Ethernet Boot loader for ARM Processor

Sayali A. Kulkarni, Charudatta V. Kulkarni, Rakesh Mehta

Abstract—The Ethernet boot loader is used in the Arm processor to boot the ARM from the external flash/ memory. The internal boot loader is programmed such that the program counter (PC) starts execution from the address of the external flash device. The Ethernet and UART have been used to program the external and internal memory respectively. All the utilities and the tools used are either evaluation versions or free softwares. Hence, the use of expensive boot loaders or boot loading devices is reduced making the module cost effective. The module uses a flash of 4Mb as the external memory and an internal flash of 512Kb. The Ethernet boot loader can be effectively used for any processor with modifications as applicable. The TFTP has been implemented for the file transfer.

Index Terms—ARM, Boot loader, Ethernet, External Boot loader, TFTP, Processor Bootloader, ARM7

1 INTRODUCTION

THE Ethernet boot loader has been designed for LPC2478 (ARM7) microcontroller having an on-chip Ethernet controller and on-chip external memory controller (EMC) interfaced with an external flash.

The boot loader is designed to download application code (in form of a binary ".bin" file) from a Server using TFTP (Trivial File Transfer Protocol) and program it into the external flash. It then jumps to the external flash start address and starts executing the application code from external flash.

The literature survey for the same is done by referring the [1] – [7] references in the reference section.

2 BOOTLOADER

2.1 What is a boot loader?

The code that runs before any operating system is called the boot loader. Operating systems have their specific set of boot loader. Boot loader may be used in several ways to boot the OS kernel. The boot loader also consists of commands for debugging and/or modifying the kernel environment. The boot loader is a highly processor and board specific software or firmware which runs at every power-up or reset.

2.2 Board and Processor Specific

Boot loader commences before any other software starts in the OS. Hence, the boot loader needs to be highly processor specific and board specific. The necessary initializations to prepare the system for booting the Operating system are performed by the boot loader. The OS is more generic and usually is associated to minimum board or processor specific code.

2.3 Execution

a) Starts from ROM (Flash)

When a CPU is in power-up or reset, the memory has certain preset values in its registers. The CPU is unaware of the on-board memory and its details. It expects to find program code at a specific address. The address is the one which points to ROM or Flash, this is the commencement of the boot loader firmware.

- The primary task of the boot loader is to map the RAM to predefined addresses.
- After RAM is mapped, the Stack pointer (SP) is setup.

This is the minimal setup required, after that the boot loader starts work.

b) Moves itself to RAM for actual work

The RAM and Stack pointer are set and ready.

- The boot loader code is moved to the RAM memory for actual execution.
- The Flash memory is not used for this purpose because of its slow execution and scarce availability.
- The Flash memory is most of the times located in the address space that is non-executable.

- *Sayali Kulkarni is currently pursuing masters degree program in VLSI and Embedded Systems from Pune University, India, PH-09657710122. E-mail: kulkarnisayo@gmail.com*
- *Charudatta V. Kulkarni is currently working as a Professor in MIT College of Engineering, Pune,India E-mail: charudatta.kulkarni@mitcoe.edu.in*
- *Rakesh Mehta is currently working as the CEO of Bitmapper Integration Technologies Pvt. Ltd. , Pune, India Email: rakesh.mehta@bitmapper.com (This information is optional; change it according to your need.)*

Example: A serial flash that can be accessed by reading repetitively from one address.
Sometimes, the actual boot loader software code is compressed, and hence must be uncompressed first, before writing into the RAM.

c) Peripheral initialization

The boot loader needs to initialize peripherals required for the operation. Hence, minimum peripheral initializations are done.

In embedded systems a terminal is required. A network connection may also be required if booting is done over the network, hence Ethernet card needs to be initialized.

Initializations done are limited for the boot loader only. They are not performed for the entire system. The OS either over rides these initializations or discards them.

d) Decide which OS image to start

Boot loaders can choose to load one of several kernel images that are known to the boot loader.

This is used in embedded systems to make sure that a kernel upgrade can be done without fear of power loss during the upgrade. Also the backup of the kernel to be loaded needs to be present.

The boot loaders accept kernel image to be loaded from a known location. It provides several ways to load the kernel.

- Load kernel image from memory
- Load kernel image from file
- Load kernel image from network
- Load kernel image to memory using TFTP

3 ARCHITECTURE

The objective of the project is to develop a low power, efficient flexible and low cost hardware which can implement and be used as a User interface for the system and can be effectively used in applications related to vehicle automation.

