乙太網路及UART相互轉傳

NuMicro® 64位元系列微處理器範例代碼介紹

文件資訊

應用簡述	此範例程式碼演示了乙太網路和 UART 之間的相互轉傳。如果用戶將 UART 的 TX/RX 連接起來，則會使以太網和 UART 之間，持續相互轉傳。
BSP 版本	MA35D1_Linux-5.10.y
開發平台	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

這個應用程式展示了如何通過網路遠端控制多個串列設備，通常應用於工業自動化資料傳輸和遠端環境監測。在這個演示中，當 MA35D1 的 UART Port 接收到資料時，會將接收到的資料通過 Ethernet 傳送到網路上對應的端口；反之亦然，當 MA35D1 從 Ethernet 接收到資料時，會根據所收到的 Ethernet 端口，將資料傳送到對應的 UART Port 上。這種方式有效地實現了不同設備之間的資料轉傳。

2. 代碼介紹

UART 及 Ethernet 間的轉傳，代碼位於 *eth2uart.c*。

在 main() 函式中，首先會配置 UART，並將成功配置的 UART 加入到設備清單中。同時，建立 Ethernet 網路接口以啟動網路。此外，還會利用 pthread_create() 創建一個執行緒 (tcp_process)，負責處理 UART 和 Ethernet 之間的資料轉傳，並利用 pthread_join() 等待此執行緒執行結束。

```
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() 的作用是打開一個指定 id 的串口設備，初始化相關信息，並返回一個指向 _TTY_INFO_s 結構體的指針。下面是對程式碼的說明：

1. 分配內存：

   - 使用 malloc() 函數分配內存以存儲 _TTY_INFO_s 結構體。變數 ptty 則用於存儲 _TTY_INFO_s 結構體的指針。

2. 初始化內存：

   - 使用 memset() 將 ptty 所指向的內存區域設置為零，以確保 _TTY_INFO_s 結構體的所有字段都被初始化為零。

3. 打開設備：

   - 將串口設備的名稱格式化為”/dev/ttyS{id}”並存入 ptty->name 中，其中 {id} 是傳入的 id 參數。
   - 使用 open() 函數以讀寫模式(O_RDWR) 和不將該設備分配為控制終端(O_NOCTTY) 打開設備。如果打開失敗，釋放之前分配的內存，打印錯誤信息，並返回 NULL。

4. 設置文件描述符標誌：

   - 使用 fcntl() 設置文件描述符的標誌，這裡將標誌設置為 0，表示禁用非阻塞模式。

5. 獲取當前終端屬性：

   - 使用 tcgetattr() 獲取當前終端的屬性並保存到 ptty->otm 結構體中。

6. 返回結果：

   - 返回指向_TTY_INFO_s 結構體的指針，該結構體包含了打開的串口設備的相關信息。

```
_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() 函數的作用是設置指定串口設備的波特率。它會讀取當前終端屬性，根據給定的波特率設置新的屬性，並應用這些設置。以下說明這個函數的內容：

