The following document contains information on Cypress products. Although the document is marked with the name "Spansion" and "Fujitsu", the company that originally developed the specification, Cypress will continue to offer these products to new and existing customers.

Continuity of Specifications
There is no change to this document as a result of offering the device as a Cypress product. Any changes that have been made are the result of normal document improvements and are noted in the document history page, where supported. Future revisions will occur when appropriate, and changes will be noted in a document history page.

Continuity of Ordering Part Numbers
Cypress continues to support existing part numbers. To order these products, please use only the Ordering Part Numbers listed in this document.

For More Information
Please contact your local sales office for additional information about Cypress products and solutions.

About Cypress
Cypress (NASDAQ: CY) delivers high-performance, high-quality solutions at the heart of today's most advanced embedded systems, from automotive, industrial and networking platforms to highly interactive consumer and mobile devices. With a broad, differentiated product portfolio that includes NOR flash memories, F-RAM™ and SRAM, Traveo™ microcontrollers, the industry's only PSoC® programmable system-on-chip solutions, analog and PMIC Power Management ICs, CapSense® capacitive touch-sensing controllers, and Wireless BLE Bluetooth® Low-Energy and USB connectivity solutions, Cypress is committed to providing its customers worldwide with consistent innovation, best-in-class support and exceptional system value.

Colophon

The products described in this document are designed, developed and manufactured as contemplated for general use, including without limitation, ordinary industrial use, general office use, personal use, and household use, but are not designed, developed and manufactured as contemplated (1) for any use that includes fatal risks or dangers that, unless extremely high safety is secured, could have a serious effect to the public, and could lead directly to death, personal injury, severe physical damage or other loss (i.e., nuclear reaction control in nuclear facility, aircraft flight control, air traffic control, mass transport control, medical life support system, missile launch control in weapon system), or (2) for any use where chance of failure is intolerable (i.e., submersible repeater and artificial satellite). Please note that Spansion will not be liable to you and/or any third party for any claims or damages arising in connection with above-mentioned uses of the products. Any semiconductor devices have an inherent chance of failure. You must protect against injury, damage or loss from such failures by incorporating safety design measures into your facility and equipment such as redundancy, fire protection, and prevention of over-current levels and other abnormal operating conditions. If any products described in this document represent goods or technologies subject to certain restrictions on export under the Foreign Exchange and Foreign Trade Law of Japan, the US Export Administration Regulations or the applicable laws of any other country, the prior authorization by the respective government entity will be required for export of those products.

Trademarks and Notice

The contents of this document are subject to change without notice. This document may contain information on a Spansion product under development by Spansion. Spansion reserves the right to change or discontinue work on any product without notice. The information in this document is provided as is without warranty or guarantee of any kind as to its accuracy, completeness, operability, fitness for particular purpose, merchantability, non-infringement of third-party rights, or any other warranty, express, implied, or statutory. Spansion assumes no liability for any damages of any kind arising out of the use of the information in this document.

FUJITSU

MCU-AN-510048-E-10

32-BIT MICROCONTROLLER

MB9B618T/S SERIES

ETHERNET BOOTLOADER MB9BF618T/S SOFTWARE

APPLICATION NOTE

Revision History

Version	Date	Updated by	Approved by	Modifications
0.1.0	2013-3-25	ChamberTeng		Initial version
0.2.0	2013-4-10	ChamberTeng		Add bootloader API functions
0.3.0	2013-4-22	QuinnXu		1. Update 1.1 overview, add update linkage for the Ethernet MAC driver 2.Update setup of HW in 6.3.3 3.Move the API introduction into appendix B 4.Add note in 6.4 5.Add AN number(MCU-AN-510048-E-10) on the coverage
0.4.0	2013-4-23	ChamberTeng		Delete function “Tftp_CalcSquare()” in sector 7

Contents

REVISION HISTORY **2**
CONTENTS **3**
1 INTRODUCTION **5**
1.1 Overview 5
1.2 Reference Documents 5
2 HARDWARE ENVIRONMENT **6**
3 DEVELOPMENT ENVIRONMENT **7**
4 FEATURES **8**
5 BOOTLOADER OPERATIONS **9**
5.1 Project Structure 9
5.2 Main loop 9
5.3 Memory map 11
5.4 Jump into user application 13
5.5 Linker settings 14
5.5.1 Bootloader linker settings 14
5.5.2 User application code linker settings 14
5.6 User application code version number settings 15
6 QUICK START USING FSSDC-9B618-EVB V1.0 ..... **16**
6.1 Download Bootloader to Flash 16
6.1.1 Settings for USB On-Board Programming 16
6.2 Configure User Application Code 18
6.3 Download User Code to Flash 18
6.3.1 Enter Ethernet bootloader mode 18
6.3.2 Configure TFTP Client Settings 19
6.3.3 Flash Empty 19
6.3.4 Flash with User Code 20
6.4 Read User Code in Flash 21
6.5 Erase User Code in Flash 21
7 BOOTLOADER API FUNCTIONS **22**
APPENDIX A MORE INFORMATION **24**
APPENDIX B API LIST **25**
Bootloader module 25
Bootloader_ReadFlashSecurityBit 25
Bootloader_JumpBoot 25
Bootloader_ExecuteUserApplication 25
Bootloader_UserCodeValid 26

Bootloader_CheckEmpty 26
Bootloader_EraseChip 27
Bootloader_ExecuteSoftwareReset 27
Bootloader_JudgeRunMode 27
Bootloader_ConfigureEthernetPins 28
Bootloader_EthernetIAPInit 28
LED_Init28
Flash module 29
FlashRomEraseSector 29
FlashDataPolling 29
FlashRomProgram 30
Tftp server module 30
Tftp_DecompressPktLenth 30
Tftp_SendBaseMessage 31
Tftp_SendAckPkt 31
Tftp_SendErrorPkt 32
Tftp_SendDatagramPkt 32
Tftp_RRQRecvedCallback 33
Tftp_WRQRecvedCallback 33
Tftp_WRQRecved 34
Tftp_RRQRecved 34
Tftp_RecvCallback 35
Tftp_server_init 36

1 Introduction

1.1 Overview

This document describes the implementation of an Ethernet boot loader which is based LwIP1.4.0 middleware and Fujitsu Ethernet driver library. Ethernet boot loader can be used to update user applications per Ethernet network.

This solution is designed primarily for Fujitsu microcontroller with existing evaluation boards.

The functions include:

-Supports TCP/IP stack, such as ICMP, ARP, IP, UDP (provided by LwIP1.4.0)

-Supports Ethernet MAC driver (provided by mb9bf61xt_ethernet_lwip library, the update can be found from http://mcu.emea.fujitsu.com/mcu_product/mcu_all_software.htm)

-Supports static IP address assignment

-Optional flash security function

-Supports configuration of user code area

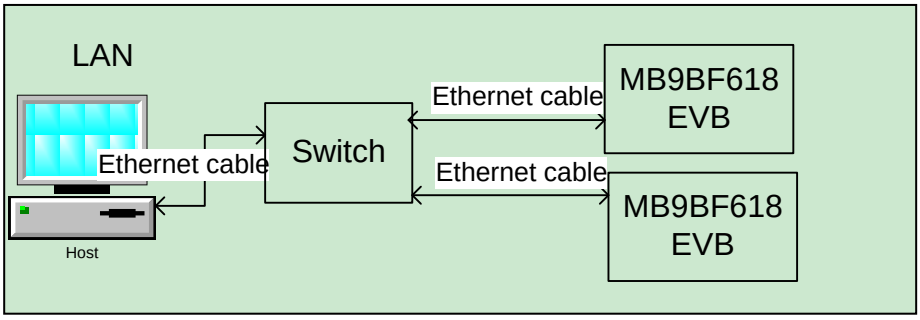

Figure 1 Interconnections

1.2 Reference Documents

[1].MB9Bxxx-MN706-00002-4v0-E.pdf
[2].MB9BF210T_610T-MN706-00015-1v0-E.pdf
[3].Design and Implementation of the LWIP TCP/IP Stack, Feb, 2001, Adam Dunkels.
[4].MB9Bx10T-MN706-00014-1v0-E.pdf
[5].MCU-AN-300411-E-V10-fm3_usb_host_masstorage_bootloader.pdf
[6].FSS-MB9B618S-EV-Board-User-Manual.pdf