The system has the following architecture:

1) Processor: The processor is an advanced fast processor which can meet the requirements like Ethernet, sufficient internal memory, inbuilt display driver, etc. ARM processor.
2) Ethernet: The system must have an Ethernet chip which can enable it to be controlled over the LAN.
3) External UART: It needs to have at-least 8 UART (serial) channels for communication, hence and external UART is a requirement as most processors cannot satisfy this demand.
4) Flash Memory: The external ROM is required to store the application code (firmware) if the internal memory is insufficient. Also the system has to boot up from this external memory.

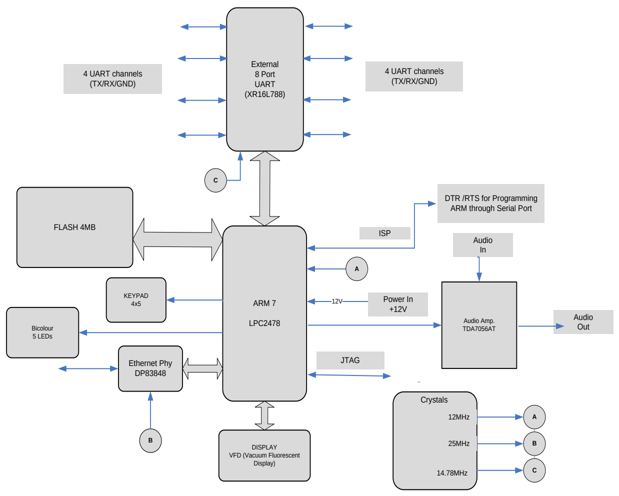

4 UART channels (TX/RX/GND)
External 8 Port UART (XR16L788)
4 UART channels (TX/RX/GND)
C
FLASH 4MB
DTR /RTS for Programming ARM through Serial Port
ISP
Audio In
A
KEYPAD 4x5
ARM 7 LPC2478
12V
Power In +12V
Bicolour 5 LEDs
Audio Amp. TDA7056AT
Audio Out
Ethernet Phy DP83848
JTAG
Crystals
12MHz
25MHz
14.78MHz
A
B
C
B
DISPLAY VFD (Vacuum Fluorescent Display)

Fig.1. Block Diagram of the system

The other interfaces and peripherals have been designed and the firmware for their interfacing is done with the information from reference [8]-[14].

4 BOOT LOADING SCHEME

The Ethernet boot loader is programmed in the internal flash of ARM controller through serial port using “Flash Magic”. After power on reset the Ethernet boot loader starts running. It initializes the ARM Ethernet controller for UDP communication and EMC (External Memory Controller) for accessing the external flash. It then sends an ARP request to PC (whose IP address is hard-coded in the boot loader) to obtain its MAC address. On receiving an ARP reply the boot loader sends a read request to a TFTP server for downloading the binary image of the application code. A TFTP Server application (as in [15] PumpKIN or [16] tftpd32) runs on PC and keeps listening for incoming TFTP requests on UDP port 69. On receiving a read request, the TFTP server prompts the user whether to grant access to the file or not. If permission is granted the TFTP server starts sending the binary file in blocks of 512 bytes to the boot loader. The boot loader accepts the data blocks and after receiving 4096 bytes programs one sector of external flash. This process continues till the entire binary file has been transferred to the external flash.
To verify that the programming of external flash has been successful the boot loader reads the external flash and uploads it as a binary file to the TFTP server. For this it sends a write request to the TFTP server. The TFTP server prompts the user whether to accept the file or not. If permission is granted the TFTP server starts accepting

data from the boot loader and stores it in a binary file. Once the uploading is complete, the received file can be compared manually with the original application code binary file using a hex editor (as in [17] HxD). The file compare must show the two files to be completely identical.

The boot loader keeps sending requests to the TFTP server for a predefined timeout period. If there is no reply from server within this period the boot loader times out and jumps to the external flash start address and starts executing the application code already present in flash. Also, if the user denies access for downloading/uploading the binary file when the TFTP server asks for permission, the TFTP server sends an error message to the boot loader. On receiving an error message the boot loader stops sending requests and jumps to the external flash start address and starts executing the application code.

