嵌入式系統研究

Bootloader

TFTP 伺服器功能的追加

專題指導老師: Joseph

學員: 李岳勳

3. 3, 2009

目錄

一、開發環境與研究目標

(一) 實驗板平台

使用的環境為 KS2410 EVB 開發板，S3C2410 32-bits RISC 微處理器使用 ARM920T。由於使用 RISC 類型的處理器，所以較為省電，且 Pipeline 效果較佳，比較 CISC 類型處理器同頻率下，能獲得更快的執行速度。KS2410 本身附有 GPIO 控制介面、USB、SD/MMC、RS232 … 許多介面與 64MB 的 NAND FLASH 與 RAM。

(二) 研究方向

用來作為開發板 Bootloader 為 U-boot 1.1.3，U-boot 是一種開放原始碼的嵌入式 BootROM 程式，採用高度模組化的編譯方式，大幅降低修改難度。U-boot 提供新增命令的機制，利用此機制設定指令供新更新機制使用。

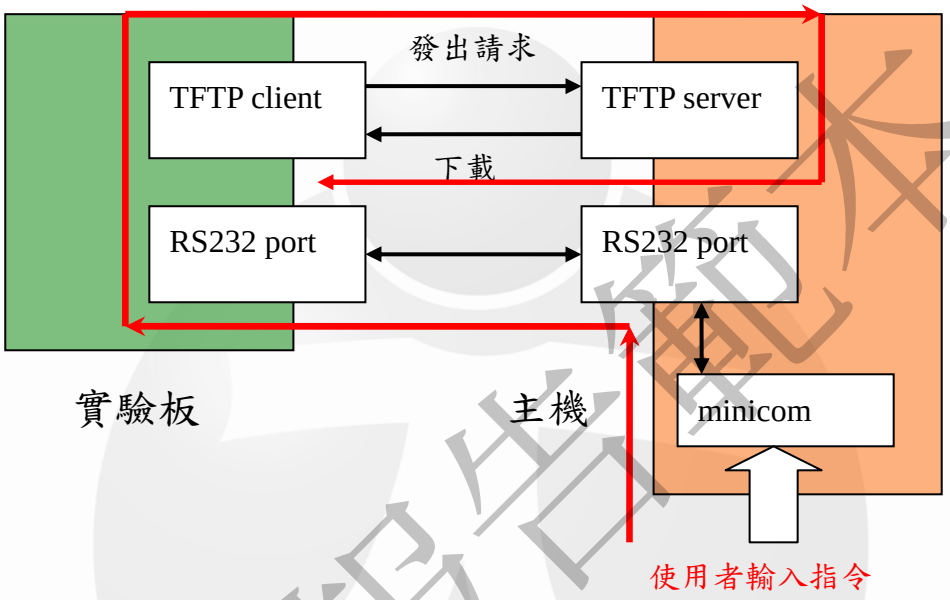
U-boot 提供一種利用 TFTP 協定傳輸能力，可由主機下載檔案。使用 tftpboot 指令從主機下載所需檔案，並利用 nand 指令寫入 NAND Flash，即成一種簡單更新系統方法，如下列指令：

```
>tftpboot 30800000 zImage
>nand erase 230000 800000
>nand write 30800000 230000 ${filesize}
```

