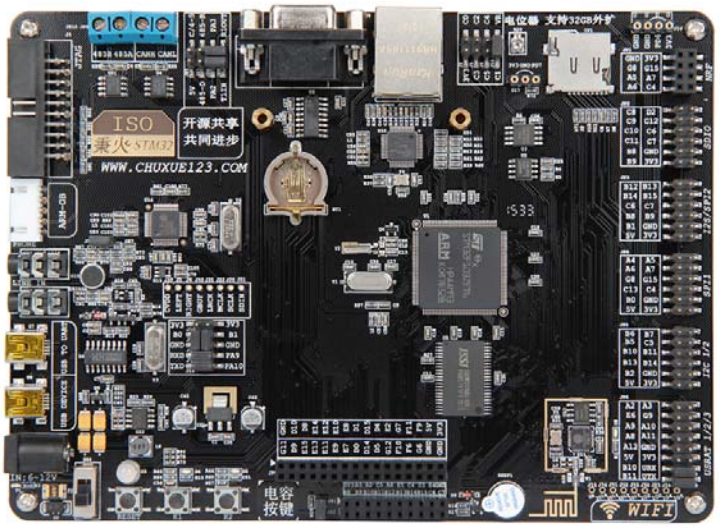
原生的串口到串口



1、原生的串口通信主要是控制器跟串口的设备或者传感器通信，不需要经过电平转换芯片来转换电平，直接就用TTL电平通信

2、GPS模块、GSM模块、串口转WIFI模块、HC04蓝牙模块

串口通信协议简介



串行异步通信数据帧

串口通信的每一帧数据由起始位（低电平）、数据位、奇偶校验位（可选）、停止位（高电平）组成。
UART 模式提供全双工异步传送，接收器中的位同步不影响发送功能。一个 UART 字节包括 1 个起始位、8 个数据位、1 个作为可选项的第 9 位数据或奇偶校验位，再加上 1 个（或 2 个）停止位。注意：虽然真实的数据包括 8位或 9 位，但是，数据传送只涉及一个字节。

图 异步通信数据帧格式

串口通信协议简介

串口数据包的基本组成

串口通信协议简介

起始位：由1个逻辑 0 的数据位表示

结束位：由 0.5、 1、 1.5 或 2 个逻辑 1 的数据位表示

有效数据：在起始位后紧接着的就是有效数据，有效数据的长度常被约定为 5、 6、 7 或 8 位长

串口通信协议简介

校验位：可选，为的是数据的抗干扰性。

校验方法分为：

1-奇校验(odd)、 2-偶校验(even)

3-0 校验(space)、 4-校验(mark)

5-无校验(noparity)

串口通信协议简介

奇校验(odd)：有效数据和校验位中“1”的个数为奇数

比如一个 8 位长的有效数据为：01101001，此时总共有 4 个“1”，为达到奇校验效果，校验位为“1”，最后传输的数据将是 8 位的有效数据加上 1 位的校验位总共 9 位

串口通信协议简介

偶校验(even)：有效数据和校验位中“1”的个数为偶数

比如一个 8 位长的有效数据为：01101001，此时总共有 4 个“1”，为达到偶校验效果，校验位为“0”，最后传输的数据将是 8 位的有效数据加上 1 位的校验位总共 9 位

串口通信协议简介

0 校验是不管有效数据中的内容是什么，校验位总为“0”。

1 校验是校验位总为“1”。

无校验就是数据包中不包含校验位。

目录

CONTENTS.