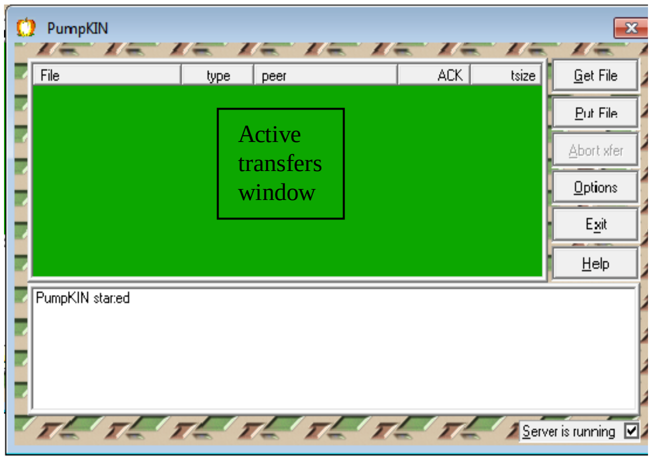
5 Trivial File Transfer Protocol

TFTP is a simple protocol to transfer files, and therefore was named the Trivial File Transfer Protocol or TFTP. It has been implemented on top of the Internet User Datagram Protocol (UDP) so it may be used to move files between machines on different networks implementing UDP. It is designed to be small and easy to implement. It can read and write files from/to a remote server. (As in [4] rfc1350 and [5] rfc2347 for detailed explanation of TFTP protocol.)

Installing and configuring the TFTP server:

The Ethernet boot loader was tested using "PumpKIN" (as in [1]) TFTP server. The procedure for installing and configuring "PumpKIN" on PC is as follows:

1. Run the executable "pumpkin-2.7.3.exe".
2. Launch "PumpKIN.exe
3. The "PumpKIN" TFTP Server window opens as shown in Fig. 2.
4. To properly configure the "PumpKIN" TFTP server click on "Options" button. The "Options" window opens as shown in Fig. 3.
5. In the "Server" tab enter the path for the TFTP root directory using the browse button. This is the folder where the TFTP server searches/stores the binary file requested by the boot loader for downloading/uploading. The user has to create a binary ".bin" file of the application code to be downloaded into the external flash and place it in this root directory. The flash image uploaded by the boot loader for verification is also stored by the Server in this root directory. The procedure for generating the binary file of the application code in "Keil

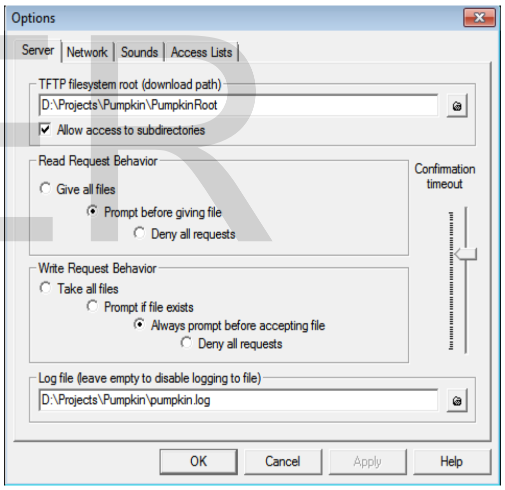


PumpKIN stared

Server is running ✓

Fig.2. PumpKIN Software window

6. In the "Read Request Behavior" and "Write Request Behavior" in the "Server" tab select "Prompt before giving file" and "Always prompt before accepting file" respectively. On using these settings the Server will ask for permission before giving or accepting a file. If this is not desired other options can be selected as convenient.



TFTP filesystem root (download path)
D:\Projects\Pumpkin\PumpkinRoot
✓ Allow access to subdirectories

Read Request Behavior
Give all files [✓ Prompt before giving file] Deny all requests

Write Request Behavior
Take all files Prompt if file exists [✓ Always prompt before accepting file] Deny all requests

Confirmation timeout

Log file (leave empty to disable logging to file)
D:\Projects\Pumpkin\pumpkin.log

Fig.3. PumpKIN Software – Options Tab - Server

7. The duration for which the Server waits for a reply from the boot loader can be adjusted using the "Confirmation timeout" slider.
8. In the "Log file" text box enter the directory path and filename of the log file to be created. The log file logs all download/upload requests received by Server. The log file can be viewed in any text editor like "notepad" etc.
9. Click on the "Network" tab. The "Network" tab opens as shown in Fig. 4.
10. In the "UDP Ports" section set the incoming and outgoing ports to 69 (standard TFTP port).
11. Enter the IP address of your PC in the "ip address" field.