2 Hardware Environment

Hardware board

- FSSDC-9B618-EVB v1.0

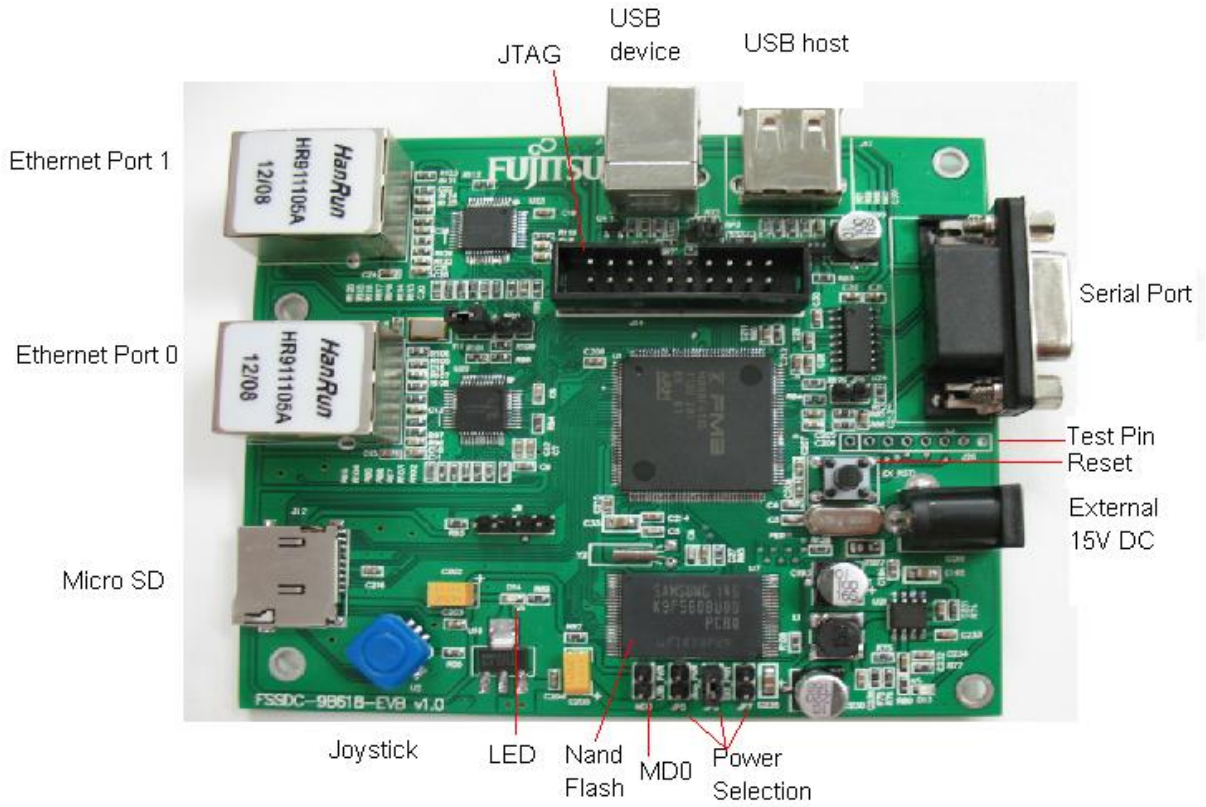

Figure 2 FSSDC-9B618-EVB v1.0

MCU: Fujitsu MB9BF618S

- MCU Frequency: 144MHz
- Ram Space: 128 K bytes
- Code Space: 1 M bytes

3 Development Environment

Name	Description	Part Number	Manufacturer	Remark
IAR EWARM	Software Developing IDE	V6.50	IAR	
J-link	MCU Emulator	J-link	IAR	

4 Features

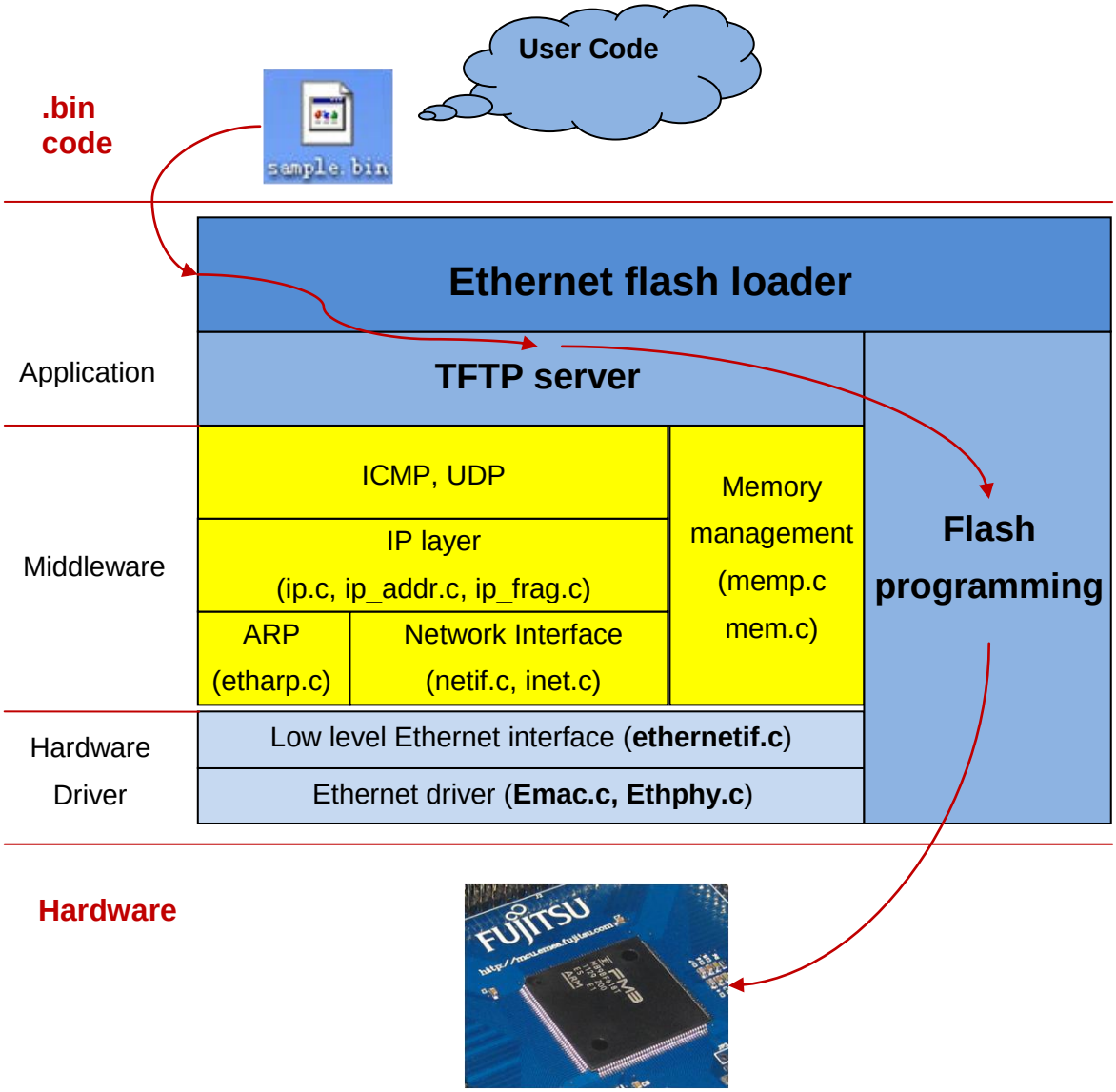

Figure 3 project block diagram

- Hardware Driver— FM3 Ethernet driver
- Middleware— LwIP 1.4.0
- Application— TFTP server
- Flash programming— MCU internal flash
- Ethernet flash loader— contains TFTP server and flash programming driver