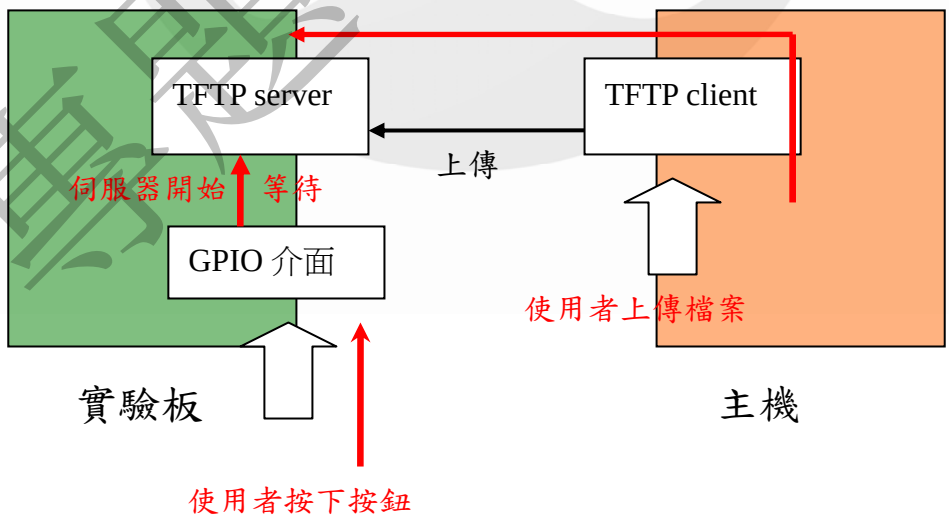
此方法雖然可以更新系統，但運用時須利用 RS-232 與終端機才能在主機端操作實驗板，執行此系列動作。除此之外，主機還需安裝 TFTP 伺服器，對開發人員雖無太大困擾，但對一般產品使用戶，其未必了解這些動作與用意。因此，設計一套新機制方便使用者更新，其方法重點即是在實驗板上架設 TFTP

伺服器。利用在 Bootloader 上架設 TFTP 伺服器功能，搭配按鍵控制，與 LED 狀態顯示，完成一套使用者可在主機利用 tftp 指令，完成更新動作。

下圖為原本更新方法示意圖：

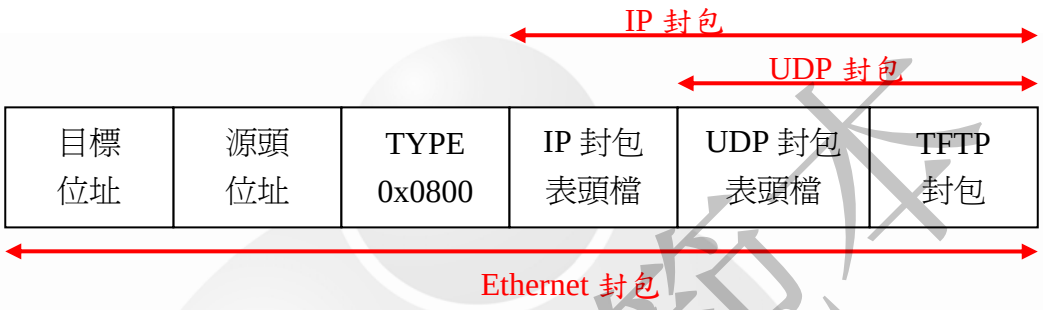


下圖為預撰寫方法示意圖：



二、TFTP 協定簡介

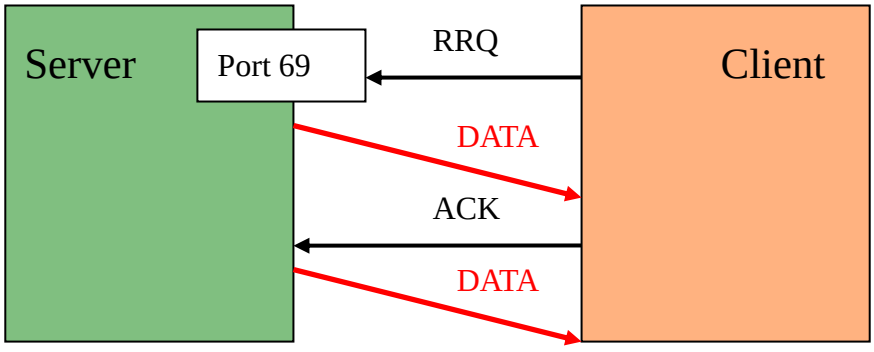
TFTP 位屬 TCP/IP 協定的應用成層，和 FTP 一樣都是用來傳輸檔案的協定，但 TFTP 適合用於小檔案且不需要 FTP 協定般複雜且多功能的傳輸場合。TFTP 使用 UDP 協定作為其傳輸層協定。傳輸封包格式如下：



