UART IP core 的使用说明

UART(Universal Asynchronous Receiver Transmitter--通用异步收发器)是广泛使用的串行数据传输协议　它的功能是将并行的数据转变为串行的数据发送或者将接受的串行数据变为并行数据。配合电平转换环节，UART 可以在较长的距离上实现全双工的串行通信。

UART 产生于 70 年代，Intel 8250 是第一代产品，UART 中比较著名的是 16550。后来 UART 中增加了硬件缓存　其代表性产品是 I6550，在功能上比 8250 多 16 字节的接收、发送 FIFO。

1. UART 的通信模式、数据格式和流控制

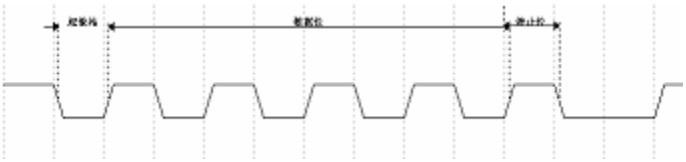
通信模式

UART 的通信模式分为三种，它们是：单工、半双工和全双工：

- 如果在通信过程的任意时刻，信息只能由一方 A 传到另一方 B，则称为单工。
- 如果在任意时刻，信息既可由 A 传到 8，又能由 8 传 A，但只能由一个方向上的传输存在，称为半双工传输。
- 如果在任意时刻，线路上存在 A 到 B 和 B 到 A 的双向信号传输，则称为全双工。

单工	半双工	全双工
A ------ B	A ------ B	A ------ B
--------→	←--------→	----------→ ←----------

数据格式



UART 的串行数据格式

所有的 UART 遵守相同的串行数据格式，如图所示。在没有数据的时候，UART 的串行输出是高电平，每一次数据的发送都以一个时钟周期的低电平开始，这个低电平叫做起始位，而时钟叫做通信的波特率　在起始位之后就是连续的串行数据位。数据位的长短是可以变化的，一般是 5—9 位。数据位之后可以直接是停止位，也可以是一个时钟周期的校验位(校验位在图中没有画出来)校验位可以是奇校验、偶校验。采用奇校验时，传送的数据位和校验位中“1”的个数为奇数，如：

- 1(校验位)0110，0101(数据位)
- 0(校验位)0110，0001(数据位)

采用偶校验时，传送的数据位和校验位中“1”的个数为偶数，如：

- 1(校验位)0100，0101(数据位)
- 0(校验位)0100，0001(数据位)

校验位之后是高电平的停止位，用以指示一次发送的完毕。停止位一般是 I 位、I．5 位或者 2 位。停止位发送结束以后，又可以通过设置起始位来开始新一轮的发送。

流控制

数据在两个 UART 之间传输时，常常会出现丢失数据的现象，如 Pc 机与微控制器之间的通讯，接收端数据缓冲区已满，则此时继续发送来的数据就会丢失。特别是使用 Pc 与 MODEM 进行数据传输的时候，这个问题就尤为突出，流控制就是用于解决这个问题的。当接收端数据处理不过来时，

就发出“不再接收”的信号，发送端就停止发送，直到收到“可以继续发送”的信号再发送数据。因此流控制可以控制数据传输的进程，防止数据的丢失。UART 常用的两种流控制是硬件流控制(包括 RTS / CTS、DTR / CTS)和软件流控制 XON / XOFF(继续 / 停止)。

硬件流控制常用的有 RTS / CTS(请求发送 / 清除发送)流控制和 DTR / DSR(数据终端就绪 / 数据设置就绪)流控制。使用 RTS / CTS 流控制时，应将通讯两端的 RTS、CTS 线对应相连，数据终端设备(如 Pc 机)使用 RTS 来起始数据通讯设备(如 MODEM)的数据流，而数据通讯设备则用 CTS 来起动和暂停来自数据终端设备的数据流。

软件流控制一般通过 XON / XOF'F″来实现软件流控制。方法是：当接收端的输入缓冲区内数据量超过设定的高位时，就向数据发送端发出 XOPP 字符(十进制的 I9)，发送端收到 XOPP 字符后就立即停止发送数据；当接收端的输入缓冲区内数据量低于设定的低位时，就向数据发送端发出 XON 字符(十进制的 I7)，发送端收到 XON 字符后就立即开始发送数据。显然，若传输的是二进制数据，标志字符也有可能在数据流中出现而引起误操作，这是软件流控制的缺陷，而硬件流控制不会有这个问题。