5 Bootloader operations

5.1 Project Structure

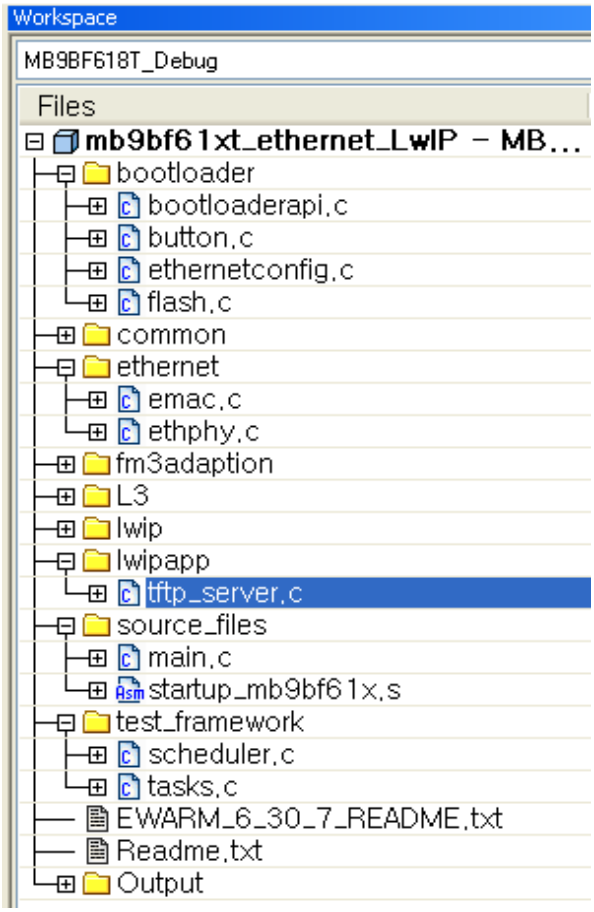

Figure 4 project structure

5.2 Main loop

Before initializing boot loader, the main loop checks the following items:

- Specified flash validation
- Start-up key button status

Figure 5 is the software procedure of running mode selection.

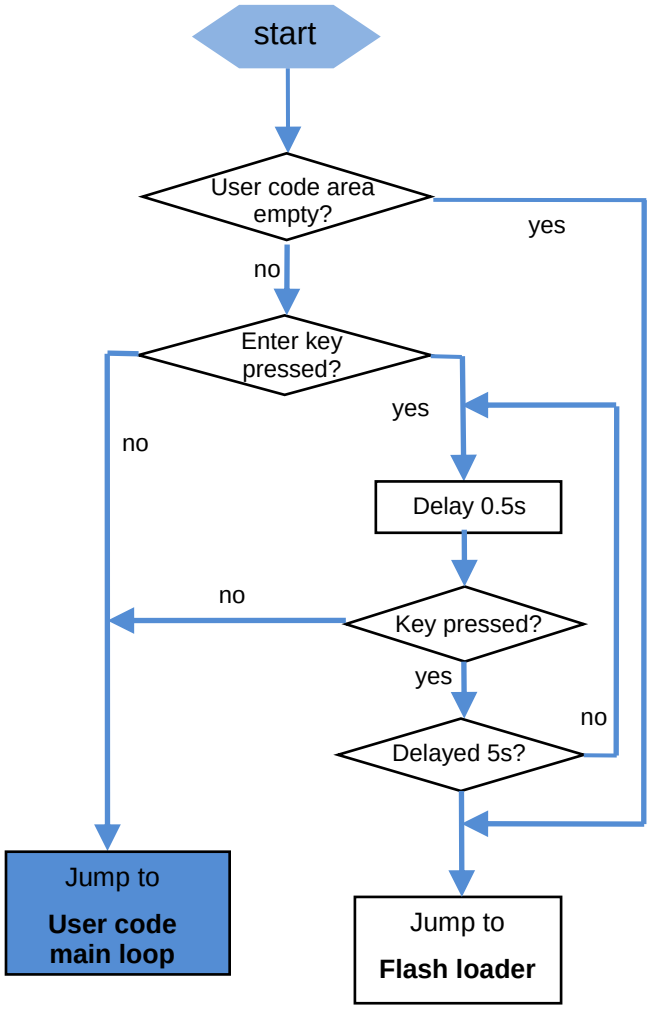

Figure 5 run mode judgement

5.3 Memory map

As described in the Technical Reference Manual of ARM for Cortex-M3 core, ARM Cortex M3 microcontrollers have its flash memory starting at 0x0000_0000, RAM memory starting at 0x2000_0000 and so on.

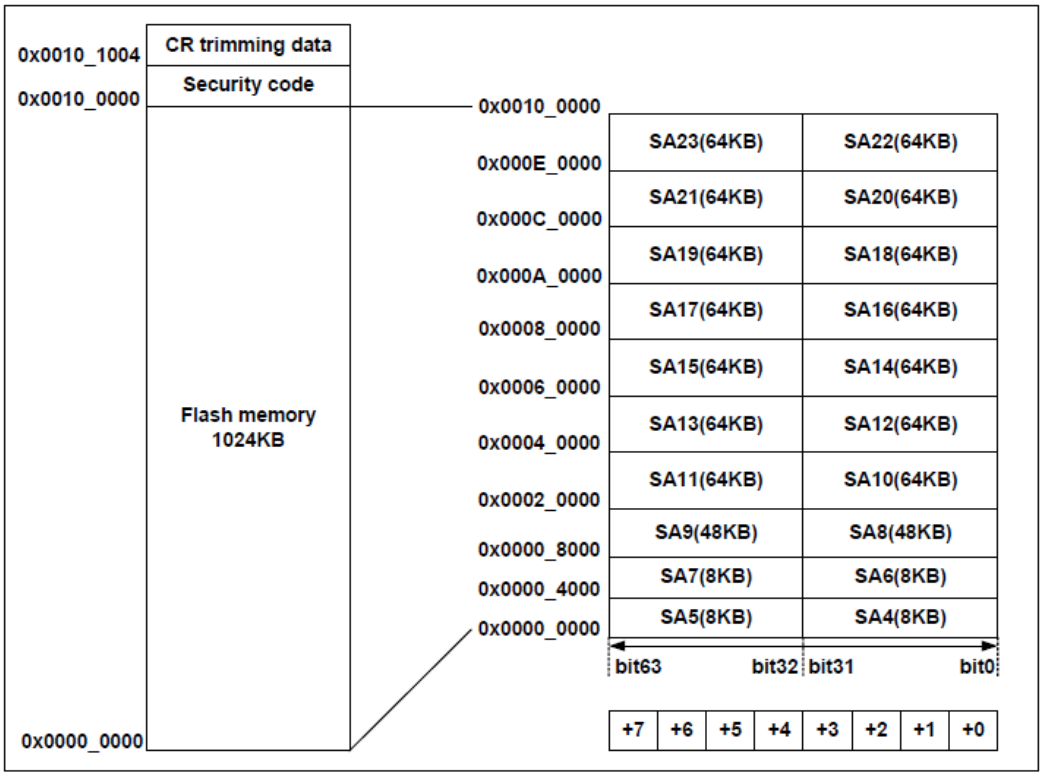

Figure 6 M9BF618S flash memory map

As defined for ARM Cortex-M3 MCUs, the vector table starts with the stack pointer followed by the reset vector and so on.
The vector table of the bootloader is fixed to 0x0000_0000 and the user code must be linked to upper areas of the flash memories. For Fujitsu Ethernet bootloader, the user code starts from 0x0002_0000.