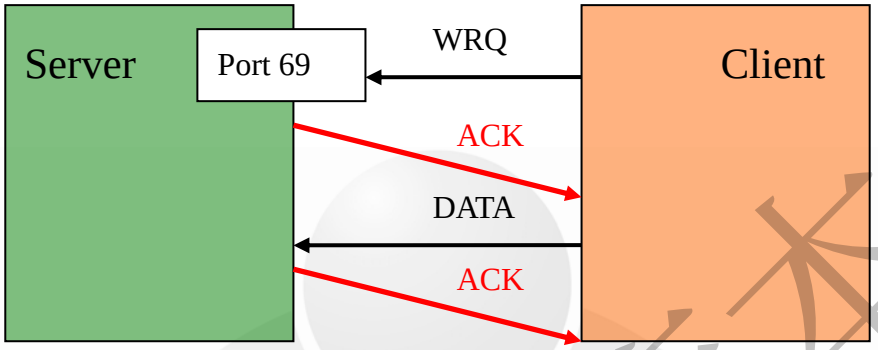
TFTP 封包類型共四種，起始封包、資料封包、ACK 封包與錯誤訊息封包。起始封包一定是由 Client 端發出，共分為兩種請求 WRQ 與 RRQ，WRQ 請求對 Server 端寫入檔案，RRQ 請求從 Server 端下載檔案。

資料封包是真正帶有待傳輸資料的封包，大小極限為 512 位元組，當接收端收到小於 512 位元組的封包，表示該封包為最後一個。ACK 封包則是用來告知對方資料封包已收到，可以繼續傳送下一個資料塊。最後，錯誤訊息封包用於當傳送過程中發生錯誤，指明錯誤的原因。

下圖為 Client 端發出 RRQ 請求時的流程，原本的更新方法採用的是此流程，其中 Server 為主機端，Client 為實驗板。



下圖為 Client 端發出 WRQ 請求時的流程，新的更新方法採用的是此流程，其中 Server 為實驗板，Client 為主機端。



三、架設 TFTP 伺服器

(一) 網路層的修改

THTP 伺服器原始碼是利用 TFTP Client 修改而成，但在撰寫 THTP 伺服器之前，須在上層協定稍作修改。

U-Boot 對網路運作的設定，於 **"net.c"** 此檔案定義詳細。該檔案利用 **NetLoop 函式**，接受上層指定協定種類，首先在 **NetLoop 函式**中加上 THTP 伺服器的初始設定。下面截取 **NetLoop 函式**一部分，紅色部分為 TFTP 伺服器所不需要，實驗板取得主機端 IP 為收到第一個請求封包時。

```
case TFTP:
    NetCopyIP(&NetOurIP, &bd->bi_ip_addr);
    NetOurGatewayIP = getenv_IPaddr ("gatewayip");
    NetOurSubnetMask= getenv_IPaddr ("netmask");
    NetOurVLAN = getenv_VLAN("vlan");
    NetOurNativeVLAN = getenv_VLAN("nvlan");
    NetServerIP = getenv_IPaddr ("serverip");
    break;
case TFTPSERVER:
    NetCopyIP(&NetOurIP, &bd->bi_ip_addr);
    NetOurGatewayIP = getenv_IPaddr ("gatewayip");
    NetOurSubnetMask= getenv_IPaddr ("netmask");
    NetOurVLAN = getenv_VLAN("vlan");
    NetOurNativeVLAN = getenv_VLAN("nvlan");
    break;
```

在初始設定後，**NetLoop 函式**呼叫 **net_check_prereq 函式**確認設定是否合法，此處修改如下。其中紅色為添加部分，在實驗板當伺服器時，NetServerIP 實指 Client IP，故不用檢查。該函式確認無誤後，會呼叫該協定的啟動函式作更進一步設定，此處添加使其呼叫 THTP 伺服器的啟動函式。

```
case TFTP:
        if (NetServerIP == 0) {
                puts ("*** ERROR: `serverip' not set\n");
                return (1);
        }
case TFTPSERVER:
        if (NetOurIP == 0) {
                puts ("*** ERROR: `ipaddr' not set\n");
                return (1);
        }
```