1. 獲取當前終端屬性：
   - 使用 tcgetattr() 函數獲取當前終端的屬性並存儲到 ptty->ntm 中。如果獲取屬性失敗，打印錯誤信息並返回 1。
2. 初始化 ntm 結構體：
   - 使用 bzero() 函數將 ptty->ntm 的內存清零，確保所有字段初始化為零。
3. 設置輸入和輸出波特率：
   - 使用 cfsetispeed() 和 cfsetospeed() 設置終端的輸入和輸出波特率為指定的速度常量(speed)，而此速度常量(speed)，則是根據輸入的 baudrate 變數設置。
4. 清空輸入緩衝區：
   - 使用 tcflush() 函數清空終端的輸入緩衝區，這樣可以丟棄尚未讀取的數據。
5. 應用新的終端屬性：
   - 使用 tcsetattr() 函數立即應用新的終端屬性設置（TCSANOW）到終端。

```
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，函數 set_tty_parity() 用於配置串口設置，包括數據位、奇偶校驗和停止位。以下說明這個函數的內容：

1. 獲取當前終端屬性：
   - tcgetattr() 用於獲取與 ptty->fd 相關的終端的當前設置，並將這些設置儲存在 ptty->ntm 中。如果獲取屬性失敗，打印錯誤信息並返回 1。
2. 設置數據位：
   - 使用 ptty->ntm.c_cflag &= ~CSIZE 清除現有的數據位設置。c_cflag 則是用於控制字符大小。
   - 根據”databits”，設置數據位為 7 位(CS7) 或 8 位(CS8)。
   - 如果數據位不受支持，打印錯誤信息並返回 5。
3. 設置奇偶校驗：
   - PARENB 用於開啟奇偶校驗位，INPCK 用於啟用奇偶校驗字符檢查。
   - 根據 “parity” 設置校驗方式，如果選擇無校驗(“0”)，則關閉校驗位。如果選擇奇校驗(“1”)或偶校驗(“2”)，則根據需求設置相應的校驗位和字符檢查標誌。如果校驗方式不支持，則打印錯誤信息並返回 2。
4. 設定停止位:
   - 根據”stopbits”設置停止位為 1 位(~CSTOPB)或 2 位(CSTOPB)。如果選擇的停止位不支持，則打印錯誤信息並返回 3。

5. 設定其他終端屬性：

- 設置為本地模式並啟用接收 (CLOCAL | CREAD)。
- 使用 c_lflag、c_oflag 和 c_iflag 配置原始模式 (關閉終端的規範模式、回顯以及處理輸出和流控的屬性)。
- 設定超時 (VTIME) 和最小讀取字節數(VMIN)，以進行非阻塞讀取。
- 使用 tcflush() 刷新輸入緩衝區，並使用 tcsetattr() 應用新設置。

```
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() 會創建一個 TCP 網路接口，綁定到指定的端口，並開始監聽連接請求，以下說明這個函數的內容：

1. 初始化結構體"m_tcpsocket_addr" (這個結構體用來設定網路接口的地址結構) ：
   - 利用 memset 用來清除 m_tcpsocket_addr 結構體中的所有位元組，確保其值為零。
   - sin_family 設置為 AF_INET，表示這是 IPv4 協議。
   - sin_addr.s_addr 設置為 INADDR_ANY，表示接受來自任意 IP 地址的連接。
2. 創建網路接口：
   - 使用 socket 函數創建一個 TCP 網路接口。AF_INET 表示 IPv4，SOCK_STREAM 表示 TCP，0 是協議類型，通常設置為 0。

3. 設置端口並綁定網路接口：

   - 使用 htons 函數將端口號從主機字節序轉換為網絡字節序。
   - 使用 bind 函數將網路接口綁定到指定的端口上。

4. 開始監聽連接請求：

   - 利用 listen() 函數將網路接口設置為被動模式，準備接受連接請求。程式中的 10 表示可以排隊的最大連接數。

5. 將設備添加到鏈表中並返回網路接口描述符 (fd)：

   - 在函數 add_link()中，會將此設備添加到鏈表中，以利之後查詢和修改設備的狀態。
   - 成功後，返回網路接口描述符 (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 執行緒中，當 MA35D1 首次檢測到通過 Ethernet 連接的設備時，會將該設備加入到設備清單的表中。如果這個設備斷開連接，它也會從設備清單中移除。

接著，當從 Ethernet 收到資料時，執行緒會根據收到資料的 Ethernet 端口，從設備清單表中找到對應的 UART 端口，然後將資料從這個 UART 端口傳送出去。同樣地，從 UART 端口收到的資料也會根據設備清單表中的對應設定，透過相應的 Ethernet 端口傳送出去。

下面是對此執行緒的說明：

1. 初始化 net_fds：
   - 使用 FD_ZERO() 初始化 net_fds (fd_set) 文件描述符集，使其為空。
2. 資料結構 fd_set 與 select() 函數一起使用來監視多個檔案描述符。它允許您追蹤哪些檔案描述符已準備好讀取、寫入或有錯誤。
3. 設置監視的文件描述符：
   - 遍歷鏈表中的所有 Links_Objects 節點，將每個有效的文件描述符利用 FD_SET()添加到 net_fds 集合中，這樣之後 select() 函數會監視這些描述符。
4. 等待文件描述符的變化：
   - 使用 select()函數等待直到至少有一個文件描述符變為就緒(select() 函數返回值大於 0 時)。tcp_max_fd 是監視的最大文件描述符加 1。
5. 處理 select() 函數返回值：
   - 根據 select() 函數的返回值處理相應的情況：錯誤 (返回值小於 0)、超時 (返回值等 0) 或有文件描述符就緒 (返回值大於 0)。