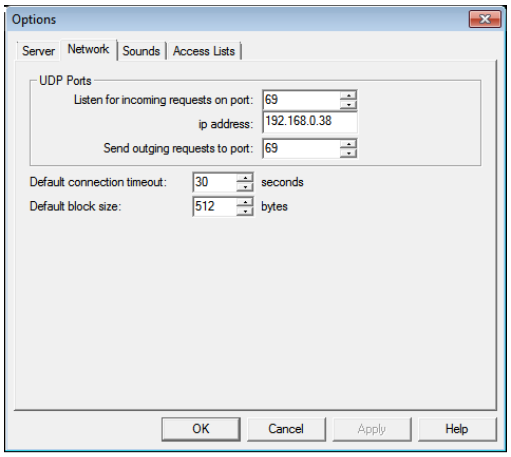

Options
Server
Network
Sounds
Access Lists
UDP Ports
Listen for incoming requests on port: 69
ip address: 192.168.0.38
Send outgoing requests to port: 69
Default connection timeout: 30 seconds
Default block size: 512 bytes
OK
Cancel
Apply
Help

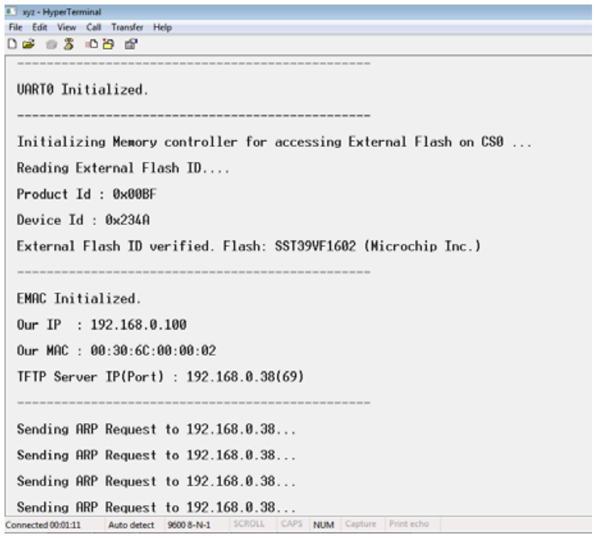
Fig.4. PumpKIN Software – Options Tab - Network

12. Set the "Default connection timeout" to 30 seconds.
13. Set the "Default block size" to 512 bytes.
14. Click on "OK" button. The TFTP Server is now configured to start listening for incoming requests.

booting

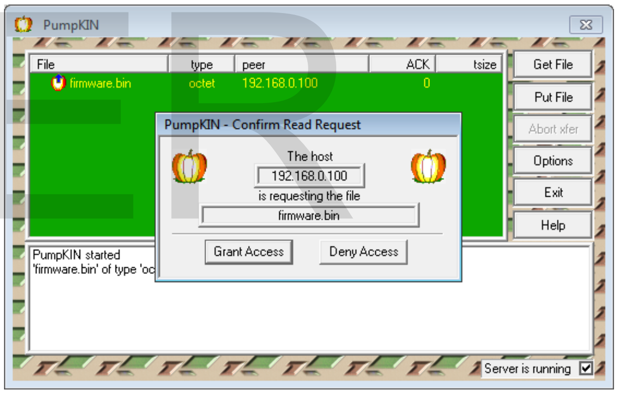
1. Generate the application code binary file (explained in next section), name it as "firmware.bin" and copy it into the "PumpKIN" TFTP server root directory.
2. Connect the board to an Ethernet port in the network using a LAN cable.
3. The boot loading progress is displayed on the "HyperTerminal" (Fig. 5).
4. When "PumpKIN" receives a read request from the boot loader it prompts the user for permission. It displays the IP address from which the request is received and the filename that is requested (Fig. 6).
5. On granting access the Server starts transferring the binary file to the boot loader. The connection status and the activity log are displayed in their respective windows (Fig. 7).
6. Similarly when "PumpKIN" receives a write request from the boot loader it prompts the user for permission. It displays the IP address from which the request is received and the filename that will be copied to the root directory (Fig. 8).
7. On granting access the Server starts accepting the binary file from the boot loader. The connection status and the activity log are displayed in the respective windows.
8. After uploading the Flash image to the "PumpKIN" TFTP Server, the boot loader jumps to the external flash start sector address and starts executing the application code. The boot loading progress is displayed in "HyperTerminal" (Fig. 9).
9. If there is no reply from Server the boot loader times out and jumps to the user code previously loaded in the external flash as shown in (Fig. 10).
10. The files "firmware.bin" and "flashIMG.bin" can be compared using a hex editor (as in [17]HxD) to verify that they are identical (Fig. 11).

