以太网络及UART相互转传

NuMicro® 64位系列微处理器范例代码介绍

文件信息

应用简述	此范例程序代码演示了以太网络和 UART 之间的相互转传。如果用户将 UART 的 TX/RX 连接起来，则会使以太网和 UART 之间，持续相互转传。
BSP 版本	MA35D1_Linux-5.10.y
开发平台	NuMaker-SOM-MA35D16A8 V2.0

The information described in this document is the exclusive intellectual property of Nuvoton Technology Corporation and shall not be reproduced without permission from Nuvoton.

Nuvoton is providing this document only for reference purposes of NuMicro microcontroller and microprocessor based system design. Nuvoton assumes no responsibility for errors or omissions.

All data and specifications are subject to change without notice.

For additional information or questions, please contact: Nuvoton Technology Corporation.

www.nuvoton.com

1. 概述

这个应用程序展示了如何通过网络远程控制多个串行设备，通常应用于工业自动化数据传输和远程环境监测。在这个演示中，当 MA35D1 的 UART Port 接收到数据时，会将接收到的数据通过 Ethernet 传送到网络上对应的端口；反之亦然，当 MA35D1 从 Ethernet 接收到资料时，会根据所收到的 Ethernet 端口，将资料传送到对应的 UART Port 上。这种方式有效地实现了不同设备之间的数据转传。

2. 代码介绍

UART 及 Ethernet 间的转传，代码位于 *eth2uart.c*。

在 main() 函数中，首先会配置 UART，并将成功配置的 UART 加入到设备清单中。同时，建立 Ethernet 网络接口以启动网络。此外，还会利用 pthread_create() 创建一个线程(tcp_process)，负责处理 UART 和 Ethernet 之间的数据转传，并利用 pthread_join() 等待此线程执行结束。

```
int main(int argc, char **argv)
{
    unsigned int i;

    char file_name[50];
    int tcp_fd;

    int32_t ret;
    uint32_t uart_port_num;

    struct Objects_Opt obj;
    FILE *fp;

    _UART_PARS_s UART_Conf;

    /* config uart */
    for(i = 1; i <= TEST_PORT_NUM; i++)
    {
        uart_port_num = i;

        UART_Conf.baudrate = 115200;
        UART_Conf.datalen = 8;
        UART_Conf.stopbit = 1;
        UART_Conf.odd = 0;

        serial_info=open_tty(uart_port_num);
        if(serial_info == NULL)
        {
            continue;
        }

        if(set_tty_speed(serial_info, UART_Conf.baudrate))
        {
            continue;
        }

        if((ret=set_tty_parity(serial_info,UART_Conf.datalen,UART_Conf.odd,
        UART_Conf.stopbit))==0)
        {
            obj.fd = serial_info->fd;
            obj.port_type = UART;
            obj.port = (PORT_BASE+uart_port_num);
```

```
            /* added device to the device list table */
            if(!add_link(&obj,&tcp_max_fd,&tcp_link_head,&tcp_link_tail))
            {
                printf("\nuart add_link occure an error\n");
                while(1);
            }
        }
        else
        {
            printf("set uart%d fail! \n", i);
        }
    }

    for(i = 1; i <= TEST_PORT_NUM; i++)
    {
        /* Create Ethernet socket */
        fd[i] = create_socket(PORT_BASE+i);
    }

    printf("\n\n");

    /* Create thread to run demo */
    if(pthread_create(&tcp_thread_id,NULL,tcp_process,0) !=0 )
    {
        printf("UserTimer thread (create) error\n");
        while(1);
    }

    pthread_join(tcp_thread_id, NULL);

    return 0;
}
```

在 main() 中，open_tty() 的作用是打开一个指定 id 的串口设备，初始化相关信息，并返回一个指向 _TTY_INFO_s 结构体的指针。下面是对程序代码的说明：

1. 分配内存：

   - 使用 malloc()函数分配内存以存储 _TTY_INFO_s 结构体。变量 ptty 则用于存储 _TTY_INFO_s 结构体的指针。

2. 初始化内存：

   - 使用 memset()将 ptty 所指向的内存区域设置为零，以确保 _TTY_INFO_s 结构体的所有字段都被初始化为零。

3. 打开设备：

   - 将串口设备的名称格式化为”/dev/ttyS{id}”并存入 ptty->name 中，其中 {id} 是传入的 id 参数。
   - 使用 open() 函数以读写模式(O_RDWR) 和不将该设备分配为控制终端(O_NOCTTY) 打开设备。如果打开失败，释放之前分配的内存，打印错误信息，并返回 NULL。