早先提到，實驗板取得主機端 IP 為收到第一個請求封包時，為此修改**"net.c"**檔案一個函式。**NetReceive 函式**處理實體層(Ethernet)、網路層(IP)封包解讀，但其並未將來源 IP 傳入應用層，因此追加紅色部分，用自創的 NetClientIP 變數儲存 IP 值。

```
NetClientIP = NetReadIP(&ip->ip_src);
(*packetHandler)(       (uchar *)ip +IP_HDR_SIZE,
                        ntohs(ip->udp_dst),
                        ntohs(ip->udp_src),
                        ntohs(ip->udp_len) – 8          );
```

還有兩個重要的函式在**"net.c"**，**NetSetHandler 函式**、**NetSetTimeout 函式**用來接受封包處理函式與 Timeout 發生處理函式的定義。

(二) TFTP 伺服器撰寫

上層協定修改後，參考利用 TFTP 原始碼修改撰寫出 TFTP 伺服器，兩個檔案大致上定義下列五個函式：TftpServerStart、TftpHandler、TftpTimeout、store_block 與 TftpSendTftpSend。

TftpServerStart 函式為 THTP 伺服器的啟動函式，設定 Timeout 發生與收到封包時的對應動作，與一些啟動相關初始設定，如狀態設為等待，將 Timeout 記數器歸零，主要內容如下：

```
NetSetTimeout (TIMEOUT * CFG_HZ, TftpTimeout);
NetSetHandler (TftpHandler);
TftpTimeoutCount = 0;
TftpState = STATE_WAIT;
TftpOurPort = WELL_KNOWN_PORT; //69
```

TftpHandler 函式負責處理接收的傳輸層封包。封包操作如 TFTP 協定簡介章節所敘，提出請求的一定是 Client 端，Clinet 的起始封包決定要對 Server 作讀或寫；如果請求寫入(WRQ)，Server 會回應 ACK 封包告訴 Client 可以上傳資料；如果請求下載(RRQ)，Server 會直接回應 Data 封包。

新的更新方法處理的狀況，主機向實驗板要求寫入，實驗板收到 WRQ 封包。原本實驗板的程式不處理此狀況，故需自行追加 WRQ 的對應處理，追加如下，紅色部分將之前發出請求的 IP 位置填入，使 Server 可以順利傳 ACK 封包給正確的位址：

```
case TFTP_RRQ:
        break;
case TFTP_WRQ:
        NetServerIP = NetClientIP;
        f = (uchar *)pkt;
        TftpBlock    = 0;
        TftpState = STATE_WRQ;
        TftpServerPort = src;
        TftpSend ();
        break;
```

其餘 **ACK Case** 與 **Data Case** 不用修改，除起始封包外，傳送方與接收方的動作是對應的，傳送方以 Data 封包回應收到的 ACK 封包，接收方以 ACK 回應收到的 Data 封包，不因 Server、Client 定位而有差異。

TftpTimeout 函式會被呼叫當發生 Timeout 事件。原版本中，**TftpSend 函式**被呼叫用以重新請求下載檔案，此為新版本所不需要，修改後內容如下：

```
if (++TftpTimeoutCount > TIMEOUT_COUNT) {
        NetStartAgain ();
} else {
        NetSetTimeout (TIMEOUT * CFG_HZ, TftpTimeout);
}
```

store_block 函式用來將封包資訊寫入記憶體，寫入的方法新舊版本皆相同，此檔案不用修改。

TftpSend 函式填寫將送出的封包，並呼叫**"net.c"**提供的函式將封包送往傳輸層。原版本此處重點著重於填寫 RRQ 封包與 ACK 封包。新版本中，實驗板為伺服器，不會送出請求封包，只需填寫 ACK 封包，告知主機可以傳輸檔案。