The code allocation of this demo as follows:

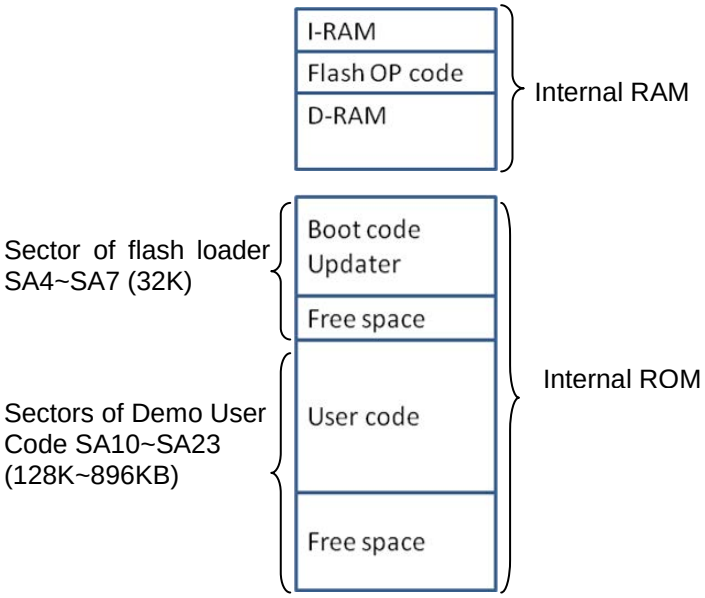

Figure 7 code allocation

The Ethernet loader code of MCU will be allocated at the SA4~SA7.

The user application program will be allocated at SA10~SA23, whose vector table shall be placed at the beginning of the sector.

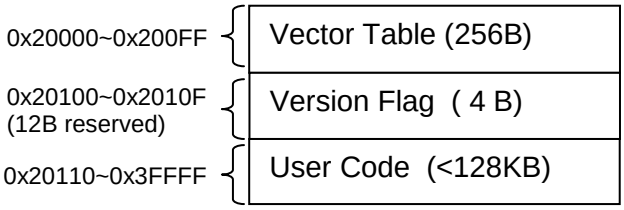

Figure 8 sample memory (Maximum: 128KB)

Maximum code size can be defined by customer, using settings in bootloader\ bootloader.h. The maximum size can reach up to 896KB.

5.4 Jump into user application

To start the user application code, the stack pointer and vector table must be updated. Following procedure is called, to start the user application code, linked to 0x0002_0000:

```
#define USER_FLASH_START    0x00020000

/**
*************************************************************************
*******
** \brief Jump to user code application
**
** \param u32Address Address of user code
** \return none
*************************************************************************
*******/
#ifdef __ICCARM__
void BootloaderAPI_JumpBoot(uint32_t u32Address)
{
    __asm("LDR SP, [R0]");     //Load new stack pointer address
    __asm("LDR PC, [R0, #4]"); //Load new program counter address
}
#elif __CC_ARM
__asm void BootloaderAPI_JumpBoot(uint32_t u32Address)
{
    LDR SP, [R0]     ;Load new stack pointer address
    LDR PC, [R0, #4] ;Load new program counter address
}
#else
#error

/**
*************************************************************************
*******
** \brief Execute main application
**
** \param none
** \return none
*************************************************************************
*******/
void BootloaderAPI_ExecuteUserApplication(void)
{
    //Change the Vector Table to the USER_FLASH_START
    SCB->VTOR = USER_FLASH_START & 0x1FFFFF80;
    BootloaderAPI_JumpBoot(USER_FLASH_START);
}
```

5.5 Linker settings

For using bootloader, linker settings must be changed, including:

- Bootloader project linker settings
  It is configured in project, no need to change.
- User application project linker settings
  This one must be set in the user application projects.

5.5.1 Bootloader linker settings

When writing data into internal flash, flash operations must be copied into RAM to process.

This can be configured in the linker files normally done by the IDE start-up code. The ROM (flash) area should be set to use only 0x0000_0000 to 0x0000_7FFF.

To copy Flash erase/programming routines to RAM automatically after start-up and be executed later in RAM, the following lines should be added to the linker file (*.icf):

```
define symbol __RAM_func_start__ = 0x20000000;
define symbol __RAM_func_end__   = 0x20007FFF;
define region RAM_func_region    = mem:[from __RAM_func_start__
to __RAM_func_end__];
define block RamCode {section .flash_ram_code};
place in RAM_func_region { block RamCode };
```

5.5.2 User application code linker settings

To coordinate the application start address in bootloader project, the following lines should be re added in the standard FM3 template linker files (*.icf):

```
/*-Specials-*/
define symbol __ICFEDIT_intvec_start__ = 0x00020000;
/*-Memory Regions-*/
define symbol __ICFEDIT_region_ROM_start__ = 0x00020000;
```

This can be done also via Project -> Options -> Linker,where .intvec start and ROM start can be specified as well.