4. 设置文件描述符标志：

   - 使用 fcntl() 设置文件描述符的标志，这里将标志设置为 0，表示禁用非阻塞模式。

5. 获取当前终端属性：

   - 使用 tcgetattr() 获取当前终端的属性并保存到 ptty->otm 结构体中。

6. 返回结果：

   - 返回指向_TTY_INFO_s 结构体的指针，该结构体包含了打开的串口设备的相关信息。

```
_TTY_INFO_s *open_tty(uint32_t id)
{
    static _TTY_INFO_s *ptty;

    ptty = (_TTY_INFO_s *)malloc(sizeof(_TTY_INFO_s));
    if(ptty == NULL)
    {
        return NULL;
    }

    memset(ptty,0,sizeof(_TTY_INFO_s));

    sprintf(ptty->name,"/dev/ttyS%d",id);

    ptty->fd = open(ptty->name, O_RDWR | O_NOCTTY,0);
    if (ptty->fd <0)
    {
        free(ptty);
        printf("open tty%d fail\n",id);
        return NULL;
    }

    fcntl(ptty->fd, F_SETFL, 0);

    tcgetattr(ptty->fd,&ptty->otm);

    return ptty;
}
```

在 main() 中，set_tty_speed() 函数的作用是设置指定串口设备的波特率。它会读取当前终端属性，根据给定的波特率设置新的属性，并应用这些设置。以下说明这个函数的内容：

1. 获取当前终端属性：
   - 使用 tcgetattr() 函数获取当前终端的属性并存储到 ptty->ntm 中。如果获取属性失败，打印错误信息并返回 1。
2. 初始化 ntm 结构体：
   - 使用 bzero() 函数将 ptty->ntm 的内存清零，确保所有字段初始化为零。
3. 设置输入和输出波特率：
   - 使用 cfsetispeed() 和 cfsetospeed() 设置终端的输入和输出波特率为指定的速度常量(speed)，而此速度常量(speed)，则是根据输入的 baudrate 变量设置。
4. 清空输入缓冲区：
   - 使用 tcflush() 函数清空终端的输入缓冲区，这样可以丢弃尚未读取的数据。
   - 应用新的终端属性：
5. 使用 tcsetattr() 函数立即应用新的终端属性设置（TCSANOW）到终端。

```
int set_tty_speed(_TTY_INFO_s *ptty, uint32_t baudrate)
{
    uint32_t speed;

    if( tcgetattr(ptty->fd,&ptty->ntm) != 0)
    {
        printf("Setup Serial [%s] speed fail\n",ptty->name);
        return 1;
    }
    bzero(&ptty->ntm, sizeof(ptty->ntm));


    switch(baudrate)
    {
        case 1200:
            speed= B1200;
        break;
        case 9600:
            speed = B9600;
        break;
        case 19200:
            speed = B19200;
        break;
```

```
        case 115200:
            speed = B115200;
        break;
        case 921600:
            speed = B921600;
        break;
    }

    cfsetispeed(&ptty->ntm, speed);
    cfsetospeed(&ptty->ntm, speed);

    tcflush(ptty->fd, TCIFLUSH);
    tcsetattr(ptty->fd,TCSANOW,&ptty->ntm);

    return 0;
}
```

在 main()中，函数 set_tty_parity() 用于配置串口设置，包括数据位、奇偶校验和停止位。以下说明这个函数的内容：

1. 获取当前终端属性：
   - tcgetattr()用于获取与 ptty->fd 相关的终端的当前设置，并将这些设置储存在 ptty->ntm 中。如果获取属性失败，打印错误信息并返回 1。
2. 设置数据位：
   - 使用 ptty->ntm.c_cflag &= ~CSIZE 清除现有的数据位设置。c_cflag 则是用于控制字符大小。
   - 根据”databits”，设置数据位为 7 位(CS7)或 8 位(CS8)。
   - 如果数据位不受支持，打印错误信息并返回 5。
3. 设置奇偶校验：
   - PARENB 用于开启奇偶校验位，INPCK 用于启用奇偶校验字符检查。
   - 根据 “parity” 设置校验方式，如果选择无校验(“0”)，则关闭校验位。如果选择奇校验(“1”)或偶校验(“2”)，则根据需求设置相应的校验位和字符检查标志。如果校验方式不支持，则打印错误信息并返回 2。
4. 设定停止位：
   - 根据”stopbits”设置停止位为 1 位(~CSTOPB)或 2 位(CSTOPB)。如果选择的停止位不支持，则打印错误信息并返回 3。