四、更新機制

(一) U-boot 指令追加

U-boot 每一個指令都是透過 U_BOOT_CMD 巨集定義，此巨集定義在 include/command.h。網路相關指令定義於 common/cmd_net.c，原機制所使用 tftpboot 指令即定義於此，仿照該指令定義 TFTP 伺服器啟動指令 thtpserver。兩者指令原型定義如下，其中 **do_tftpb** 與 **do_tftpserver 函式**為指令所對應處理動作。

```
U_BOOT_CMD(
        tftpboot,  3,     1,     do_tftpb,
        "tftpboot- boot image via network using TFTP protocol\n",
        "[loadAddress] [bootfilename]\n"
);
U_BOOT_CMD(
        tftpserver,      1,     1,     do_tftpserver,
        "tftpserver- set TFTP server, wait client data.\n",
        "[] []\n"
);
```

兩個指令對應處理動作中皆呼叫 netboot_common 函式，如下列函式。此函式定義於同檔案，用於呼叫先前在”net.c”提到的 NetLoop 函式。

```
netboot_common (TFTP, cmdtp, argc, argv);
netboot_common (TFTPSERVER, cmdtp, argc, argv);
```

為使更新動作能正常運行，避免接收並燒錄非法檔案，在傳輸檔案上加入一個檔頭供實驗板判斷，檔頭含有燒入位置的資訊、資料類別(u-boot / kernel / root file system)、資料 CRC32 的值，檔頭的宣告放於**"net.h"**中，結構如下：

```
typedef struct check_header
{
        int magic;
        char type[4];
        int model;
        unsigned long load_addr;
        unsigned long burn_addr;
        unsigned long length;
        unsigned long CRC32;
} CHACK_HEADER_t;
```

判斷方式寫於 tftpserver 指令內，在 **netboot_common 函式**結束且成功後執行。主要取出資料檔頭加以解析，如 CRC32 值的比對。CRC32 函式 u-boot 原本就有提供，可直接使用。一旦判別正確，將會依照檔頭要求的位置燒錄 nand flash 完成更新。

下列定義一用於啟動更新機制 update 指令，tftpserver 指令在完成一次傳輸後就會結束，因此需要一個更上層的機制控制。由於 tftpserver 指令本身是個完整的機制，也適合單獨使用。如果硬將更新模式寫入 tftpserver 指令，將使其除了更新外別無他處，畢竟只要在表頭使用 DATA 此資料種類，也可以用來當作單純的資料接收。

```
U_BOOT_CMD(
        update,    1,    1,    do_update,
        "update- update the system.\n",
        "\n"
);
```