- 當有文件描述符就緒 (返回值大於 0)，則會根據不同端口類型處理。

6. 處理不同端口類型：

- 當 port_type == ETHERNET 時，表示有新的乙太網路接上，則接受新的連接，並將其添加到鏈表中，同時將此 port_type 改為 CONNECT_TCP。
- 當 port_type == UART 時，表示 UART 接收到新數據，則讀取 UART 中的數據並發送到對應的乙太網路端口中。
- 當 port_type == CONNECT_TCP 時，表示此乙太網路端口接收到新數據，則讀取來自乙太網路的數據，並根據端口號將數據寫入相應的 UART 端口。但如果讀取到的數據大小為 0 時，表示乙太網路連接被關閉，則關閉該文件描述符，從集合中移除，並刪除鏈表中的連接。

```
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. 軟體與硬體需求

3.1 軟體需求

- BSP 版本
  - MA35D1_Linux-5.10.y

3.2 硬體需求

- 電路元件
  - NuMaker-SOM-MA35D16A8 V2.0
- 示意圖
  - 將 RS232_11 連接到 PC COM Port。

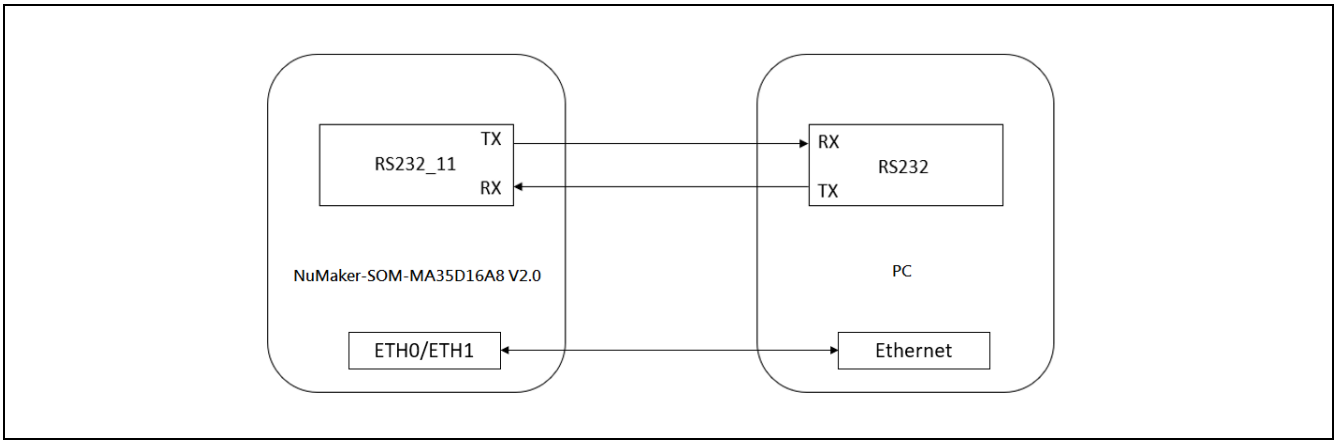


圖 3-1 MA35D1 RS232_11 和 PC RS232 COM Port 連接

4. 目錄資訊