5. 设定其他终端属性：

- 设置为本地模式并启用接收(CLOCAL | CREAD)。
- 使用 c_lflag、c_oflag 和 c_iflag 配置原始模式(关闭终端的规范模式、回显以及处理输出和流控的属性)。
- 设定超时(VTIME) 和最小读取字节数(VMIN) ，以进行非阻塞读取。
- 使用 tcflush() 刷新输入缓冲区，并使用 tcsetattr() 应用新设置。

```
int set_tty_parity(_TTY_INFO_s *ptty,uint32_t databits,uint32_t parity,uint32_t stopbits)
{
    if( tcgetattr(ptty->fd,&ptty->ntm) != 0)
    {
        printf("Setup Serial [%s] parity fail\n",ptty->name);
        return 1;
    }

    ptty->ntm.c_cflag &= ~CSIZE;
    switch (databits)
    {
        case 7:
            ptty->ntm.c_cflag |= CS7;
        break;
        case 8:
            ptty->ntm.c_cflag |= CS8;
        break;
        default:
            printf("Unsupported data size\n");
        return 5;
    }

    /* set parity */
    switch (parity)
    {
        case 0: // non-parity
            ptty->ntm.c_cflag &= ~PARENB;
            ptty->ntm.c_iflag &= ~INPCK;
        break;

        case 1: // odd-parity
            ptty->ntm.c_cflag |= (PARODD|PARENB);
            ptty->ntm.c_iflag |= INPCK;
        break;

        case 2: // even-parity
            ptty->ntm.c_cflag |= PARENB;
            ptty->ntm.c_cflag &= ~PARODD;
            ptty->ntm.c_iflag |= INPCK;
        break;

        default:
```

```
            printf("Unsupported parity\n");
        return 2;
    }

    /* stop-bits */
    switch (stopbits)
    {
        case 1:
            ptty->ntm.c_cflag &= ~CSTOPB;
        break;
        case 2:
            ptty->ntm.c_cflag |= CSTOPB;
        break;
        default:
            printf("Unsupported stop bits\n");
        return 3;
    }

    ptty->ntm.c_cflag |= (CLOCAL | CREAD);
    ptty->ntm.c_lflag &=~(ICANON | ECHO | ECHOE | ISIG);
    ptty->ntm.c_oflag &=~OPOST;
    ptty->ntm.c_iflag &=~(IXON | ICRNL);

    ptty->ntm.c_cc[VTIME] =0;
    ptty->ntm.c_cc[VMIN] = 1;
    tcflush(ptty->fd, TCIFLUSH);
    if (tcsetattr(ptty->fd,TCSANOW,&ptty->ntm) != 0)
    {
        printf("Setup Serial [%s] parity fail\n");
        return 4;
    }

    return 0;
}
```

在 main() 中，create_socket() 会创建一个 TCP 网络接口，绑定到指定的端口，并开始监听连接请求，以下说明这个函数的内容：

1. 初始化结构体"m_tcpsocket_addr"(这个结构体用来设定网络接口的地址结构)：
   - 利用 memset 用来清除 m_tcpsocket_addr 结构体中的所有字节，确保其值为零。
2. sin_family 设置为 AF_INET，表示这是 IPv4 协议。
   - sin_addr.s_addr 设置为 INADDR_ANY，表示接受来自任意 IP 地址的连接。
3. 创建网络接口：
   - 使用 socket 函数创建一个 TCP 网络接口。AF_INET 表示 IPv4，SOCK_STREAM 表示 TCP，0 是协议类型，通常设置为 0。