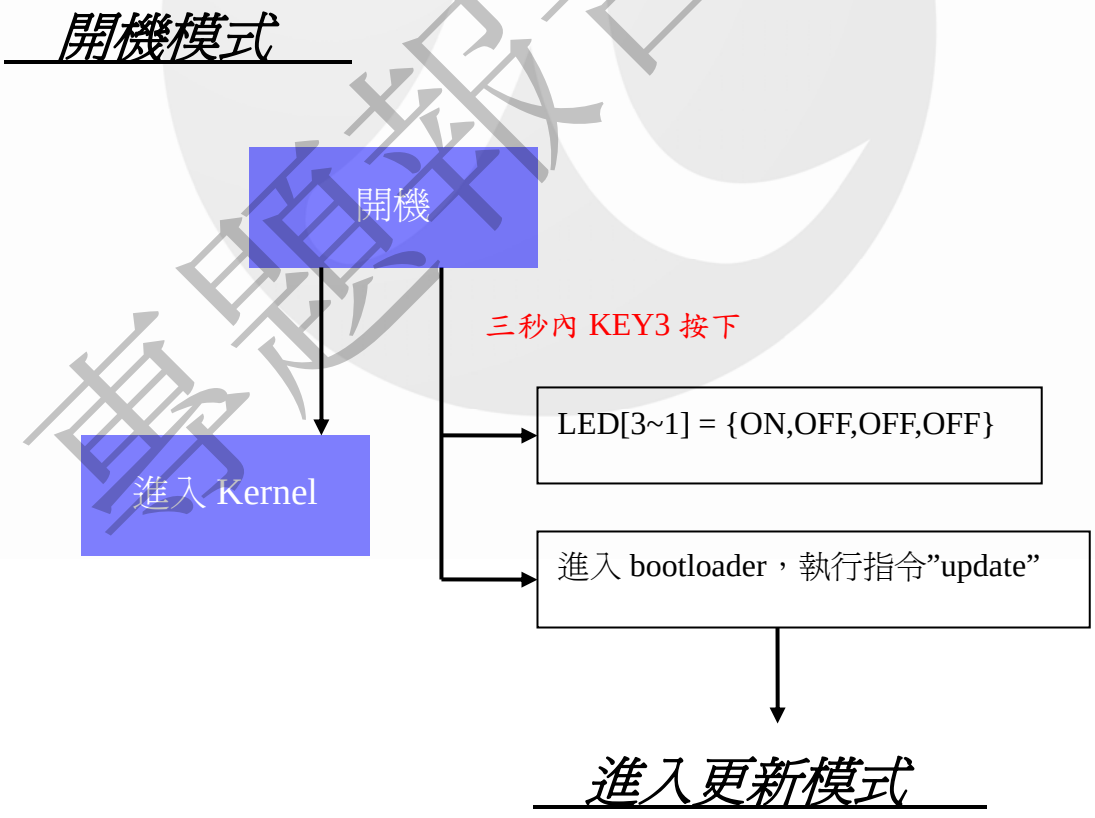
(二) GPIO 的控制

用於更新模式，主要給予 LED 燈亮暗設定，與按鍵是否按下的偵測。取得四顆 LED 燈與四個按鍵的狀況，主要利用下列程式，詳細硬體定義於規格書。

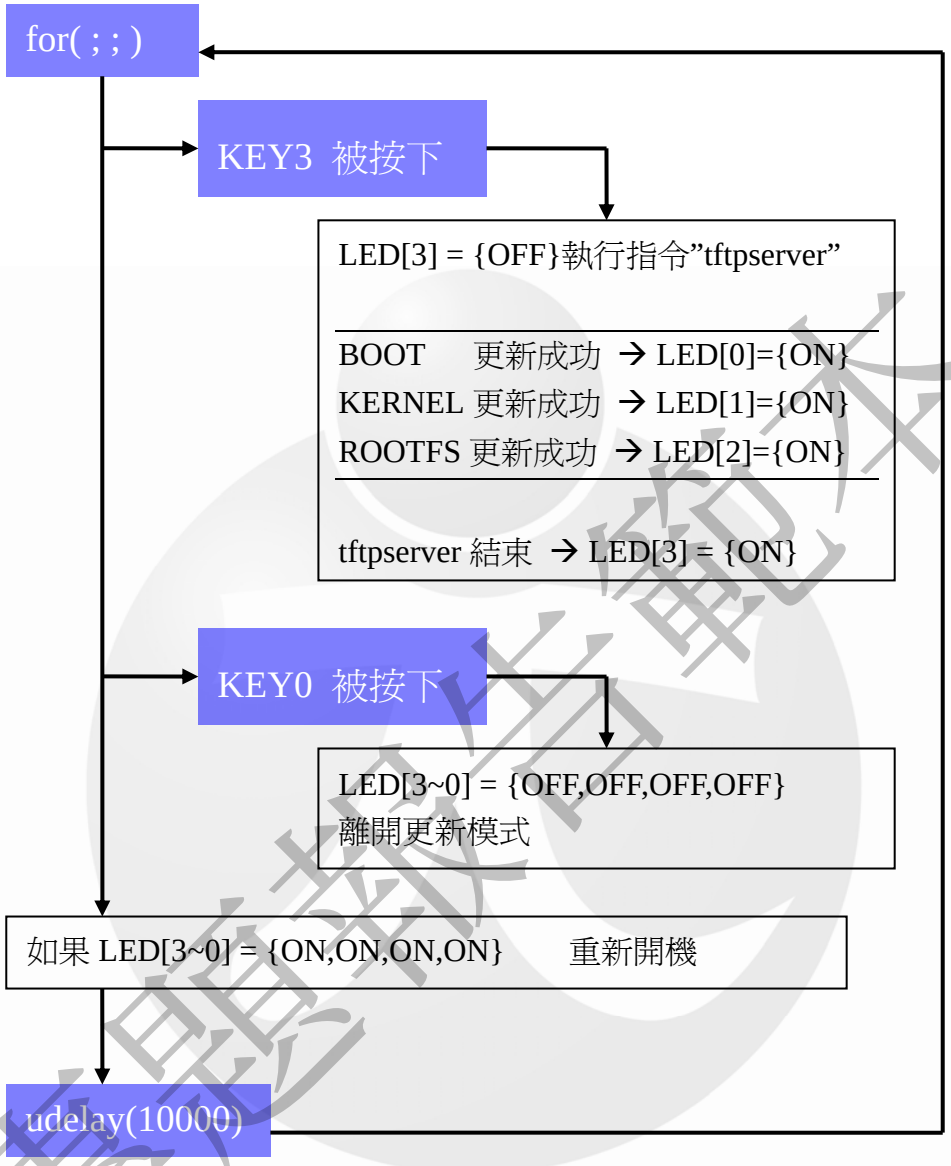
```
S3C24X0_GPIO * const gpio = S3C24X0_GetBase_GPIO();
unsigned int val32;
val32 = gpio->GPFDAT;    //狀態存於 val32
```