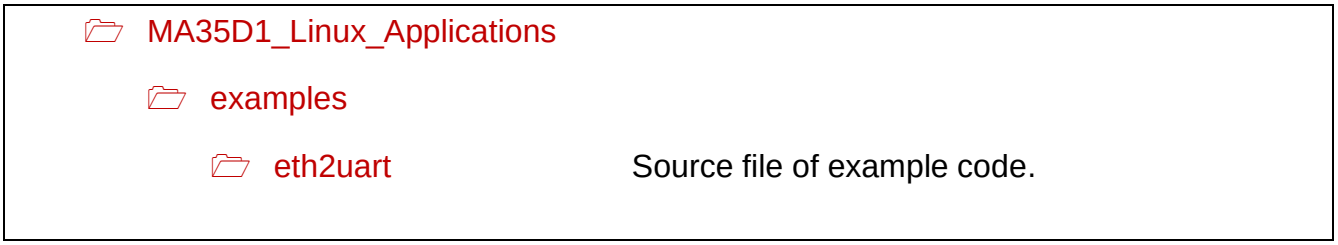


圖 4-1 目錄資訊

5. 範例程式執行

1. 根據目錄資訊章節進入 ETH2UART 資料夾中，並使用 " Make " 命令編譯範例程式，將編譯出來的檔案 " eth2uart " 複製一份到 USB 硬碟中。

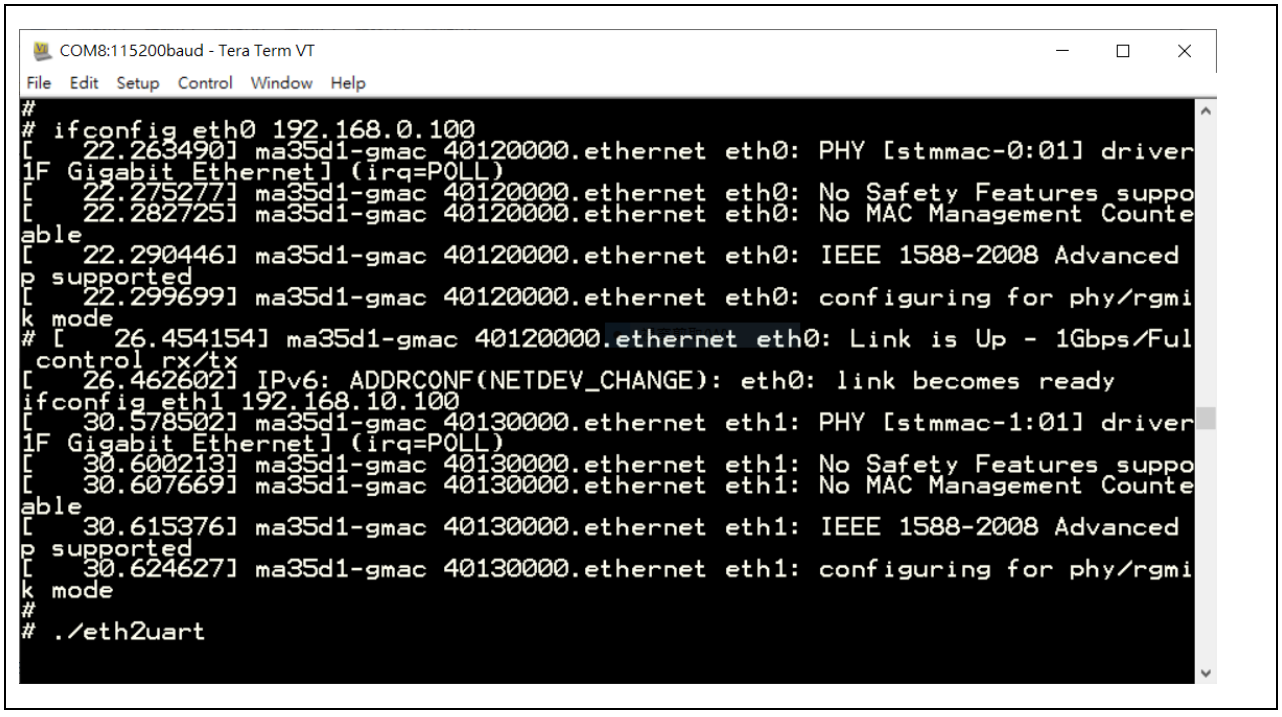
2. 參照文件(https://www.nuvoton.com/export/resource-files/en-us--MA35D1_Buildroot_Quick_Start_Rev1.01.pdf)，編譯可在 SOM 板上執行的 Linux 核心程式並執行開機。

3. 將 UART 串口連接到 PC 並使用終端工具 (例如：Putty 或 Teraterm) 打開 Serial COM Port，波特率設定為 115200。

4. 使用 Terminal 工具開啟 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 " 啟動網路。

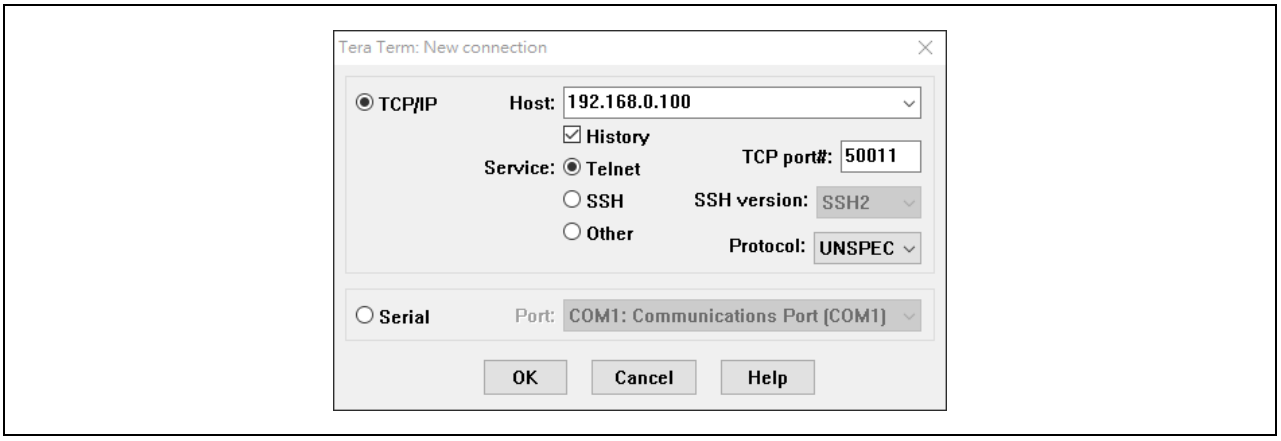
6. 將 USB 硬碟中的 " eth2uart " 檔，複製到 SOM 的 Linux 上，並利用點命令 " ./eth2uart " ，執行範例程式。



7. 使用 Terminal 工具開啟 TCP/IP 連線：