4. 设置端口并绑定网络接口：

   - 使用 htons 函数将端口号从主机字节序转换为网络字节序。
   - 使用 bind 函数将网络接口绑定到指定的端口上。

5. 开始监听连接请求：

   - 利用 listen() 函数将网络接口设置为被动模式，准备接受连接请求。程序中的 10 表示可以排队的最大连接数。

6. 将设备添加到链表中并返回网络接口描述符(fd)：

   - 在函数 add_link() 中，会将此设备添加到链表中，以利之后查询和修改设备的状态。
   - 成功后，返回网络接口描述符(fd)。

```
int create_socket(unsigned short port)
{
    int fd;
    int Client_Sockfd;
    struct sockaddr_in m_tcpsocket_addr;
    struct sockaddr_in Client_addr;
    struct Objects_Opt obj;

    /* initial memory */
    memset(&m_tcpsocket_addr,0,sizeof(m_tcpsocket_addr));

    m_tcpsocket_addr.sin_family = AF_INET;

    m_tcpsocket_addr.sin_addr.s_addr = INADDR_ANY;

    fd = socket(AF_INET,SOCK_STREAM,0);

    if(-1 == fd)
    {
        printf("create tcp socket failed,errno = %d\n",errno);
        return -1;
    }

    m_tcpsocket_addr.sin_port = htons(port);
    if(bind(fd,(struct sockaddr *)&m_tcpsocket_addr,sizeof(m_tcpsocket_addr))==-1)
    {
        printf("Bind1 failed!\n");
        return -1;
    }

    if(listen(fd,10)==-1)
    {
        printf("Listen is failed!\n");
        return -1;
    }
```

```
    obj.port = port;
    obj.fd   = fd;
    obj.port_type = ETHERNET;

    if(!add_link(&obj,&tcp_max_fd,&tcp_link_head,&tcp_link_tail))
    {
        printf("\nadd_link occure an error\n");
        while(1);
    }

    return fd;
}
```

在 tcp_process 线程中，当 MA35D1 首次检测到通过 Ethernet 连接的设备时，会将该设备加入到设备清单的表中。如果这个设备断开连接，它也会从设备清单中移除。

接着，当从 Ethernet 收到资料时，线程会根据收到资料的 Ethernet 端口，从设备清单表中找到对应的 UART 端口，然后将数据从这个 UART 端口传送出去。同样地，从 UART 端口收到的数据也会根据设备列表中的对应设定，透过相应的 Ethernet 端口传送出去。

下面是对此线程的说明：

1. 初始化 net_fds：
   - 使用 FD_ZERO() 初始化 net_fds (fd_set) 文件描述符集，使其为空。
   - 数据结构 fd_set 与 select() 函数一起使用来监视多个档案描述符。它允许您追踪哪些档案描述符已准备好读取、写入或有错误。
2. 设置监视的文件描述符：
3. 遍历链表中的所有 Links_Objects 节点，将每个有效的文件描述符利用 FD_SET() 添加到 net_fds 集合中，这样之后 select() 函数会监视这些描述符。
4. 等待文件描述符的变化：
   - 使用 select() 函数等待直到至少有一个文件描述符变为就绪(select() 函数返回值大于 0 时)。tcp_max_fd 是监视的最大文件描述符加 1。
5. 处理 select() 函数返回值：
   - 根据 select() 函数的返回值处理相应的情况：错误(返回值小于 0)、超时(返回值等 0) 或有文件描述符就绪(返回值大于 0)。