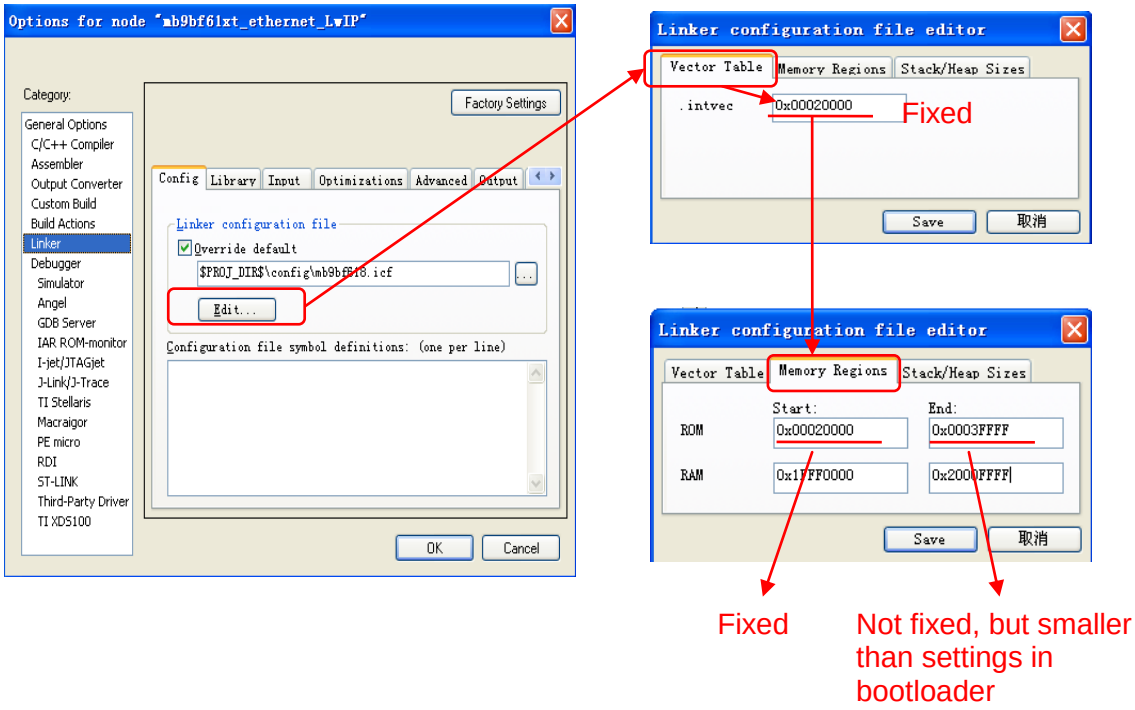

Figure 9 user code project settings

5.6 User application code version number settings

Step 1: Define version information array

```
#pragma location = ".INFORSEC" //.INFORSEC
const char sVersion[16] = {'V','1','.','0',0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
```

Step 2: Quote version information array in main() function

```
char *addr = (char *)sVersion;
```

Step 3: Add memory placement command to *.icf file.

```
place in ROM_region   { readonly };
place in RAM_region   { readwrite,
                        block CSTACK, block HEAP };
place at address mem:0x00020100 { section .INFORSEC };
```

This command is used to locate variables of sVersion to 0x0002_0100.

6 Quick start using FSSDC-9B618-EVB v1.0

6.1 Download Bootloader to Flash

To use Ethernet bootloader, customer must download the bootloader code to MCU flash.

Demo board FSSDC-9B618-EVB v1.0 support On-Board Programming via USB. This is convenient to download bootloader code to flash.

6.1.1 Settings for USB On-Board Programming

- Hardware settings below:

Connector	Function	Setting
MD0	Mode setting	short
JP9	USB programming	short

- When connecting with PC via USB cable, the EVB can be identified as a USB device after power on.

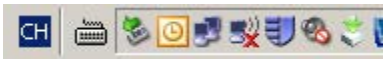

Figure 10 USB device Sign

- Check the COM port for this USB device in the device manager.

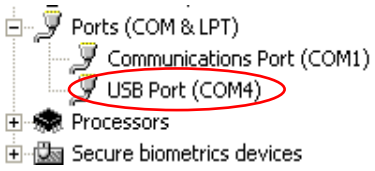

Figure 11 COM Port in Device Manager

- Install the USB programmer: usbdirect-v01l07.zip. (It can be downloaded from Fujitsu official website)
- Run the software and set the parameter as shown in following figures, and select Hex file. Click "**Full Operation**" button.

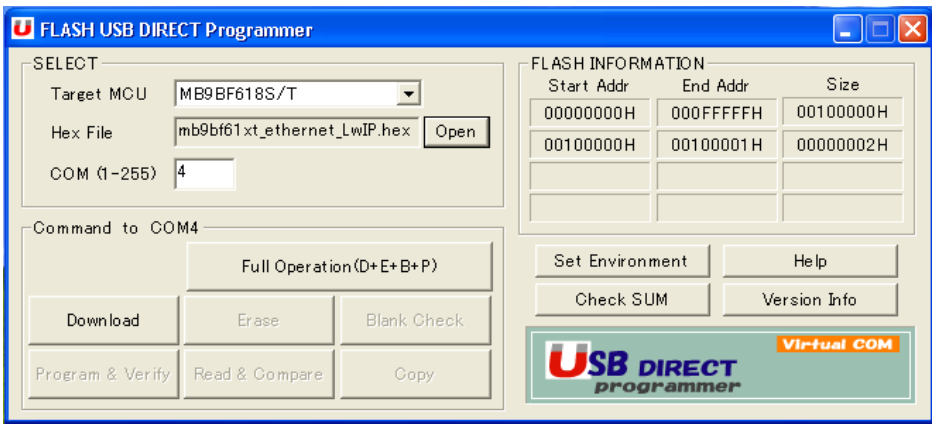

Figure 12 settings of USB programmer

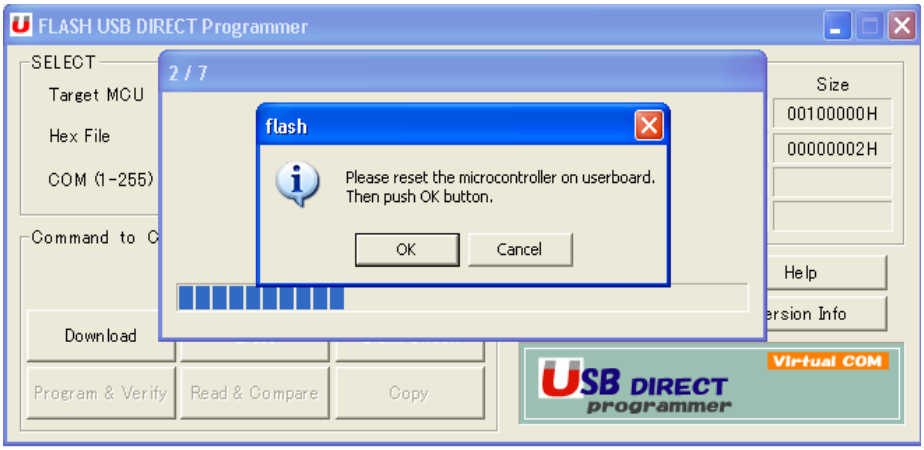

Figure 13 reset message during programming

- Reset EVB first and then press “OK” button in USB programmer

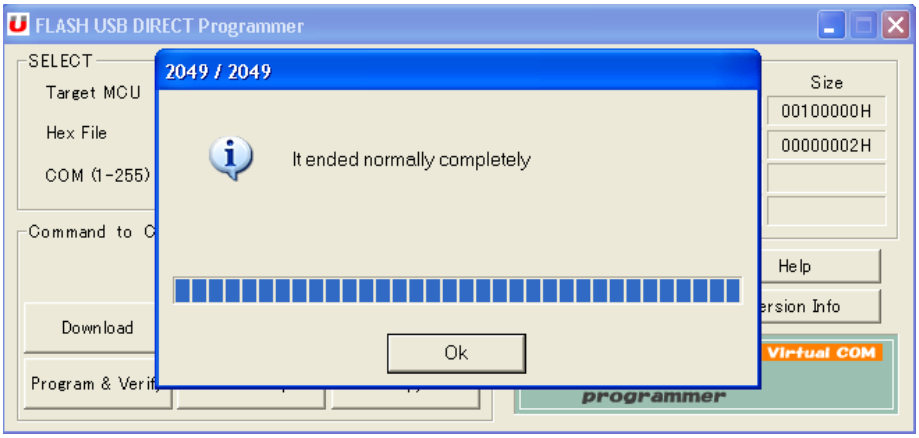

Figure 14 programming complete successfully

Please refer to FSS-MB9B618S-EV-Board-User-Manual.pdf for the details.

6.2 Configure User Application Code

Customer must set user application code as mentioned in sector 5.5.2 and 5.6.

Set user application code linker settings.

>Vector Start Address: set to 0x0002_0000

>Memory Regions: set Rom Start address to 0x0002_0000

>Rom End address depends on the code size

Then, configure user code version information as described in sector 5.6 and build the user code into binary file.