2. UART I6550

16550 是实际上的工业标准 UART，下面我们就以 16550C 为例进行介绍。

特性

- 可编程 5、6、7 或 8 个数据位，奇、偶或无校验，1、1. 5、2 个停止位；
- 可编程的自动 RTS、CTS 流控制模式；
- 软件与 I6450 完全兼容，复位时寄存器与 I6450 设置相同；
- 可编程波特率发生器；
- 可以使用独立的输入时钟；
- 可独立控制的发送、接受、在线状态和数据中断；
- 完整的状态报告能力；
- 掉线信号产生和检测功能；
- 白循环和内部的错误模拟功能；
- Modem 控制功能(CTS，RTS，DSR，DTR，RI 和 DCD 信号)。

16550C 的串行接口信号描述

信号	功能
SIN	串行数据输入
RCLK	外部输入时钟
RTS	请求发送
CTS	清除发送
BAUDOUT	波特率输出
SOUT	串行数据输出
DSR	数据设置就绪
DTR	数据终端就绪
DCD	数据载波指示
RI	振铃指示

16550C 的寄存器列表。

名称	地址	宽度	访问	描述
DLAB=0，RBR	0	8	只读	接受缓冲
DLAB=0，TBR	0	8	只写	发送保持
DLAB=1，DLL	0	8	读写	波特率因子低字节
DLAB 0，IER	1	8	读写	中断控制
DLAB=1，DLM	1	8	读写	波特率因子高字节
IIR	2	8	只读	中断标志
FCR	2	8	只写	FIFO控制
LCR	3	8	读写	线控制
MCR	4	8	只写	Modem控制
LSR	5	8	只读	线状态
MSR	6	8	只读	Modem状态

RBR、THR、DLL 和 IER、DLM 这 5 个寄存器由 DLAB 位决定谁可以被访问。DLL 和 DLM 组成 16 位的波特率因子寄存器。RBR 和 THR 分别是接受 FIFO 的出口和发送 FIFO 的入口。

IER 用以实现中断使能等控制，各个位的功能如下所示。

0	读写	接受中断使能控制　　0 禁止；1使能。
1	读写	THR寄存器空中断使能控制：　0 禁止；1使能。
2	读写	接受线路状态中断使能控制：0 禁止；1使能。
3	读写	Modem状态中断使能控制：　　0 禁止；1使能。
7 4	读写	保留。

IIR 的功能是显示当前中断的状态

位	访问	描述
0	只读	0 有中断 1 没有中断
1	只读	中断ID
2 3		011 接受线状态； 010 接受数据准备好； 110 字符超时； 001 THR空； 000 Modem状态。

4	只读	固定值0。
5	只读	固定值0。
6	只读	1 启用了FIFO；0 没有启用。
7	只读	1 启用了FIFO；0 没有启用。

FCR 用于控制 FIFO,, 各个位的功能

位	访问	描述
0	只写	0 不启用FIFO；1 启用FIFO。
1	只写	写1 清除接收FIFO。
2	只写	写1 清除发送FIFO。
3	只写	复位 RXRDY和TXRDY信号。
5 4		保留。
7 6	只写	定义FIFO大小： 00 1字节 01 4字节 10 8字节 11 14字节

LCR 控制特性的基本协议，各个位的功能

位	访问	描述
1 0	读写	数据位长度： 00 5位； 01 6位； 10 7位； 11 8位。
2	读写	停止位长度： 0 1个停止位； 1 5位数据位的时候使用1.5个停止位， 其他使用2个停止位。
3	读写	校验位： 1 有校验。 0 无校验；
4	读写	校验类型位： 0 奇校验； 1 偶校验。
5	读写	粘附奇偶检查位： 0 禁止粘附奇偶检查位； 1 使能粘附奇偶检查位。