STM32-5

25

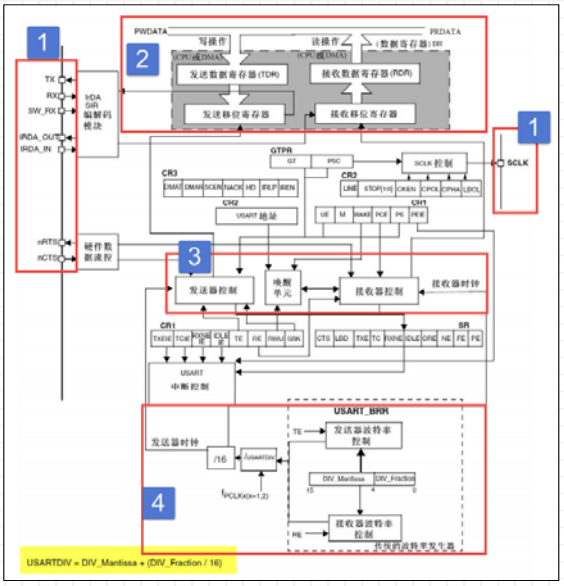
串口功能框图讲解

1-引脚

2-数据寄存器

3-控制器

4-波特率



串口功能框图讲解

表 21-3 STM32F103ZET6 芯片的 USART 引脚

引脚	APB2 总线	APB1 总线			
	USART1	USART2	USART3	UART4	UART5
TX	PA9	PA2	PB10	PC10	PC12
RX	PA10	PA3	PB11	PC11	PD2
SCLK	PA8	PA4	PB12		
nCTS	PA11	PA0	PB13		
nRTS	PA12	PA1	PB14		

表 21-3 STM32F103VET6 芯片的 USART 引脚

引脚	APB2 总线	APB1 总线			
	USART1	USART2	USART3	UART4	UART5
TX	PA9	PA2	PB10	PC10	PC12
RX	PA10	PA3	PB11	PC11	PD2
SCLK	PA8	PA4	PB12		
nCTS	PA11	PA0	PB13		
nRTS	PA12	PA1	PB14		

STM32F10x数据手册—Pinouts and pin description。

ST每个系列的芯片都有一个数据手册，里面有引脚的详细功能。

串口功能框图讲解

TX： 数据发送

RX： 是数据接收

SCLK： 时钟，仅同步通信时使用

nRTS： 请求发送(Request To Send)

nCTS： 允许发送(Clear To Send)

串口功能框图讲解

数据寄存器—USART_DR：9位有效，包含一个发送数据寄存器TDR和一个接收数据寄存器RDR。一个地址对应了两个物理内存。

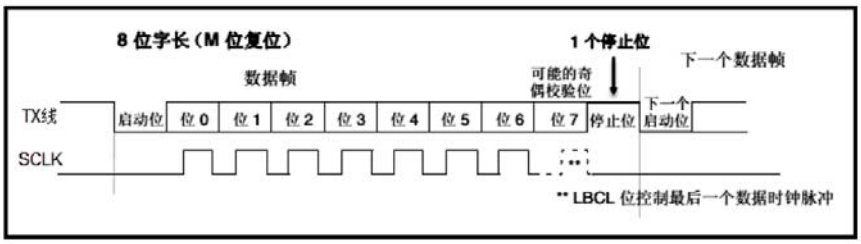
串口功能框图讲解

数据发送

数据接收

具体流程?

串口功能框图讲解



USART_CR1：M位，0：8bit，1：9bit

USART_CR2：STOP

USART_CR1：PCE、PS、PEIE

USART_SR ：PE

USART_CR1寄存器上的PCE位，使能奇偶控制

M位，定义帧长度

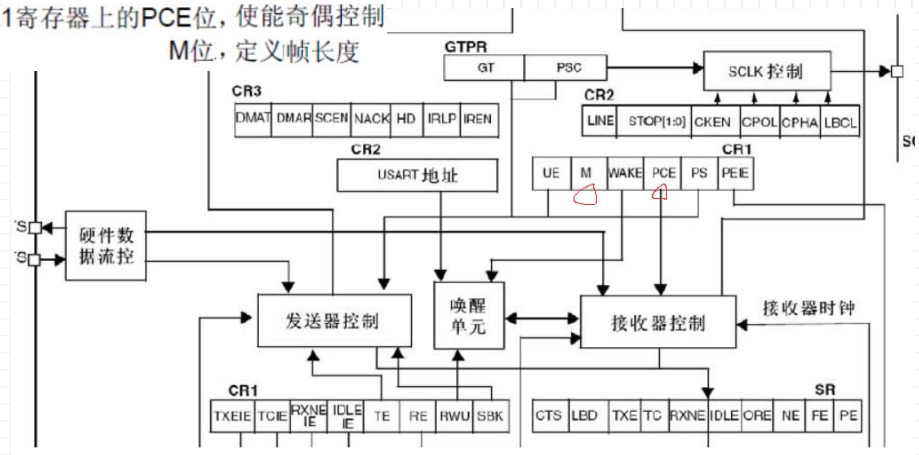


表179 帧格式

M位	PCE位	USART帧
0	0	| 起始位 | 8位数据 | 停止位 |
0	1	| 起始位 | 7位数据 | 奇偶检验位 | 停止位 |
1	0	| 起始位 | 9位数据 | 停止位 |
1	1	| 起始位 | 8位数据 | 奇偶检验位 | 停止位 |