The results of the files transferred can be viewed on the hyper terminal. The result will be as follows:



```
----------------------------------------------------

UART0 Initialized.

----------------------------------------------------

Initializing Memory controller for accessing External Flash on CS0 ...
Reading External Flash ID....
Product Id : 0x00BF
Device Id : 0x234A
External Flash ID verified. Flash: SST39VF1602 (Microchip Inc.)
----------------------------------------------------

EMAC Initialized.
Our IP  : 192.168.0.100
Our MAC : 00:30:6C:00:00:02
TFTP Server IP(Port) : 192.168.0.38(69)
----------------------------------------------------

Sending ARP Request to 192.168.0.38...
Sending ARP Request to 192.168.0.38...
Sending ARP Request to 192.168.0.38...
Sending ARP Request to 192.168.0.38...
```

Fig.5. Hyperterminal - UART0 Initialisation

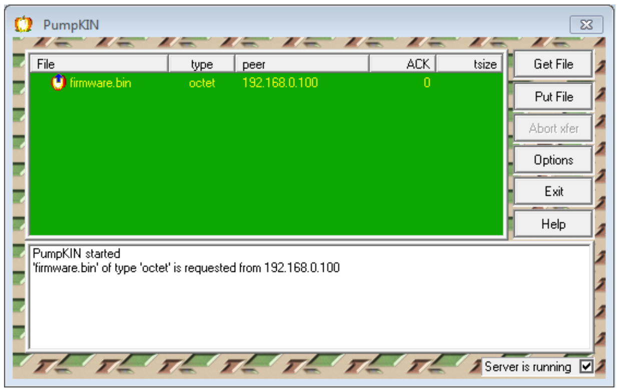


File	type	peer	ACK	tsize
firmware.bin	octet	192.168.0.100	0	

The host
192.168.0.100
is requesting the file
firmware.bin

PumpKIN started
'firmware.bin' of type 'oc

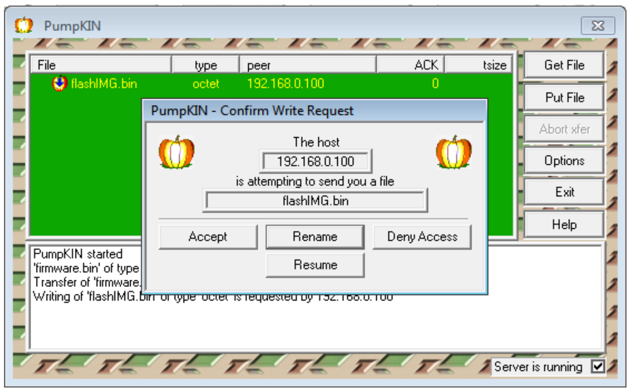
Fig.6. PumpKIN Window – Read Request Confirmation



File	type	peer	ACK	tsize
firmware.bin	octet	192.168.0.100	0	

PumpKIN started
'firmware.bin' of type 'octet' is requested from 192.168.0.100

Fig.7. PumpKIN Window – Transfer Begins

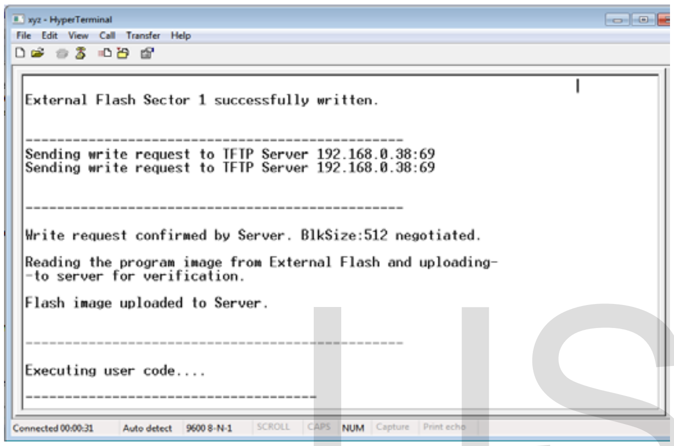


File	type	peer	ACK	tsize
flashIMG.bin	octet	192.168.0.100	0	

The host
192.168.0.100
is attempting to send you a file
flashIMG.bin

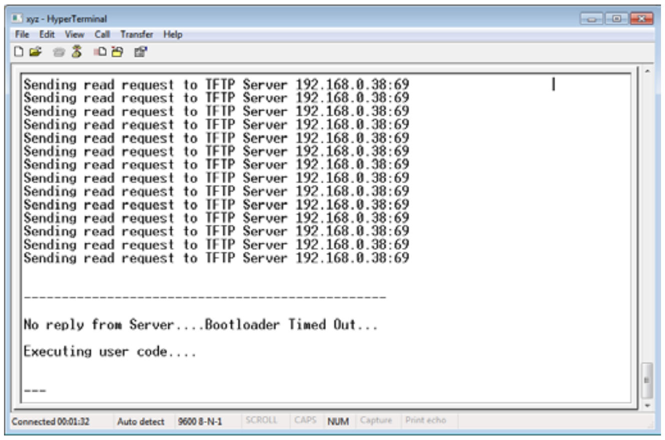
```
PumpKIN started
'firmware.bin' of type
Transfer of 'firmware.
Writing of 'flashIMG.bin' of type 'octet' is requested by 192.168.0.100
```

Fig.8. PumpKIN Window – Write Request Confirmation


xyz - HyperTerminal
File Edit View Call Transfer Help
Connected 00:00:31 Auto detect 9600 8-N-1 SCROLL CAPS NUM Capture Print echo

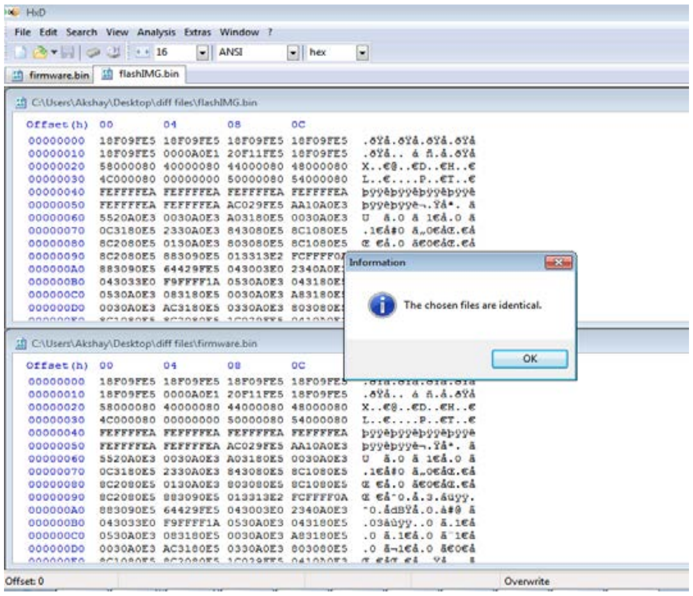
```
External Flash Sector 1 successfully written.

--------------------------------------------------
Sending write request to TFTP Server 192.168.0.38:69
Sending write request to TFTP Server 192.168.0.38:69


--------------------------------------------------
Write request confirmed by Server. BlkSize:512 negotiated.

Reading the program image from External Flash and uploading-
-to server for verification.

Flash image uploaded to Server.


--------------------------------------------------
Executing user code....

--------------------------------------------------
```

Fig.9. Hyperterminal – Report of Flash Write


xyz - HyperTerminal
File Edit View Call Transfer Help
Connected 00:01:32 Auto detect 9600 8-N-1 SCROLL CAPS NUM Capture Print echo

```
Sending read request to TFTP Server 192.168.0.38:69
Sending read request to TFTP Server 192.168.0.38:69
Sending read request to TFTP Server 192.168.0.38:69
Sending read request to TFTP Server 192.168.0.38:69
Sending read request to TFTP Server 192.168.0.38:69
Sending read request to TFTP Server 192.168.0.38:69
Sending read request to TFTP Server 192.168.0.38:69
Sending read request to TFTP Server 192.168.0.38:69
Sending read request to TFTP Server 192.168.0.38:69
Sending read request to TFTP Server 192.168.0.38:69
Sending read request to TFTP Server 192.168.0.38:69
Sending read request to TFTP Server 192.168.0.38:69
Sending read request to TFTP Server 192.168.0.38:69
Sending read request to TFTP Server 192.168.0.38:69


--------------------------------------------------
No reply from Server....Bootloader Timed Out...

Executing user code....


---
```

Fig.10. Hyperterminal - No Reply from Server



```
C:\Users\Akshay\Desktop\diff files\flashIMG.bin
Offset(h) 00       04       08       0C
00000000  18F09FE5 18F09FE5 18F09FE5 18F09FE5  .ðŸå.ðŸå.ðŸå.ðŸå
00000010  18F09FE5 0000A0E1 20F11FE5 18F09FE5  .ðŸå..  á ñ.å.ðŸå
00000020  58000080 40000080 44000080 48000080  X..€@..€D..€H..€
00000030  4C000080 00000000 50000080 54000080  L..€....P..€T..€
00000040  FEFFFFEA FEFFFFEA FEFFFFEA FEFFFFEA  þÿÿêþÿÿêþÿÿêþÿÿê
00000050  FEFFFFEA FEFFFFEA AC029FE5 AA10A0E3  þÿÿêþÿÿê¬.Ÿåª. ã
00000060  5520A0E3 0030A0E3 A03180E5 0030A0E3  U  ã.0 ã 1€å.0 ã
00000070  0C3180E5 2330A0E3 843080E5 8C1080E5  .1€å#0 ã„0€åŒ.€å
00000080  8C2080E5 0130A0E3 803080E5 8C1080E5  Œ €å.0 ã€0€åŒ.€å
00000090  8C2080E5 883090E5 013313E2 FCFFFF0A
000000A0  883090E5 64429FE5 043003E0 2340A0E3
000000B0  043033E0 F9FFFF1A 0530A0E3 043180E5
000000C0  0530A0E3 083180E5 0030A0E3 A83180E5
000000D0  0030A0E3 AC3180E5 0330A0E3 803080E5

The chosen files are identical.

C:\Users\Akshay\Desktop\diff files\firmware.bin
Offset(h) 00       04       08       0C
00000000  18F09FE5 18F09FE5 18F09FE5 18F09FE5
00000010  18F09FE5 0000A0E1 20F11FE5 18F09FE5  .ðŸå..  á ñ.å.ðŸå
00000020  58000080 40000080 44000080 48000080  X..€@..€D..€H..€
00000030  4C000080 00000000 50000080 54000080  L..€....P..€T..€
00000040  FEFFFFEA FEFFFFEA FEFFFFEA FEFFFFEA  þÿÿêþÿÿêþÿÿêþÿÿê
00000050  FEFFFFEA FEFFFFEA AC029FE5 AA10A0E3  þÿÿêþÿÿê¬.Ÿåª. ã
00000060  5520A0E3 0030A0E3 A03180E5 0030A0E3  U  ã.0 ã 1€å.0 ã
00000070  0C3180E5 2330A0E3 843080E5 8C1080E5  .1€å#0 ã„0€åŒ.€å
00000080  8C2080E5 0130A0E3 803080E5 8C1080E5  Œ €å.0 ã€0€åŒ.€å
00000090  8C2080E5 883090E5 013313E2 FCFFFF0A  Œ €å˜0.å.3.âüÿÿ.
000000A0  883090E5 64429FE5 043003E0 2340A0E3  ˆ0.ådBŸå.0.à#@ ã
000000B0  043033E0 F9FFFF1A 0530A0E3 043180E5  .03àùÿÿ..0 ã.1€å
000000C0  0530A0E3 083180E5 0030A0E3 A83180E5  .0 ã.1€å.0 ã¨1€å
000000D0  0030A0E3 AC3180E5 0330A0E3 803080E5  .0 ã¬1€å.0 ã€0€å
```

Fig.11. HDK Software - Identical files: 1.External Flash 2.Created by User

The part of the code for the impelementation of UDP for TFTP is given below:

```
void UDPLowLevelInit(void)
{
  Init_EMAC();
  TransmitControl = 0;

  print("\n------------------------------------------------\n");
  print ("\nEMAC Initialized. \n");
  print ("\nOur IP  : 192.168.0.100 \n");
  print ("\nOur MAC : 00:30:6C:00:00:02 \n");
  print ("\nTFTP Server IP(Port) : 192.168.0.38(69) \n ");
  print("\n------------------------------------------------\n");
}

/* This function reads the length of the received ethernet
frame   and checks if the destination address is a broadcast
message or not */

unsigned int IsBroadcast(void) {
  unsigned short RecdDestMAC[3];
 // 48-bit MAC   Address

  RecdFrameLength = StartReadFrame();

  CopyFromFrame_EMAC(&RecdDestMAC,  6);
 // Receive DA to see if it was a broadcast
  CopyFromFrame_EMAC(&RecdFrameMAC, 6);
 // Store SA (for our answer)

  if ((RecdDestMAC[0] == 0xFFFF) &&
  // Check if destination address is broadcast
      (RecdDestMAC[1] == 0xFFFF) &&
      (RecdDestMAC[2] == 0xFFFF)) {
    return(1);
  } else {
   return (0);
  }
}
    For Flash:

      EMC_STA_WAITRD0   = 0x1f;
      EMC_STA_WAITPAGE0 = 0x1f;
      EMC_STA_WAITWR0   = 0x1f;
      EMC_STA_WAITTURN0 = 0xf;

      delay(10000);

      print("\nReading External Flash ID....\n");
      if ( FLASHCheckID() == FALSE )
      {
        print ("\nFlash External ID Invalid.\n");
      }
  print ("\nExternal Flash ID verified. Flash: SST39VF1602
(Microchip Inc.)\n");  .
```