6	读写	问断控制位： 1 串行输出固定在逻辑0； 0 不使用问断。
7	读写	DLAB： 1 访问波特率因子寄存器； 0 访问正常寄存器。

MCR 用于控制 Modem 信号，各个位的功能

位	访问	描述
0	只写	DTR信号控制 ‘0’ DTR is ‘1’ ‘1’ DTR is ‘0’
1	只写	RTS信号控制： ‘0’ RTS is ‘1’ ‘1’ RTS is ‘0’
2	只写	Out1控制在Loopback模式时连接到 RI信号输入。
3	只写	Out2控制。在Loopback模式时连接到DCD信号输入。
4	只写	Loopback模式： 0 正常工作模式； 1 Loopback模式。此时，SOUT保持逻辑1。 发送移位寄存器的输出与接收移位寄存器的 输入在内部连接。
5	只写	自动流控制使能： 0 不使用自动模式 1 MCR1 0时，只使用CTS自动模式； MCR1 1时，使用CTS自动和RTS自动模式

LSR 用于指示发送和接收的状态，各个位的功能表

位	访问	描述
0	只读	接收数据标志： 0 接收FIFO空； 1 接收FIFO不空。
1	只读	接收FIFO溢出标志： 1 FIFO满并且正在接收移位寄存器正在接收 新数据。读LSR以后自动清除该位。 0 没有接收FIFO溢出。

2	只读	校验错误标志： 1 FIFO顶部的字节有校验错误。读LSR以后自动清除该位。 0 当前字节没有校验错误。
3	只读	帧错误标志： 1 FIFO顶部的字节有帧错误。读LSR以后 自动清除该位。 0 当前字节没有帧错误。
4	只读	问断中断标志： 1 当前字节有问断错误。当串行接收信号保持一段时问的逻辑0时，认为发生问断错误。此时，一个0字节进入FIFO。读LSR以后自动清除该位。 0 当前字节没有问断错误。
5	只读	发送FIFO空标志： 1 THR变为空，但是发送移位寄存器不为空，产生了THR空中断。向发送FIFO写入数据后自动清除该位。 0 其他情况。
6	只读	发送数据空标志： 1 THR变为空和发送移位寄存器都变为空，产生了THR空中断。向发送FIFO写入数据后自动清除该位。 0 其他情况。
7	只读	1 FIFO模式时，至少有一个检验错误、帧错误或者问断发生，读LSR以后自动清除该位。 0 其他情况。

MSR 寄存器用以显示 Modem 信号的状态，各个位的功能表

位	访问	描述
0	只读	CTS变化标志
1	只读	DSR变化标志：1 DSR发生了变化。
2	只读	RI变化标志。RI信号由低变为高。
3	只读	DCD变化标志：1 DCD发生了变化。
4	只读	CTS信号的输入，在Loopback模式与RTS位相同。
5	只读	DSR信号的输入，在Loopback模式与DTR位相同。
6	只读	RI信号的输入，在Loopback模式与Outl位相同。
7	只读	DCD信号的输入在Loopback模式与Out2位相同。

3. 兼容 16550 的 UART IP Core 概述

OpenCores 的 UART 16550 IP Core 基本实现了与 16550C 功能和寄存器级别的兼容，不同之处如下：

- 使用 Wishbone 并行总线，可以设置为 8 位总线模式，也可以设置为 32 位总线；
- 使用 32 位 Wishbone 总线的时候还有额外的 2 个调试寄存器；

• 只支持 FIFO 模式；

• 与 NSI6550C 兼容，但是不兼容 I6450 部分；

• 不支持 RCLK，OUTI、OUT2 信号只在 Loopback 模式中支持。

• 从串行接口信号和寄存器上说，除了不支持非 FIFO 模式以外，与 I6552 和 16554 是完全一致的。

寄存器

由于只支持 FIFO 模式，因此 FCR 的 0 位将被忽略。针对 Wishbone 总线的情况，FCR 的第 3 位也被忽略。额外添加的 2 个 32 位调试寄存器只能用于调试目的，它们都是只读的，并且对它们的访问不对内部工作状态产生任何影响。

UART16550 IP 的调试寄存器 I 定义

位	访问	描述
7.. 0	只读	LSR值。
11..8	只读	IER的0—3位的值。
15.. 12	只读	IIR的0—3位的值。
23.. 16	只读	LCR值
32.. 24	只读	MSR值

UART16550 IP 的调试寄存器 2 定义

位	访问	描述
2.. 0	只读	发送状态机。
7.. 3	只读	发送FIFO内的字符数。
11..8	只读	接收状态机
16..12	只读	接收FIFO内的字符数。
18..17	只读	IIR的0—4位的值。
23..19	只读	FCR的6—7位值。
31..24	只读	保留。