(三) 使用者更新機制

下面列出使用者更新機制的狀態圖：



更新模式



五、成果

下圖為開啟伺服器功能時，所顯示資訊。此處 NetServerIP 實際代表為主機 IP，而 NetOurIP 則代表開發板 IP。NetServerIP 雖顯示 192.168.1.113，但收到主機上傳請求後會改為正確 IP 位置。

```
Hit any key to stop autoboot:  1
Ready to start tftpserver.....................
Buttom [ KEY3 ] : start server.
Buttom [ KEY0 ] : exit update.
[Info] tftpserver start (Please put up date)..
To Start Server..............v1.0
Original define:(before start)
NetOurIP          : C0A80164
NetServerIP       : 192.168.1.113
NetOurIP          : 192.168.1.100
NetOurGatewayIP   : 0.0.0.0
NetOurSubnetMask  : 255.255.255.0
NetOurSubnetMask  : 255.255.255.0
NetBootFileSize   :0
Write address: 0x30008000
Wait write..........
Tftp Server Start.
Now is waiting.........
```

之後，主機利用 thtp 指令連接到實驗板，並上傳預更新之檔案。

```
[root@localhost tftpboot]# tftp 192.168.1.100
tftp> trace
Packet tracing on.
tftp> bin
tftp> put u-boot.bin
```

實驗板接收完檔案後，會檢查表頭檔，如果發現錯誤將不採用此次檔案作更新。

```
            To check file

          <  Basic data >
****************************************
        Boot file name :u-boot.bin
flash_address = E59FF014
real file size = 1f194
real file addr = 3000801C
****************************************

Magic number ..........[ ERROR ]
ERROR.
```

反之，如果確認無誤，將根據表頭檔指定位置作燒錄。

```
            To check file

          <  Basic data >
****************************************
        Boot file name :test-zImage
flash_address = 230000
real file size = 1990d4
real file addr = 3000801C
****************************************

Magic number ..........[  O K  ]
CRC32 number ..........[  O K  ]
Header Type ...........[ KERN  ]

NAND erase: device 0 offset 2293760, size 8388608 ... OK

NAND write: device 0 offset 2293760, size 1675476 ...  1
SECCUSS.
```

REFERENCES

[1] TCP/IP 通訊協定 – 入門與應用，方盈編著，博碩文化

[2]嵌入式設計及 Linux 驅動開發指南 – 基於 ARM9 處理器（ 第二版 ），大學出版社。