6.3 Download User Code to Flash

6.3.1 Enter Ethernet bootloader mode

If user code flash area is empty (no user code), the MCU will enter bootloader mode directly after power on. And LED “D14” will turn ON when running in bootloader mode.

If user code exits in the specified flash area, MCU may execute user application code directly. At this condition, we have two methods to enter bootloader mode.

Method 1:

Press joystick OK button (middle button of the joystick) without release, then power on the demo board. After 5 seconds, LED “D14” will turn on indicating demo board enters bootloader mode. Then release the button.

Method 2:

When demo board is power on, the existing user application will be executed. Press the joystick OK button without release. Then press reset key (EX_RST1) to reset MCU. After 5 seconds, the D14 turns on indicating demo board enters bootloader mode. Then release the button.Then release the button.

6.3.2 Configure TFTP Client Settings

We need a TFTP client to transmit user application code to internal flash.

For this demo, we choose 3CDaemon.exe to serve as a TFTP client.

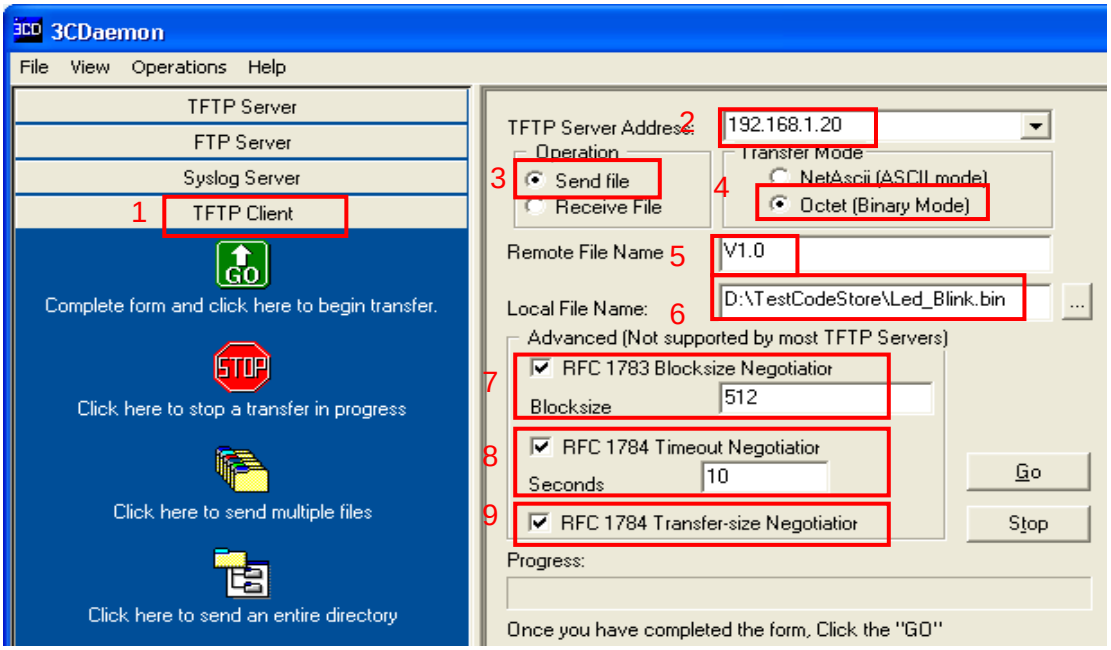

Figure 15 TFTP client settings

1. Set to TFTP client mode
2. Set TFTP Server Address to "192.168.1.20"
3. Set Operation mode:
   >Write flash: choose Send file
   >Read flash: choose Receive file
4. Set Transfer Mode to "Octet (Binary Mode)"
5. Fill Remote File Name with version information
   >Version Format: " Vx.x " (example: V1.0/ V1.1/ V2.0/ V2.3)
   Please control the version number and format as the examples.
6. Set Local File to the target test code routine, and select the Test_LED.bin file
7. Select RFC 1783 Blocksize Negotiation
8. Select RFC 1784 Timeout Negotiation and set seconds to 10
9. Select RFC 1784 Transfer-size Negotiation.

6.3.3 Flash Empty

To describe execute steps clear, several operations will be shown below.

Before running, customer must confirm user application code project's linker settings and version number settings are correct. Compile and build the binary file (.BIN). Any unclear, please refer to chapter 5.5 and chapter 5.6.

After download bootloader code into internal flash, specified user application code area is default empty.

Power on the demo board, system will enter bootloader mode directly.

1. Configure TFTP client settings as Figure 10
2. Power on the EVB with Ethernet cable plugged into Ethernet Port0 (J1)
3. After PC assigned IP address, press 3CDaemon “Go” button to download user code.

After datagram transferred successfully, demo board will execute the sample code and 3CDaemon will display like below:

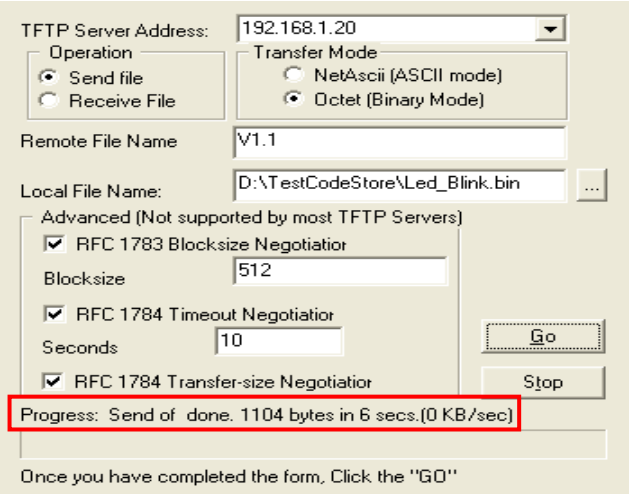

Figure 16 data transferred successfully

6.3.4 Flash with User Code

This part will describe the condition when user code is already in flash.

Use the steps in chapter 6.2.1 to enter bootloader mode.

Please confirm typing correct commands in “Remote File Name” of 3CDaemon window.

Version number should be same with the settings in user code project (refer to chapter 5.6).

If version is lower or the same as the code in flash already. TFTP client will display error command as below:

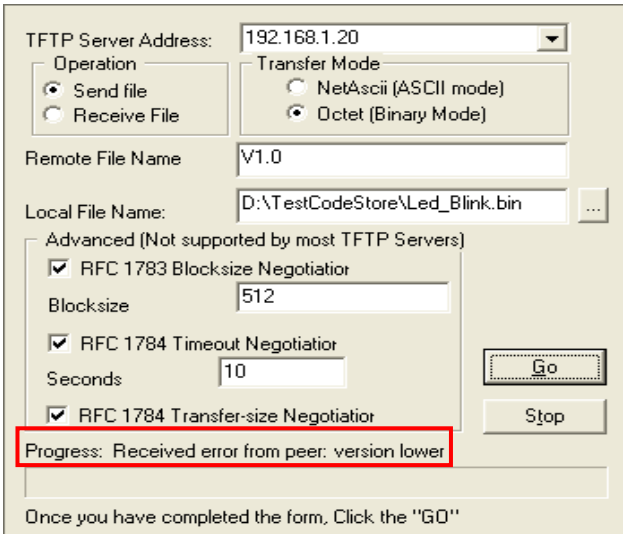

Figure 17 code version number lower

If code size is bigger than the defined size in bootloader, 3CDaemon will display error command “Progress: Received error from peer: code oversize.”