I/O 端口 端口信号分为 3 类：Wishbone 总线信号接口信号

信号	宽度	方向	描述
CLK	1	Input	既是wishbone时钟，也是波特率时钟
WB_RST_I	1	Input	异步复位。
WB_ADDR_I	5 or 3	Input	地址总线
WB_SEL_I	4	Input	选择信号
WB_DAT_I	32 or 8	Input	数据输入
WB_DAT_O	32 or 8	Output	数据输出

WB_WE_I	1	Input	写使能信号
WB_STB_I	1	Input	选通信号
WB_CYC_I	1	Input	循环信号
WB_ACK_O	1	Output	应答信号
INT_O	1	Output	中断信号

UART16550 IP 的串行接口信号

信号	宽度	方向	描述
STX_PAD_O	1	输出	串行数据输出
SRX_PAD_I	1	输入	串行数据输入
RTS_PAD_O	1	输出	请求发送
DTR_PAD_O	1	输出	数据终端就绪
CTS_PAD_I	1	输入	清除发送
DSR_PAD_I	1	输入	数据设置就绪
RI_PAD_I	1	输入	振铃指示
DCD_PAD_I	1	输入	数据载波指示
BAUD_O	1	输出	波特率输出

4. UART IP Core 的使用

第一步 首先要初始定义，包括时钟频率，波特率，接口地址等，我们把它写成一个 uart.h 文件

```
#define UART_BASE               0x90000000     // 接口地址
#define IN_CLK                  48000000       // UART IP Core 的工作频率
#define UART_BAUD_RATE          115200         // 波特率定义

/* Register access macros */
#define REG8(add) *((volatile unsigned char *)(add))
#define REG16(add) *((volatile unsigned short *)(add))
#define REG32(add) *((volatile unsigned long *)(add))
#if 1
#define UART_RX       0     /* In:  Receive buffer (DLAB=0) */
#define UART_TX       0     /* Out: Transmit buffer (DLAB=0) */
#define UART_DLL 0     /* Out: Divisor Latch Low (DLAB=1) */
#define UART_DLM 1     /* Out: Divisor Latch High (DLAB=1) */
#define UART_IER 1     /* Out: Interrupt Enable Register */
#define UART_IIR 2     /* In:  Interrupt ID Register */
#define UART_FCR 2     /* Out: FIFO Control Register */
#define UART_EFR 2     /* I/O: Extended Features Register */
                       /* (DLAB=1, 16C660 only) */
```

```
#define UART_LCR 3     /* Out: Line Control Register */
#define UART_MCR 4     /* Out: Modem Control Register */
#define UART_LSR 5     /* In:  Line Status Register */
#define UART_MSR 6     /* In:  Modem Status Register */
#define UART_SCR 7     /* I/O: Scratch Register */
#else

#define UART_RX        0     /* In:  Receive buffer (DLAB=0) */
#define UART_TX        0     /* Out: Transmit buffer (DLAB=0) */
#define UART_DLL 0     /* Out: Divisor Latch Low (DLAB=1) */
#define UART_DLM 4     /* Out: Divisor Latch High (DLAB=1) */
#define UART_IER 4     /* Out: Interrupt Enable Register */
#define UART_IIR 8     /* In:  Interrupt ID Register */
#define UART_FCR 8     /* Out: FIFO Control Register */
#define UART_EFR 8     /* I/O: Extended Features Register */
                       /* (DLAB=1, 16C660 only) */
#define UART_LCR 12    /* Out: Line Control Register */
#define UART_MCR 12    /* Out: Modem Control Register */
#define UART_LSR 20    /* In:  Line Status Register */
#define UART_MSR 24    /* In:  Modem Status Register */
#define UART_SCR 28    /* I/O: Scratch Register */
#endif

/*
 * These are the definitions for the FIFO Control Register
 * (16650 only)
 */
#define UART_FCR_ENABLE_FIFO      0x01 /* Enable the FIFO */
#define UART_FCR_CLEAR_RCVR 0x02 /* Clear the RCVR FIFO */
#define UART_FCR_CLEAR_XMIT 0x04 /* Clear the XMIT FIFO */
#define UART_FCR_DMA_SELECT 0x08 /* For DMA applications */
#define UART_FCR_TRIGGER_MASK     0xC0 /* Mask for the FIFO trigger range */
#define UART_FCR_TRIGGER_1  0x00 /* Mask for trigger set at 1 */
#define UART_FCR_TRIGGER_4  0x40 /* Mask for trigger set at 4 */
#define UART_FCR_TRIGGER_8  0x80 /* Mask for trigger set at 8 */
#define UART_FCR_TRIGGER_14 0xC0 /* Mask for trigger set at 14 */
/* 16650 redefinitions */
#define UART_FCR6_R_TRIGGER_8     0x00 /* Mask for receive trigger set at 1 */
#define UART_FCR6_R_TRIGGER_16    0x40 /* Mask for receive trigger set at 4 */
#define UART_FCR6_R_TRIGGER_24  0x80 /* Mask for receive trigger set at 8 */
#define UART_FCR6_R_TRIGGER_28    0xC0 /* Mask for receive trigger set at 14 */