   乙太網路 0 IP 位址為 192.168.0.100，連接埠號碼從 50001 到 50016。
   乙太網路 1 IP 位址為 192.168.10.100，連接埠號碼從 50001 到 50016。

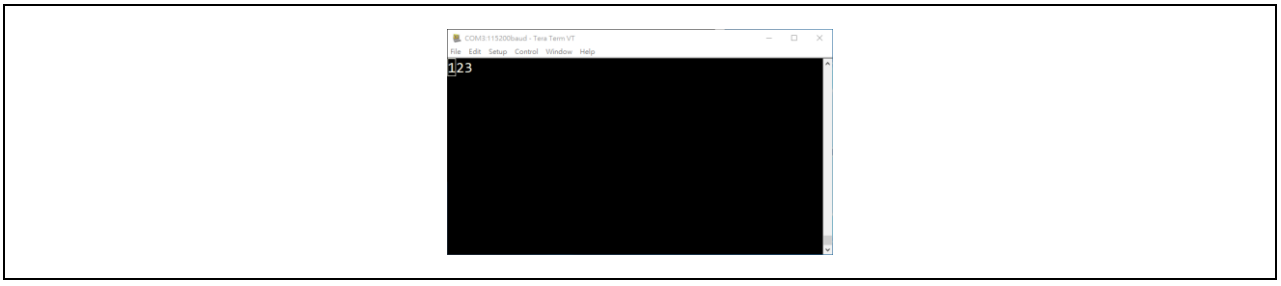
8. (連接埠號 50001 ~ 50016 對應到與 UART1 ~ 16 進行傳輸或接收) (在 SOM 板上，使用 UART11，所以使用的連接埠號為 50011)。



9. 乙太網路傳送資料至 UART 演示：

   在 TCP/IP 連線視窗中輸入“123”，連接埠號碼為 50011。

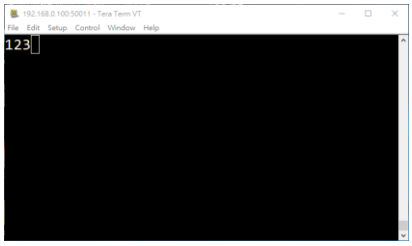
10. UART11 串列 COM 埠視窗將顯示“123”。



11. UART 傳送資料到乙太網路演示：

    在 UART11序列 COM 連接埠中輸入“123”。

12. 連接埠號碼為 50011 的 TCP/IP 連線視窗將顯示“123”。


File Edit Setup Control Window Help
123

6. 修訂紀錄

Date	Revision	Description
2024.06.18	1.00	初始發布。

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.