- 当有文件描述符就绪(返回值大于 0)，则会根据不同端口类型处理。

6. 处理不同端口类型：

- 当 port_type == ETHERNET 时，表示有新的以太网络接上，则接受新的连接，并将其添加到链表中，同时将此 port_type 改为 CONNECT_TCP。
- 当 port_type == UART 时，表示 UART 接收到新数据，则读取 UART 中的数据并发送到对应的以太网络端口中。
- 当 port_type == CONNECT_TCP 时，表示此以太网络端口接收到新数据，则读取来自以太网络的数据，并根据端口号将数据写入相应的 UART 端口。但如果读取到的数据大小为 0 时，表示以太网络连接被关闭，则关闭该文件描述符，从集合中移除，并删除链表中的连接。

```
void *tcp_process (void *arg)
{
    int ret, uart_rev, tcp_rev, Client_Sockfd, eth_connect_fd;
    uint32_t i, r;
    struct Objects_Opt obj;
    struct sockaddr_in Client_addr;
    struct Links_Objects *p;
    socklen_t sin_size;
    fd_set net_fds;
    int wd;

    while(1)
    {
        /* initialize net_fds to zero */
        FD_ZERO(&net_fds);

        p =  tcp_link_head;

        for(;p != NULL;p = p->next)
        {
            if((p->opt).fd > 0)
            {
                /* add a file descriptor to net_fds */
                FD_SET((p->opt).fd,&net_fds);
            }
        }

        /* Wait file descriptors be set */
        ret = select(tcp_max_fd +1,&net_fds,NULL,NULL,NULL);

        if (ret < 0)
        {
            /* get number of ready file descriptors fail */
        }
        else if (ret == 0)
```

```
        {
            /* no file descriptors be set */
        }
        else if(ret > 0)
        {
            p =  tcp_link_head;

            for(;p != NULL;p = p->next)
            {
                if((p->opt).fd > 0)
                {
                    /* check if a fd is set in net_fds */
                    if (FD_ISSET((p->opt).fd, &net_fds))
                    {
                        if((p->opt).port_type == ETHERNET)
                        {   /* If a device is connected to the MA35D1,
                               add the new device to device list */

                            eth_connect_fd = accept((p->opt).fd, (struct sockaddr
*)&Client_addr, &sin_size);

                            if (eth_connect_fd <= 0)
                            {
                                printf("fd=%d accept error! \n",(p->opt).fd);
                                continue;
                            }

                            obj.fd = eth_connect_fd;
                            obj.port_type = CONNECT_TCP;
                            obj.port = (p->opt).port;

                            if(!add_link(&obj,&tcp_max_fd,&tcp_link_head,&tcp_link_tail))
                            {
                                printf("\nconnect tcp add_link occure an error\n");
                                while(1);
                            }

                        }
                        else if((p->opt).port_type == UART)
                        {   /* If the UART port receives data, the data is sent out
                               through Ethernet */

                            uart_rev = read((p->opt).fd,RxBuffer,MAX_PACKAGE);

                            if(uart_rev > 0)
                            {
                                struct Links_Objects *pp;
                                pp =  tcp_link_head;
                                for(;pp != NULL;pp = pp->next)
                                {
                                    if((pp->opt).port_type == CONNECT_TCP)
                                    {
                                        if((pp->opt).port == (p->opt).port)
                                        {
                                            send((pp->opt).fd, RxBuffer, uart_rev, 0);
                                        }
```

```
                                    }
                                }
                            }

                        }
                        else if((p->opt).port_type == CONNECT_TCP)
                        {   /* If the Ethernet receives data, the data is sent out through
                               UART */

                            tcp_rev = recv((p->opt).fd,tcp_buf,sizeof(tcp_buf),0);

                            if (tcp_rev == -1)
                            {
                                printf("tcp receive error!\n");
                                while(1);
                            }
                            else if (tcp_rev == 0)
                            {   /* Ethernet connect lose */

                                close((p->opt).fd);

                                /* clear a file descriptor from a set */
                                FD_CLR((p->opt).fd, &net_fds);

                                /* delete the connection from the linked list */
                                del_link((p->opt).fd,&tcp_link_head,&tcp_link_tail);
                            }
                            else
                            {
                                struct Links_Objects *pp;

                                pp =  tcp_link_head;
                                for(;pp != NULL;pp = pp->next)
                                {
                                    if((pp->opt).port_type == UART)
                                    {
                                        if((pp->opt).port == (p->opt).port)
                                        {
                                            write((pp->opt).fd,tcp_buf,tcp_rev);
                                        }
                                    }
                                }
                            }
                        }
                    }

                }
            }

        }

    }

    pthread_exit(NULL);
}
```