#define UART_FCR6_T_TRIGGER_16    0x00 /* Mask for transmit trigger set at 16 */
#define UART_FCR6_T_TRIGGER_8     0x10 /* Mask for transmit trigger set at 8 */
#define UART_FCR6_T_TRIGGER_24  0x20 /* Mask for transmit trigger set at 24 */
```

```
#define UART_FCR6_T_TRIGGER_30    0x30 /* Mask for transmit trigger set at 30 */

/*
 * These are the definitions for the Line Control Register
 *
 * Note: if the word length is 5 bits (UART_LCR_WLEN5), then setting
 * UART_LCR_STOP will select 1.5 stop bits, not 2 stop bits.
 */
#define UART_LCR_DLAB 0x80  /* Divisor latch access bit */
#define UART_LCR_SBC  0x40  /* Set break control */
#define UART_LCR_SPAR 0x20  /* Stick parity (?) */
#define UART_LCR_EPAR 0x10  /* Even parity select */
#define UART_LCR_PARITY     0x08  /* Parity Enable */
#define UART_LCR_STOP 0x04  /* Stop bits: 0=1 stop bit, 1= 2 stop bits */
#define UART_LCR_WLEN5  0x00      /* Wordlength: 5 bits */
#define UART_LCR_WLEN6  0x01      /* Wordlength: 6 bits */
#define UART_LCR_WLEN7  0x02      /* Wordlength: 7 bits */
#define UART_LCR_WLEN8  0x03      /* Wordlength: 8 bits */

/*
 * These are the definitions for the Line Status Register
 */
#define UART_LSR_TEMT 0x40  /* Transmitter empty */
#define UART_LSR_THRE 0x20  /* Transmit-hold-register empty */
#define UART_LSR_BI   0x10  /* Break interrupt indicator */
#define UART_LSR_FE   0x08  /* Frame error indicator */
#define UART_LSR_PE   0x04  /* Parity error indicator */
#define UART_LSR_OE   0x02  /* Overrun error indicator */
#define UART_LSR_DR   0x01  /* Receiver data ready */

/*
 * These are the definitions for the Interrupt Identification Register
 */
#define UART_IIR_NO_INT     0x01  /* No interrupts pending */
#define UART_IIR_ID   0x06  /* Mask for the interrupt ID */

#define UART_IIR_MSI  0x00  /* Modem status interrupt */
#define UART_IIR_THRI 0x02  /* Transmitter holding register empty */
#define UART_IIR_TOI  0x0c  /* Receive time out interrupt */
#define UART_IIR_RDI  0x04  /* Receiver data interrupt */
#define UART_IIR_RLSI 0x06  /* Receiver line status interrupt */

/*
 * These are the definitions for the Interrupt Enable Register
```

```
 */
#define UART_IER_MSI   0x08  /* Enable Modem status interrupt */
#define UART_IER_RLSI  0x04  /* Enable receiver line status interrupt */
#define UART_IER_THRI  0x02  /* Enable Transmitter holding register int. */
#define UART_IER_RDI   0x01  /* Enable receiver data interrupt */

/*
 * These are the definitions for the Modem Control Register
 */
#define UART_MCR_LOOP 0x10  /* Enable loopback test mode */
#define UART_MCR_OUT2 0x08  /* Out2 complement */
#define UART_MCR_OUT1 0x04  /* Out1 complement */
#define UART_MCR_RTS   0x02  /* RTS complement */
#define UART_MCR_DTR   0x01  /* DTR complement */

/*
 * These are the definitions for the Modem Status Register
 */
#define UART_MSR_DCD   0x80  /* Data Carrier Detect */
#define UART_MSR_RI    0x40  /* Ring Indicator */
#define UART_MSR_DSR   0x20  /* Data Set Ready */
#define UART_MSR_CTS   0x10  /* Clear to Send */
#define UART_MSR_DDCD 0x08  /* Delta DCD */
#define UART_MSR_TERI 0x04  /* Trailing edge ring indicator */
#define UART_MSR_DDSR 0x02  /* Delta DSR */
#define UART_MSR_DCTS 0x01  /* Delta CTS */
#define UART_MSR_ANY_DELTA 0x0F   /* Any of the delta bits! */

/*
 * These are the definitions for the Extended Features Register
 * (StarTech 16C660 only, when DLAB=1)
 */
#define UART_EFR_CTS   0x80  /* CTS flow control */
#define UART_EFR_RTS   0x40  /* RTS flow control */
#define UART_EFR_SCD   0x20  /* Special character detect */
#define UART_EFR_ENI   0x10  /* Enhanced Interrupt */
```

第二步 UART IP Core 的软件初始化

```
void uart_init(void)
{
   int devisor;
    /* Reset receiver and transmiter */
    REG8(UART_BASE + UART_FCR) = UART_FCR_ENABLE_FIFO | UART_FCR_CLEAR_RCVR | UART_FCR_CLEAR_XMIT |
UART_FCR_TRIGGER_14;
    /* Disable all interrupts */
```

```
        REG8(UART_BASE + UART_IER) = 0x00;
        /* Set 8 bit char, 1 stop bit, no parity */
        REG8(UART_BASE + UART_LCR) = UART_LCR_WLEN8 & ~(UART_LCR_STOP | UART_LCR_PARITY);
         /* Set baud rate */
        devisor = IN_CLK/(16 * UART_BAUD_RATE);
        REG8(UART_BASE + UART_LCR) |= UART_LCR_DLAB;
        REG8(UART_BASE + UART_DLL) = devisor & 0x000000ff;
        REG8(UART_BASE + UART_DLM) = (devisor >> 8) & 0x000000ff;
        REG8(UART_BASE + UART_LCR) &= ~(UART_LCR_DLAB);
        return;
}
```

再写一些基本的串口收发函数，就可以使用它了

```
#define BOTH_EMPTY (UART_LSR_TEMT | UART_LSR_THRE)
#define WAIT_FOR_XMITR \
  do { \
       lsr = REG8(UART_BASE + UART_LSR); \
     } while ((lsr & BOTH_EMPTY) != BOTH_EMPTY)
#define WAIT_FOR_THRE \
  do { \
       lsr = REG8(UART_BASE + UART_LSR); \
     } while ((lsr & UART_LSR_THRE) != UART_LSR_THRE)
#define CHECK_FOR_CHAR (REG8(UART_BASE + UART_LSR) & UART_LSR_DR)
#define WAIT_FOR_CHAR \
  do { \
       lsr = REG8(UART_BASE + UART_LSR); \
    } while ((lsr & UART_LSR_DR) != UART_LSR_DR)
#define UART_TX_BUFF_LEN 32
#define UART_TX_BUFF_MASK (UART_TX_BUFF_LEN -1)
char tx_buff[UART_TX_BUFF_LEN];
volatile int tx_level, rx_level;

void uart_putc(char c)
{     unsigned char lsr;
      WAIT_FOR_THRE;
      REG8(UART_BASE + UART_TX) = c;
      if(c == '\n') {
        WAIT_FOR_THRE;
        REG8(UART_BASE + UART_TX) = '\r';
      }
      WAIT_FOR_XMITR;
}

void uart_print_str(char *p)
{
```

```
    while(*p != 0) {
      uart_putc(*p);
      p++;
   }
}
void uart_print_long(unsigned long ul)
{
  int i;
  char c;
  uart_print_str("0x");
  for(i=0; i<8; i++) {
  c = (char) (ul>>((7-i)*4)) & 0xf;
  if(c >= 0x0 && c<=0x9)
    c += '0';
  else
    c += 'a' - 10;
  uart_putc(c);
  }
}
```

这样我就可以在程序中调用串口了

```
#include "uart.h"
#include "stdio.h"
int main()
{
      uart_init();
      uart_print_str("Uart Baud Rate:115200\r\n");
      while(1);
      return 0;
}
```

开源的 UART IP core 使用就写到这里，下面再介绍一个简化的，占用很少的资源的 uart ip