32

串口功能框图讲解

USART_CR1：UE、TE、RE

USART_SR：TXE，Transmit data register empty

USART_CR1：TXEIE

USART_SR：TC，Transmission complete

USART_CR1：TCIE

USART_SR：RXNE，Read data register not empty

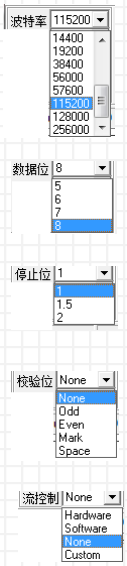
USART_CR1：RXNEIE

控制器—USART_CR1、CR2、CR3

熟读手册即可

1. 串口号
2. 波特率
3. 数据位
4. 停止位
5. 校验位
6. 流控制

例如：
COM6,115200,8,1,
N,N



波特率

- 串口通信双方的波特率必须一致
- 波特率 vs 比特率
  - 波特率表示每秒钟传送的码元符号的个数，它用单位时间内载波调制状态改变的次数来表示。单位：码元/s，所以它与比特率是不同的概念
  - 比特率表示每秒钟通过信道传输的信息量，也就是每秒钟传送的二进制位数。 单位：bit/s、比特/秒
  - 不同的调制方法可在一个码元上负载多个比特信息。比特率=波特率*单个调制状态对应的二进制位数
  - 对于二进制的信号，码元速率和信息速率在数值上是相等的。 因此，对于串口来说，比特率=波特率
  - 但要注意：这里的比特率并不是应用层的有效数据传输率，因为开始位、停止位、校验位占了开销。

串口功能框图讲解

波特率—每秒钟要发送多少数据

USART_BRR：波特率寄存器

USART_CR1：OVER8

串口功能框图讲解

$$\text{Tx / Rx 波特率} = \frac{f_{CK}}{(16 * USARTDIV)}$$

USARTDIV：无符号的定点数

FCK：串口的时钟，注区分APB2和APB1两条总线

串口功能框图讲解

USART：USART1，时钟为72M

波特率：115200

$$115200 = \frac{72000000}{16 * USARTDIV}$$

解得 USARTDIV=39.0625，可算得 DIV_Fraction=0.0625*16=1=0x01，DIV_Mantissa=39=0x17，即应该设置 USART_BRR 的值为 0x171。

目录

CONTENTS.

STM32-5

40

初始化结构体讲解

USART初始化结构体

```
typedef struct
{
  uint32_t USART_BaudRate;        //波特率 BRR
  uint16_t USART_WordLength;      //字长 CR1_M
  uint16_t USART_StopBits;        //停止位 CR2_STOP
  uint16_t USART_Parity;          //校验控制 CR1_PCE、CR1_PS
  uint16_t USART_Mode;            //模式选择CR1_TE、CR1_RE
  // 硬件流选择 CR3_CTSE、CR3_RTSE
  uint16_t USART_HardwareFlowControl;
} USART_InitTypeDef;
```

初始化结构体讲解

同步时钟初始化结构体

```
typedef struct
{
  uint16_t USART_Clock;       // 同步时钟 CR2_CLKEN
  uint16_t USART_CPOL;        // 极性 CR2_CPOL
  uint16_t USART_CPHA;        // 相位 CR2_CPHA
  uint16_t USART_LastBit;     //最后一个位的时钟脉冲 CR2_LBC
} USART_ClockInitTypeDef;
```