If format of version number is not correct, 3CDaemon will display “Progress: version format error”

6.4 Read User Code in Flash

All steps are the same as flash empty section (6.3.3) except changing operation mode to “Receive File” in 3CDaemon.

If Flash is secured (address 0x0010_0000 is written with “0x0001”) , as well as the macro definition of ***ENABLE_FLASH_SECURITY*** (in bootloader.h) is enabled, the user code can’t be read out. And 3CDaemon will display error message” Progress: Received error from peer: flash security secured, can't be read. ”

Note:

The only way to make the use code readable is by erasing the security bit of MCU flash with Fujitsu USB on board program tool (chip erase) and then flash the use code again.

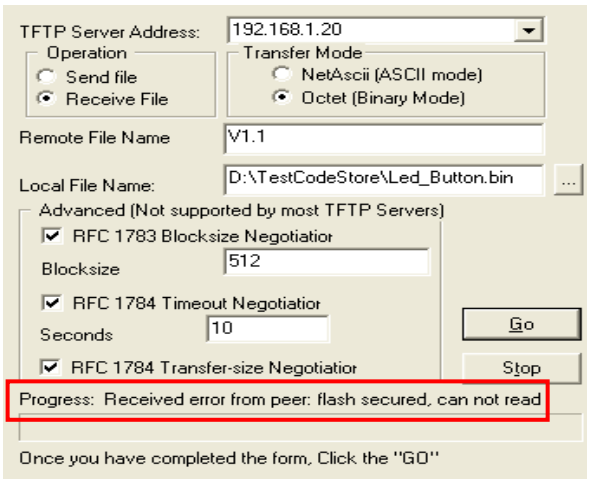

Figure 18 flash secured

6.5 Erase User Code in Flash

This demo supports erase user code flash area through using special command. Configure 3CDaemon.exe settings as figure 14.

Type command “Erase” in Remote File Name. Then press “Go” button.

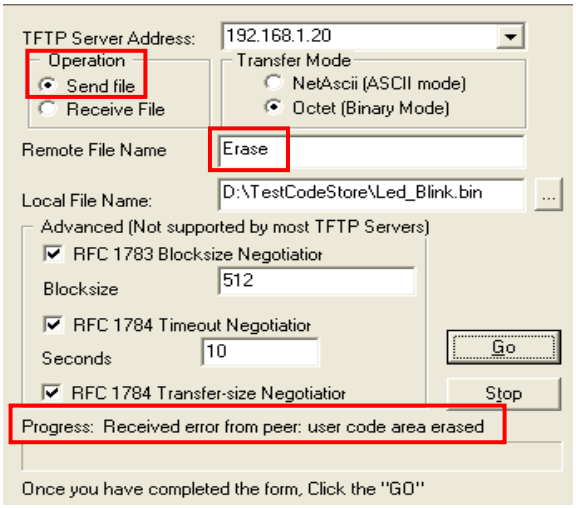

Figure 19 erase user code area

After user code area erased, 3CDaemon will display “Progress: Received error from peer: user code area erased.”

Notes:

This command will erase all flash sectors of user applications after 0x00020000.

7 Bootloader API functions

Refer to Appendix B for more details:

1. Bootloader module

- uint16_t Bootloader_ReadFlashSecurityBit(void);
- void Bootloader_JumpBoot(uint32_t u32Address);
- void Bootloader_ExecuteUserApplication(void);
- boolean_t Bootloader_UserCodeValid(void);
- boolean_t Bootloader_CheckEmpty(void);
- void Bootloader_EraseChip(void);
- void Bootloader_ExecuteSoftwareReset(void);
- en_run_mode_t Bootloader_JudgeRunMode(void);
- static void ConfigureEthernetPins(void);
- int32_t Bootloader_EthernetIAPInit(void);
- static void LED_Init(void);

2. Flash module

- int32_t FlashRomEraseSector(uint32_t u32SectorEraseAddress);
- int32_t FlashDataPolling (uint32_t u32PollAddress, uint16_t u16PollData);
- int32_t FlashRomProgram(uint32_t u32ProgramAddress, uint16_t u16ProgamData) ;

4. Tftp server module

- uint32_t Tftp_DecompressPktLenth(uint8_t *pu8PktBuf);
- static void Tftp_SendBaseMessage(struct udp_pcb *upcb, struct ip_addr *addr,uint16_t port, uint8_t *buf,uint16_t buflen)
- void Tftp_SendAckPkt(struct udp_pcb *upcb, struct ip_addr * addr, uint16_t port, uint16_t block)
- void Tftp_SendErrorPkt(struct udp_pcb *upcb, struct ip_addr * addr, uint16_t port, uint8_t u8ErrorType)
- void Tftp_SendDatagramPkt(struct udp_pcb *upcb, struct ip_addr * addr, uint16_t port, uint16_t block,uint8_t *data,uint16_t datalenth)
- void Tftp_RRQRecvedCallback(void *args, struct udp_pcb *upcb, struct pbuf *pkt_buf, struct ip_addr *addr, u16_t port)
- void Tftp_WRQRecvedCallback(void *args, struct udp_pcb *upcb, struct pbuf *pkt_buf, struct ip_addr *addr, u16_t port)
- void Tftp_WRQRecved(struct udp_pcb *upcb, struct ip_addr *IpAddr, uint16_t port,struct pbuf *pkt_buf)
- void Tftp_RRQRecved(struct udp_pcb *upcb, struct ip_addr *IpAddr, uint16_t port, struct pbuf *pkt_buf)
- void Tftp_RecvCallback(void *arg, struct udp_pcb *upcb, struct pbuf *pkt_buf, struct ip_addr *addr, u16_t port)
- void Tftp_server_init(void)

Appendix A More Information

For more Information on FUJITSU semiconductor products, visit the following websites:

English version address:

http://www.fujitsu.com/global/services/microelectronics/product/micom/support/sample/fm3.html

http://mcu.emea.fujitsu.com/mcu_product/mcu_all_software.htm

http://www.fujitsu.com/cn/fss/mcu/lineup/fm3/

Appendix B API List

Bootloader module

Bootloader_ReadFlashSecurityBit

Read flash security statement

Prototype:

uint16_t Bootloader_ReadFlashSecurityBit (void);

Para:

None

Return:

uint16_t: the stored value of flash security bit

Describe:

Note:

Bootloader_JumpBoot

Jump to user code application

Prototype:

void Bootloader_JumpBoot(uint32_t u32Address);

Parameter:

u32Address: Address of user code

Return:

None

Describe:

Note:

Bootloader_ExecuteUserApplication

Execute user application code

Prototype:

void Bootloader_ExecuteUserApplication(void);

Parameter:

None

Return:

None

Describe:

Note:

Bootloader_UserCodeValid

Judge whether the user code is valid

Prototype:

boolean_t Bootloader_UserCodeValid(void);

Parameter:

None

Return:

Bootlean_t:

TRUE: user code is valid

FALSE : user code is not valid

Describe:

This function will judge whether the bootloader user code area is empty and user code is valid.