Main DataTransfer Code (part):

```
FLASHInit();
 /* Initialize EMC for accessing External Flash */

/* Intialize Remote IP */
  *(unsigned char *)RemoteIP = 192;
  *((unsigned char *)RemoteIP + 1) = 168;
  *((unsigned char *)RemoteIP + 2) = 0;
  *((unsigned char *)RemoteIP + 3) = 38;

/* Intialize Remote MAC */
  *(unsigned char *)RemoteMAC = 0x3C;
  *((unsigned char *)RemoteMAC + 1) = 0x4A;
  *((unsigned char *)RemoteMAC + 2) = 0x92;
  *((unsigned char *)RemoteMAC + 3) = 0xD5;
  *((unsigned char *)RemoteMAC + 4) = 0x29;
  *((unsigned char *)RemoteMAC + 5) = 0x84;

  UDPLowLevelInit();     /* Initialize EMAC for UDP */

  UDPLocalPort = 1200;   /* Set port we want to listen to
*/
  UDPRemotePort = 1024; /* Set port we want to Send to
*/

  while (1)
  {
   if (Arp_reply == 0)
/* If ARP reply not received yet */
     {
  /* Send ARP request to Server to obtain its MAC address
*/
       PrepareARP_REQUEST();

       /* Print on Hyperterminal */
       print ("\nSending ARP Request to 192.168.0.38... \n");
     }
```

6 CONCLUSION

Using the above procedure the external memory of the system can be used as the internal memory for booting up. The flash memory in this case contains the firmware for the system, which was supposed to be programmed in the internal flash. Hence, due to the wide space available for the firmware, firm wares of up to 4GB can be programmed in the external flash for execution.

The procedure uses all the free and evaluation version soft-wares. The process is hence cost effective and does not provide any overhead to the system with respect to finances. The results show that the data corruption check is also performed which confirms the programming of the correct firmware in the external memory.

The process can be modified and used according to the requirements for various processors and embedded systems. The paper describes important parts of the process for the use of external memory as the booting memory. Some code parts have been provided for quick references.

The TFTP and UDP are the common protocols used widely; hence they have been used to give a good scope for using the process to be used.

The prime benefit of this procedure is its cost effectiveness. The use of chips such as Smart Fusion cSoC by Micro Semi (as in [1]), JTAG connectors for programming is hence not necessary.

The paper provides an overlook on, an option for the secondary / external boot loader to be implemented in an easy, convenient and cost effective way.

7 REFERENCES

[1] Micro Semi Corporation , Application Note AC346, "SmartFusion cSoC: Loading and Booting from External Memories", pp. 11, Feb 2012
[2] Mohit Arora, and Varun Jain, "Understanding embedded-system-boot techniques", Freescale Semiconductors, Feb 2011. site:http://www.edn.com/design/systems-design/4363984/Understanding-embedded-system-boot-techniques-4363984
[3] Texas Instruments Incorporated, "Creating a Second-Level Bootloader for FLASH Bootloading on TMS320C6000 Platform with Code Composer Studio",Texas Instruments, May 2006.
[4] Zhang guolong, Xu xiaosu, "Implemention of Secondary Bootloader for Large-scale Embedded System", Computer Engineering, vol. 36, No.13, , pp 219-221, July 2010.
[5] Chen daiyuan, "External FLASH Memory's Bootloader System for C6000," Telecommunication Engineering, vol.49, No5, pp. 86-88, May, 2009.
[6] Daniel Allred, and Gaurav Agarwal, "Software and hardware design Challenges due to the dynamic raw NAND market", Texas Instruments, Apr 2011.
[7] Odd Jostein Svendsli, "Atmel's Self-Programming Flash Microcontrollers" , Atmel, March 2010.
[8] John Peatman, "Microcomputer based Design", published by Tata McGraw Hill, 2005 edition,pg. 365.
[9] Michael Barr, "Programming Embedded Systems in C and C++", Oreilly Publication, edition January 1999.

[10] Byte Craft, "First Steps with Embedded Systems", ByteCraft Limited, first edition.
[11] Andrew Sloss, Dominic Symes and Chris Wright "ARM System Developers Guide – Designing and Optimizing System Software" , published by ELSEVIER, 2009.
[12] Ted Van Sickle, "Programming Microcontrollers in C", LLH Technology Publishing, 2001.
[13] Michael Pont, "Embedded C", Pearson Publication, 2002.
[14] David Katz and Rick Gentile, "Fundamentals of Booting for Embedded Processors", Sept 2009.
Site:http://www.embedded.com/design/mcus-processors-and-socs/4008796/Fundamentals-of-Booting-for-Embedded-Processors
[15] NXP, "User Manual for ARM 7", NXP website.
[16] Website for PumpKIN: http://kin.klever.net/pumpkin
[17] Website for TFTP : http://tftpd32.jounin.net/
[18] Website for HXD software: http://mh-nexus.de/en/hxd/
[19] Website for Remote Server: http://tools.ietf.org/html/rfc1350
[20] Website for Remote Server: http://tools.ietf.org/html/rfc2347
[21] Website for Flash Magic: http://flashmagic.com