3. 软件与硬件需求

3.1 软件需求

- BSP 版本
  - MA35D1_Linux-5.10.y

3.2 硬件需求

- 电路组件
  - NuMaker-SOM-MA35D16A8 V2.0
- 示意图
  - 将 RS232_11 连接到 PC COM Port

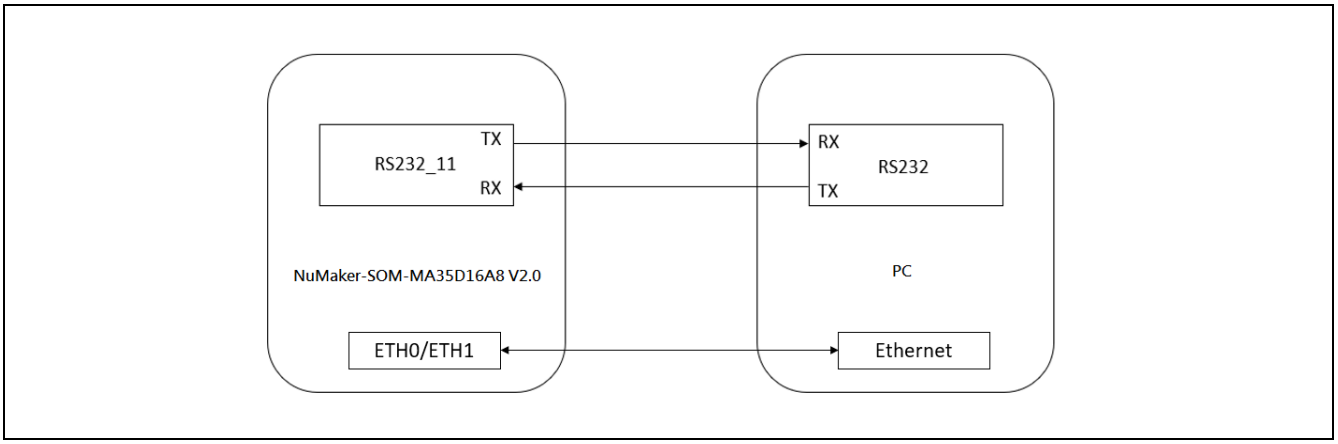


图 3-1 MA35D1 RS232_11 和 PC RS232 COM Port 连接

4. 目录信息

MA35D1_Linux_Applications	
examples	
eth2uart	Source file of example code.

图 4-1 目录信息

5. 范例程序执行

1. 根据目录信息章节进入 ETH2UART 文件夹中，并使用 " Make " 命令编译范例程序，将编译出来的档案 " eth2uart " 复制一份到 USB 硬盘中。

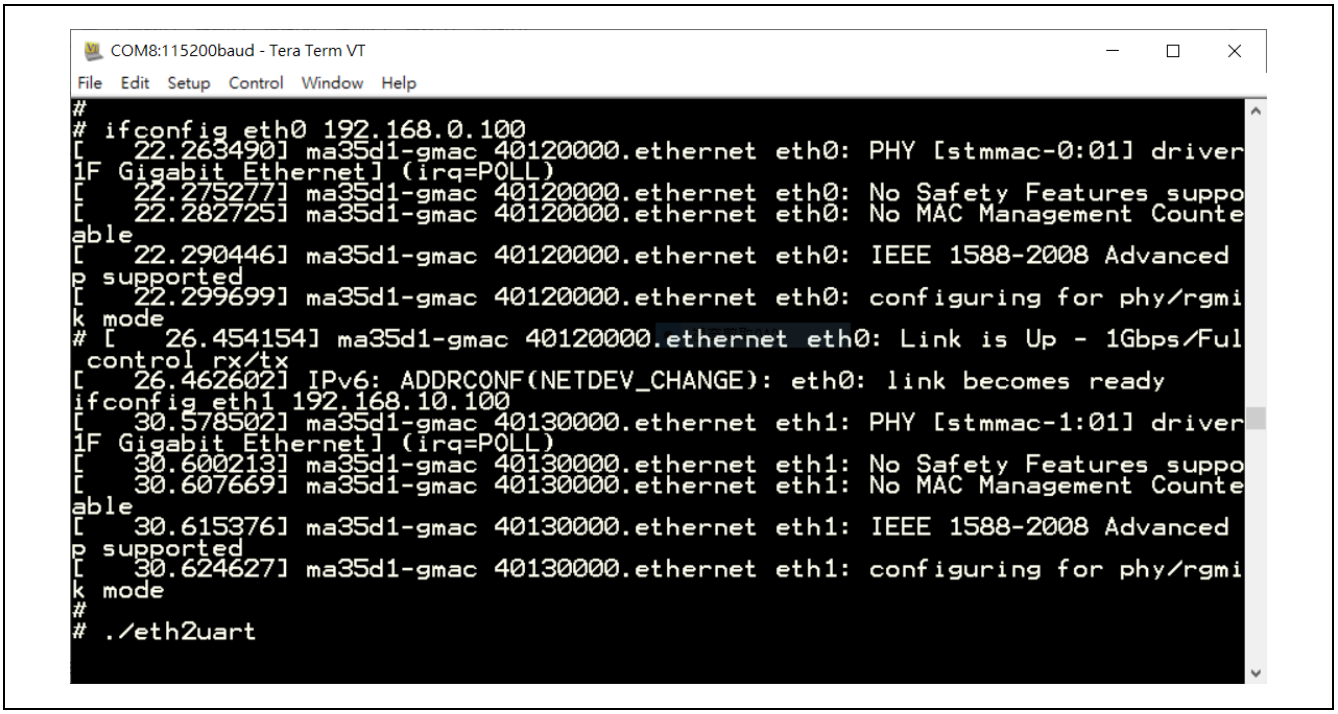
2. 参照文件(https://www.nuvoton.com/export/resource-files/en-us--MA35D1_Buildroot_Quick_Start_Rev1.01.pdf)，编译可在 SOM 板上执行的 Linux 核心程序并执行开机。

3. 将 UART 串口连接到 PC 并使用终端工具（例如：Putty 或 Teraterm）打开 Serial COM Port，波特率设定为 115200。

4. 使用 Terminal 工具开启 Serial COM Port（在 SOM 板上，使用 UART11 及 UART16）。

5. 使用 " ifconfig eth0 192.168.0.100 " 及 " ifconfig eth1 192.168.10.100 " 启动网络。

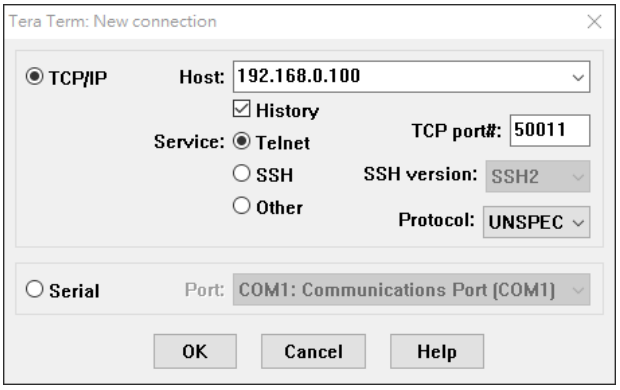
6. 将 USB 硬盘中的 " eth2uart " 档，复制到 SOM 的 Linux 上，并利用点命令 " ./eth2uart "，执行范例程序。



7. 使用 Terminal 工具开启 TCP/IP 联机：

   以太网络 0 IP 地址为 192.168.0.100，端口号码从 50001 到 50016。
   以太网络 1 IP 地址为 192.168.10.100，端口号码从 50001 到 50016。

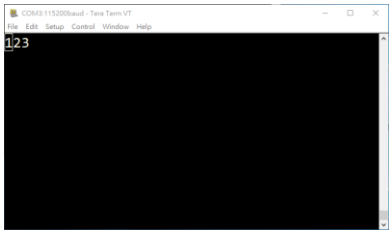
8. (端口号 50001 ~ 50016 对应到与 UART1 ~ 16 进行传输或接收)(在 SOM 板上，使用 UART11，所以使用的端口号为 50011)。



9. 以太网络传送数据至 UART 演示：

   在 TCP/IP 联机窗口中输入“123”，端口号码为 50011。

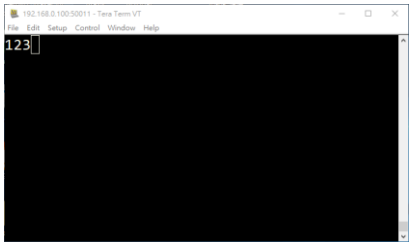
10. UART11 串行 COM 端口窗口将显示“123”。



11. UART 传送数据到以太网络演示：

    在 UART11序列 COM 端口中输入“123”。

12. 端口号码为 50011 的 TCP/IP 联机窗口将显示“123” 。



6. 修订纪录

Date	Revision	Description
2024.06.18	1.00	初始发布。

Important Notice

Nuvoton Products are neither intended nor warranted for usage in systems or equipment, any malfunction or failure of which may cause loss of human life, bodily injury or severe property damage. Such applications are deemed, "Insecure Usage".

Insecure usage includes, but is not limited to: equipment for surgical implementation, atomic energy control instruments, airplane or spaceship instruments, the control or operation of dynamic, brake or safety systems designed for vehicular use, traffic signal instruments, all types of safety devices, and other applications intended to support or sustain life.

All Insecure Usage shall be made at customer's risk, and in the event that third parties lay claims to Nuvoton as a result of customer's Insecure Usage, customer shall indemnify the damages and liabilities thus incurred by Nuvoton.

Please note that all data and specifications are subject to change without notice.
All the trademarks of products and companies mentioned in this datasheet belong to their respective owners.