串口资源定义

```
//stm32f10x.h
/  * @brief Universal Synchronous Asynchronous Receiver Transmitter
                UART的寄存器成组封装*/

typedef struct
{
  __IO uint16_t SR;
  uint16_t  RESERVED0;
  __IO uint16_t DR;
  uint16_t  RESERVED1;
  __IO uint16_t BRR;
  uint16_t  RESERVED2;
  __IO uint16_t CR1;
  uint16_t  RESERVED3;
  __IO uint16_t CR2;
  uint16_t  RESERVED4;
  __IO uint16_t CR3;
  uint16_t  RESERVED5;
  __IO uint16_t GTPR;
  uint16_t  RESERVED6;
} USART_TypeDef;
```

```
#define USART1      ((USART_TypeDef *) USART1_BASE)
#define USART2      ((USART_TypeDef *) USART2_BASE)
#define USART3      ((USART_TypeDef *) USART3_BASE)
#define UART4      ((USART_TypeDef *) UART4_BASE)
#define UART5      ((USART_TypeDef *) UART5_BASE)
```

43

编程时需要用到的固件库函数

1-串口初始化函数

void USART_Init

(USART_TypeDef* USARTx, USART_InitTypeDef* USART_InitStruct)

2-中断配置函数

void USART_ITConfig

(USART_TypeDef* USARTx, uint16_t USART_IT,

FunctionalState NewState)

3-串口使能函数

void USART_Cmd(USART_TypeDef* USARTx,

FunctionalState NewState)

编程时需要用到的固件库函数

4-数据发送函数

void USART_SendData
(USART_TypeDef* USARTx, uint16_t Data)

5-数据接收函数

uint16_t USART_ReceiveData(USART_TypeDef* USARTx)

6-中断状态位获取函数

ITStatus USART_GetITStatus
(USART_TypeDef* USARTx, uint16_t USART_IT)

目录

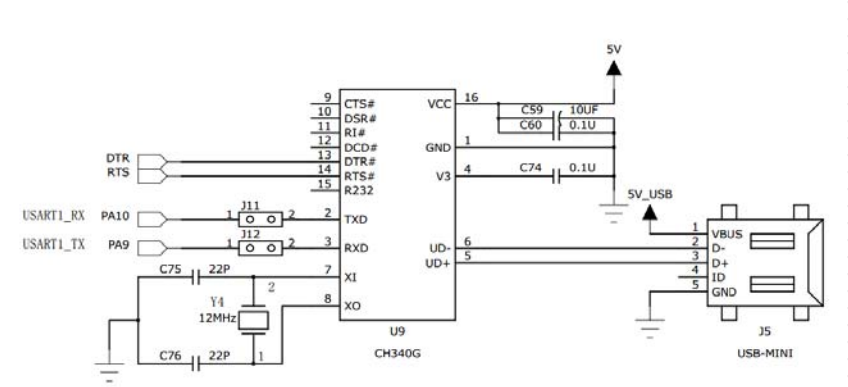
CONTENTS.

STM32-5

46

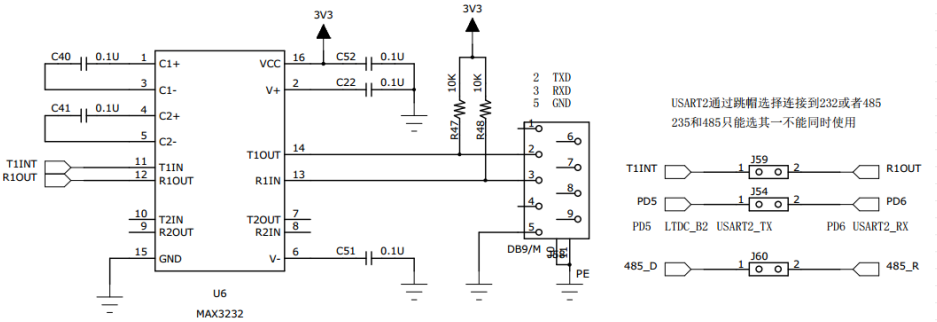
硬件设计

硬件设计-1：TTL电平转USB电平
（使用时电脑上记得安装CH340驱动）



硬件设计

硬件设计-2：TTL电平转232电平



编程要点

- 1-初始化串口需要用到的GPIO
- 2-初始化串口，USART_InitTypeDef
- 3-中断配置（接收中断，中断优先级）
- 4-使能串口
- 5-编写发送和接收函数
- 6-编写中断服务函数

编程的疑问?

1-如何发送16位的数据?

2-如何发送一个数组?

3-如何使用printf()？如何使用scanf()?

4-电脑端发送过来的数据是什么格式的？十进制？十六进制？字符?

Ex1.中断接收和发送

实验-1：

- 单片机给电脑发送数据，电脑上位机把数据打印出来；
- 电脑上位机给单片机发数据，单片机接收到数据之后立马发回给电脑，并打印出来。

主函数：
首先，调用 USART_Config 函数完成 USART 初始化配置，包括：
GPIO 配置， USART 配置，接收中断使能。
然后，调用字符发送函数把数据发送给串口调试助手了。
最后，主函数什么都不做，只是静静地等待USART 接收中断的产生，并在中断服务函数把数据回传。

```
int main(void) {
    /*初始化 USART 配置模式为 115200 8-N-1，中断接收*/
    USART_Config();

    Usart_SendString( DEBUG_USARTx,"这是一个串口中断接收回显实验\n");

    while (1) { }
}
```

Ex1.中断接收和发送

```
/* 串口宏定义，不同的串口挂载的总线和 IO 不一样，移植时需要修改这几个宏*/

// 串口 1-USART1
#define  DEBUG_USARTx                   USART1
#define  DEBUG_USART_CLK                RCC_APB2Periph_USART1
#define  DEBUG_USART_APBxClkCmd         RCC_APB2PeriphClockCmd
#define  DEBUG_USART_BAUDRATE           115200

// USART GPIO 引脚宏定义
#define  DEBUG_USART_GPIO_CLK           (RCC_APB2Periph_GPIOA)
#define  DEBUG_USART_GPIO_APBxClkCmd    RCC_APB2PeriphClockCmd

#define  DEBUG_USART_TX_GPIO_PORT       GPIOA
#define  DEBUG_USART_TX_GPIO_PIN        GPIO_Pin_9
#define  DEBUG_USART_RX_GPIO_PORT       GPIOA
#define  DEBUG_USART_RX_GPIO_PIN        GPIO_Pin_10
```

Ex1.中断接收和发送

USART 初始化配置

```
void USART_Config(void)
{
  GPIO_InitTypeDef GPIO_InitStructure;
  USART_InitTypeDef USART_InitStructure;

  // 打开串口 GPIO 的时钟
  DEBUG_USART_GPIO_APBxClkCmd(DEBUG_USART_GPIO_CLK, ENABLE);

  // 打开串口外设的时钟
  DEBUG_USART_APBxClkCmd(DEBUG_USART_CLK, ENABLE);

  // 将 USART Tx 的GPIO 配置为推挽复用模式
  GPIO_InitStructure.GPIO_Pin = DEBUG_USART_TX_GPIO_PIN;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
  GPIO_Init(DEBUG_USART_TX_GPIO_PORT, &GPIO_InitStructure);

  // 将 USART Rx 的GPIO 配置为浮空输入模式
  GPIO_InitStructure.GPIO_Pin = DEBUG_USART_RX_GPIO_PIN;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;
  GPIO_Init(DEBUG_USART_RX_GPIO_PORT, &GPIO_InitStructure);
```

Ex1.中断接收和发送

USART 初始化配置

```
  // 配置串口的工作参数
  // 配置波特率
  USART_InitStructure.USART_BaudRate = DEBUG_USART_BAUDRATE;
  // 配置 针数据字长
  USART_InitStructure.USART_WordLength = USART_WordLength_8b;
  // 配置停止位
  USART_InitStructure.USART_StopBits = USART_StopBits_1;
  // 配置校验位
  USART_InitStructure.USART_Parity = USART_Parity_No ;
  // 配置硬件流控制
  USART_InitStructure.USART_HardwareFlowControl =   USART_HardwareFlowControl_None;
  // 配置工作模式，收发一起
  USART_InitStructure.USART_Mode = USART_Mode_Rx | USART_Mode_Tx;
  // 完成串口的初始化配置
  USART_Init(DEBUG_USARTx, &USART_InitStructure);

  // 串口中断优先级配置
  NVIC_Configuration();

  // 使能串口接收中断
  USART_ITConfig(DEBUG_USARTx, USART_IT_RXNE, ENABLE);

  // 使能串口
  USART_Cmd(DEBUG_USARTx, ENABLE);
}
```

Ex1.中断接收和发送

配置USART 作为中断源

```
static void NVIC_Configuration(void)
{
    NVIC_InitTypeDef NVIC_InitStructure;

    /* 嵌套向量中断控制器组选择 */
    NVIC_PriorityGroupConfig(NVIC_PriorityGroup_2);

    /* 配置 USART 为中断源 */
    NVIC_InitStructure.NVIC_IRQChannel = USART1_IRQn;

   /* 抢断优先级为 1 */
    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 1;
    /* 子优先级为 1 */
    NVIC_InitStructure.NVIC_IRQChannelSubPriority = 1;
    /* 使能中断 */
    NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
    /* 初始化配置 NVIC */
    NVIC_Init(&NVIC_InitStructure);
}
```

Ex1.中断接收和发送

字符发送函数

```
/****************** 发送一个字符 **********************/
void Usart_SendByte( USART_TypeDef * pUSARTx, uint8_t ch)
{
/* 发送一个字节数据到 USART */
    USART_SendData(pUSARTx,ch);

    /* 等待发送数据寄存器为空 */
    while (USART_GetFlagStatus(pUSARTx, USART_FLAG_TXE) == RESET);
}

/****************** 发送字符串 **********************/
void Usart_SendString( USART_TypeDef * pUSARTx, char *str)
{
    unsigned int k=0;
    do {
        Usart_SendByte( pUSARTx, *(str + k) );
        k++;
    } while (*(str + k)!='\0');

    /* 等待发送完成 */
    while (USART_GetFlagStatus(pUSARTx,USART_FLAG_TC)==RESET) {
    }
}
```

TXE仅反映TDR状态，与移位寄存器无关。即使TDR为空，移位寄存器可能仍在发送数据。
TC直接关联移位寄存器，需确保所有数据位（含停止位）已物理发送至总线。

多字节发送时，通常在循环结束后等待TC=1以确保所有字节发送完成

25.6.2 数据寄存器(USART_DR)

地址偏移：0x04

复位值：不确定

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
保留															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
保留							DR[8:0]								
							rw	rw	rw	rw	rw	rw	rw	rw	rw

位31:9	保留位，硬件强制为0
位8:0	**DR[8:0]**：数据值 (Data value) 包含了发送或接收的数据。由于它是由两个寄存器组成的，一个给发送用(TDR)，一个给接收用(RDR)，该寄存器兼具读和写的功能。TDR寄存器提供了内部总线和输出移位寄存器之间的并行接口(参见图248)。RDR寄存器提供了输入移位寄存器和内部总线之间的并行接口。 当使能校验位(USART_CR1中PCE位被置位)进行发送时，写到MSB的值(根据数据的长度不同，MSB是第7位或者第8位)会被后来的校验位该取代。 当使能校验位进行接收时，读到的MSB位是接收到的校验位。

57

Ex1.中断接收和发送

串口接收中断服务程序，放在 stm32f10x_it.c 文件中

```
void USART1_IRQHandler(void)  {
    uint8_t ucTemp;
    if (USART_GetITStatus(DEBUG_USARTx,USART_IT_RXNE)!=RESET) {
       ucTemp = USART_ReceiveData( DEBUG_USARTx );
       USART_SendData(USARTx,ucTemp);    }
}
```

主函数：首先调用 USART_Config 函数完成 USART 初始化配置，包括 GPIO 配置，USART 配置，接收中断使能等等信息。
然后调用字符发送函数把数据发送给串口调试助手了。
最后主函数什么都不做，只是静静地等待USART 接收中断的产生，并在中断服务函数把数据回传。

```
int main(void) {
    /*初始化 USART 配置模式为 115200 8-N-1，中断接收*/
    USART_Config();

    Usart_SendString( DEBUG_USARTx,"这是一个串口中断接收回显实验\n");

    while (1) { }
}
```