Note:

Bootloader_CheckEmpty

Check whether user code area is empty

Prototype:

boolean_t Bootloader_CheckEmpty(void);

Parameter:

None

Return:

TRUE : user code area is empty

FALSE : user code area is not empty

Describe:

This function will judge user code area. And return memory statement

Note:

Bootloader_EraseChip

Erase user code area flash memory

Prototype:

void Bootloader_EraseChip(void);

Parameter:

None

Return:

None

Describe:

This function will erase user code area directly

Note:

Bootloader_ExecuteSoftwareReset

Software reset the code

Prototype:

void Bootloader_ExecuteSoftwareReset (void);

Parameter:

None

Return:

None

Describe:

Software reset the code

Note:

Bootloader_JudgeRunMode

Judge which mode to run

Prototype:

en_run_mode_t Bootloader_JudgeRunMode(void);

Parameter:

None

Return:

en_run_mode_t:

USER_MODE:

BOOTLOADER_MODE:

Describe:

Judge which mode to run. If return BOOTLOADER_MODE, the program will run bootloader code, else run the user application code

Note:

Bootloader_ConfigureEthernetPins

Configure general IO ports for Ethernet pins

Prototype:

void Bootloader_ConfigureEthernetPins(void);

Parameter:

None

Return:

None

Describe:

Configure the IO ports for Ethernet EMAC0/EMAC1

Note:

Bootloader_EthernetIAPInit

Initial Ethernet IAP function

Prototype:

int32_t Bootloader_EthernetIAPInit(void)

Parameter:

None

Return:

int32_t

Describe:

Initial Ethernet IAP

Note:

LED_Init

Initial IO ports for LED control

Prototype:

void LED_Init (void);

Parameter:

None

Return:

None

Describe:

Initial IO ports for LED state control

Note:

Flash module

FlashRomEraseSector

Initial Ethernet IAP function

Prototype:

int32_t FlashRomEraseSector(uint32_t u32SectorEraseAddress)

Parameter:

u32SectorEraseAddress :

flash sector address need to be erase

Return:

int32_t :

FLASH_OK

FLASH_ERROR

Describe:

Erase one flash sector

Note:

FlashDataPolling

Data Polling alogithm

Prototype:

int32_t FlashDataPolling (uint32_t u32PollAddress, uint16_t u16PollData)

Parameter:

u32PollAddress: Polling address

u16PollData : Polling data

Return:

int32_t :

FLASH_OK : flash polling OK

FLASH_TIMEOUT : flash polling error

Describe:

A flash data polling alogithm

Note:

FlashRomProgram

Write data to specified flash area

Prototype:

int32_t FlashRomProgram(uint32_t u32ProgramAddress, uint16_t u16ProgamData)

Parameter:

u32ProgramAddress: target address

u16ProgamData : specified data

Return:

int32_t :

FLASH_OK : flash polling OK

FLASH_TIMEOUT : flash polling error

Describe:

This function is mainly at write data to flash memory

Note:

1. *It only can write half-word(16 bit) data to flash memory*
2. *u32ProgramAddress must be even value*

Tftp server module

Tftp_DecompressPktLenth

Decompress total packet length of Tftp trans file

Prototype:

uint32_t Tftp_DecompressPktLenth(uint8_t *pu8PktBuf)

Parameter:

pu8PktBuf: datagram packet buffer

Return:

uint32_t: datagram packet total length

Describe:

This function can decompress total length of file

Note: *This decompress function is only for WRQ request*

Tftp_SendBaseMessage

Send basic data packet using UDP protocol

Prototype:

void Tftp_SendBaseMessage(struct udp_pcb *upcb,
struct ip_addr *addr,
uint16_t port,
uint8_t *buf,
uint16_t buflen))

Parameter:

upcb*: udp pcb to be send*

addr : ip address

port : port number need to be bind

buf : datagram packet buffer

buflen : datagram packet buffer length

Return:

None

Describe:

Note:

Tftp_SendAckPkt

Sendback Ack to Tftp client

Prototype:

void Tftp_SendAckPkt (struct udp_pcb *upcb,
struct ip_addr *addr,
uint16_t port,
uint16_t block)

Parameter:

upcb: udp pcb to be send

addr: ip address

port: port number need to be bind

block: Ack block number

Return:

None

Describe:

Tftp server send Ack packet to client for feedback

Note:

Tftp_SendErrorPkt

Sendback ERROR packet to Tftp client

Prototype:

void Tftp_SendErrorPkt (struct udp_pcb *upcb,
struct ip_addr *addr,
uint16_t port,
uint8_t u8ErrorType)

Parameter:

upcb: udp pcb to be send

addr: ip address

port: port number need to be bind

u8ErrorType:

ERRORTYPE_SECURITY_ENABLED ;
ERRORTYPE_VERSION_LOWER
ERRORTYPE_VERSION_NOTSET
ERRORTYPE_CODESIZE_OVERFLOW
ERRORTYPE_ERASE_CHIP
ERRORTYPE_FLASHWRITE_ERROR

Return:

None

Describe:

Tftp server send Error packet to client for feedback

Note:

Tftp_SendDatagramPkt

Sendback Ack to Tftp client

Prototype:

void Tftp_SendDatagramPkt (struct udp_pcb *upcb,
struct ip_addr *addr,
uint16_t port,
uint16_t block,
uint8_t *data,
uint16_t datalenth)

Parameter:

upcb: udp pcb to be send

addr: ip address

port: port number need to be bind

block: Ack block number

data : data buffer

datalenth : length of data buffer

Return:

None

Describe:

Tftp server send datagram packet to client, and it can control datagram length

Note:

Tftp_RRQRecvedCallback

Callback function when RRQ request received

Prototype:

void Tftp_RRQRecvedCallback(void *args,
struct udp_pcb *upcb,
struct pbuf *pkt_buf,
struct ip_addr *addr,
uint16_t u16_t port,)

Parameter:

Args: user supplied argument to match the callback function

upcb*: udp pcb to be send*

pkt_buf: buffer stored with data packet

port: port number need to be bind

Return:

None

Describe:

This is a callback function when Tftp server received RRQ request

Note:

Tftp_WRQRecvedCallback

Callback function when WRQ request received

Prototype:

void Tftp_WRQRecvedCallback(void *args,

struct udp_pcb *upcb,
struct pbuf *pkt_buf,
struct ip_addr *addr,
uint16_t u16_t port,)

Parameter:

Args: user supplied argument to match the callback function

upcb: udp pcb to be send

pkt_buf: buffer stored with data packet

port: port number need to be bind

Return:

None

Describe:

This is a callback function when Tftp server received WRQ request

Note:

Tftp_WRQRecved

Callback function when WRQ request received

Prototype:

void Tftp_WRQRecved (struct udp_pcb *upcb,
struct ip_addr *IpAddr,
uint16_t port,
struct pbuf *pkt_buf)

Parameter:

upcb: udp pcb to be send

IpAddr: IP address

port: port number need to be bind

pkt_buf: buffer stored with data packet

Return:

None

Describe:

Some operations when WRQ request received

Note:

Tftp_RRQRecved

Callback function when RRQ request received

Prototype:

void Tftp_RRQRecved (struct udp_pcb *upcb,
struct ip_addr *IpAddr,
uint16_t port,
struct pbuf *pkt_buf)

Parameter:

upcb*: udp pcb to be send*

IpAddr: IP address

port: port number need to be bind

pkt_buf: buffer stored with data packet

Return:

None

Describe:

Some operations when RRQ request received

Note:

Tftp_RecvCallback

Callback function when received data through UDP protocol

Prototype:

void Tftp_RecvCallback (void *arg,
struct udp_pcb *upcb,
struct pbuf *pkt_buf ,
struct ip_addr *addr
u16_t port)

Parameter:

arg: user supplied argument to match the callback function

upcb: udp pcb to be send

pkt_buf: buffer stored with data packet

addr: IP address

port: port number need to be bind

Return:

None

Describe:

Some operations when received data through UDP protocol

Note:

Tftp_server_init

Initial tftp server

Prototype:

void Tftp_*server_init* (void)

Parameter:

None

Return:

None

Describe:

Initial tftp server

Note: