Bootloader Manual

Bootloader version: 0.0.3

REVISION HISTORY

Revision	Date	Description
0.0.1	2003/05/06	Initial draft
0.0.2	2004/2/9	Fixed for linux development
0.0.3	2004/3/1	Remove rootdir Add configuration option ‘a’

1. Objective

This document is created to describe the loader program and corresponding initialization, driver, file system, and download modules. The main function of the loader program is to load the runtime image from the flash to DRAM, decompress it, and start execution of runtime code. Besides, the loader program provides the download and update ability.

2. File List

The source tree contains four directories.

2.1. src directory

This directory contains source codes, including C and assembly files.

- bdinfo.c: Board information, e.g. MAC address, access.
- calloc.c: Simple and small memory allocation management.
- dev.c: Proprietary device interface.
- filesystem.c: File system.
- flashdrv.c: Flash driver.
- loader.c: Main routine.
- ns16550.c: NS16550 compatible UART driver.
- uart.c: Uart function interface and queue management.
  - xmodem.c: XMODEM download.
- lx4180 directory: Lexra 4180 CPU specific source code.
  - genexcpt.c: General exception handler.
  - crt0.s: CPU initialization and startup.
  - lx4180.s: CPU specific utilities.
  - vectors.s: Reset and exception vectors.
- rtl8650 directory: RTL8650 ASIC specific source code.
  - int.c: Interrupt dispatcher.
  - phy.c: PHY register access.

  - swCore.c: Switch core access interface.
  - swNic_poll.c: Switch NIC polling mode driver.
  - swTable.c: Switch core table access.
  - swUtil.c: String utilities for switch corresponding files.
  - tick.c: Tick timer driver.
  - vlanTable.c: VLAN table access.
  - initmem.s: Memory controller driver.
- gzip directory: GZIP decompression utility.
  - gzip.c, gziputil.c, inflate.c, unzip.c, crypt.h, getopt.h, gzip.h, lzw.h, revision.h, tailor.h.
- tftpnaive directory: BOOTP and TFTP utility.
  - arp.c, bootp.c, busywait.c, icmp.c, ip.c, net.c, tftp.c, udp.c, net.h, tftpnaive.h.

2.2. inc directory

This directory contains header files.

- board.h: Board corresponding definitions.
- flash_map.h: Flash block map for file system.
- ns16550.h: NS16550 compatible UART definitions.
- rtl_bdinfo.h: Board information definitions.
- rtl_dev.h: Proprietary device interface definitions and function prototypes.
- rtl_errno.h: Error number definitions.
- flashdrv.h: Flash driver definitions and function prototypes.
- rtl_image.h: Root directory and image header definitions for file system.
- rtl_types.h: Proprietary type definitions.
- semaphore.h: POSIX compatible semaphore definitions and prototypes.
- uart.h: UART driver definitions.
- compiler directory: Compiler specific header file.

    - ghs \ rtl_depend.h: Green Hills compiler specific definitions.
    - gnu \ rtl_depend.h: GNU compiler specific definitions.
- lx4180 directory: Lexra 4180 CPU specific header file.
    - eregdef.h: CPU specific register definitions.
- rtl8650 directory: RTL8650 ASIC specific header files.
    - asicRegs.h: Peripheral and switch core register definitions.
    - phy.h: PHY access definitions.
    - swCore.h: Switch core interface prototypes.
    - swNic_poll.h: Switch NIC driver prototypes.
    - vlanTable.h: VLAN table access definitions.

2.3. bin directory

This directory contains binary utility programs and third-party libraries and header files.

- packbin.exe: Pack and pad output images.
    - 1.10.0.

2.4. make directory

3. Build Loader

1. Change directory to the product directory, e.g. ‘**/loader_srcroot/**
2. Download ldr.bix to the target board.

4. Loader Process

After the device is powered on and before the runtime codes could perform various functionalities, the CPU must be initialized, the various peripherals must be tested, and the preparation for the runtime codes must be made. These jobs are handled by one code image called the loader. In the following section, each step of the loader process is described.

4.1. Flow Chart

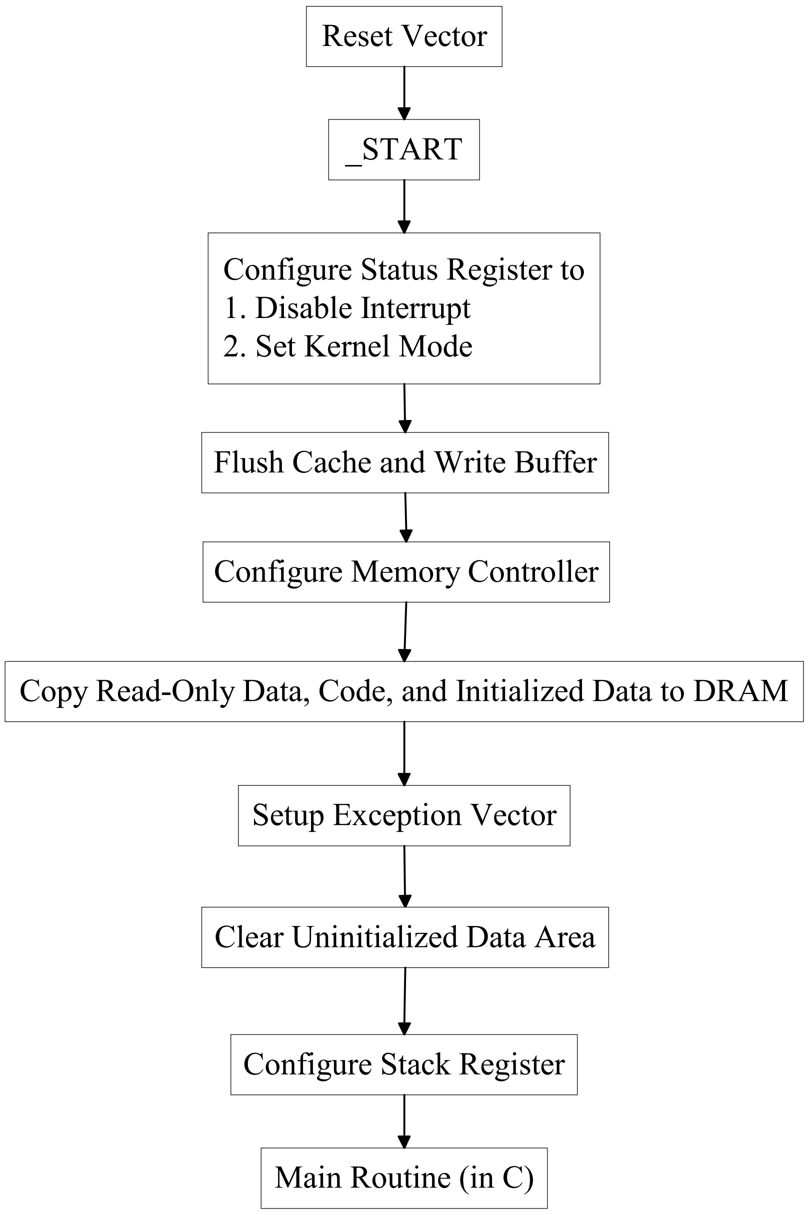
Reset Vector
_START
Configure Status Register to
1. Disable Interrupt
2. Set Kernel Mode
Flush Cache and Write Buffer
Configure Memory Controller
Copy Read-Only Data, Code, and Initialized Data to DRAM
Setup Exception Vector
Clear Uninitialized Data Area
Configure Stack Register
Main Routine (in C)

Figure 4.1. Loader Process: CPU Initialization and Test.

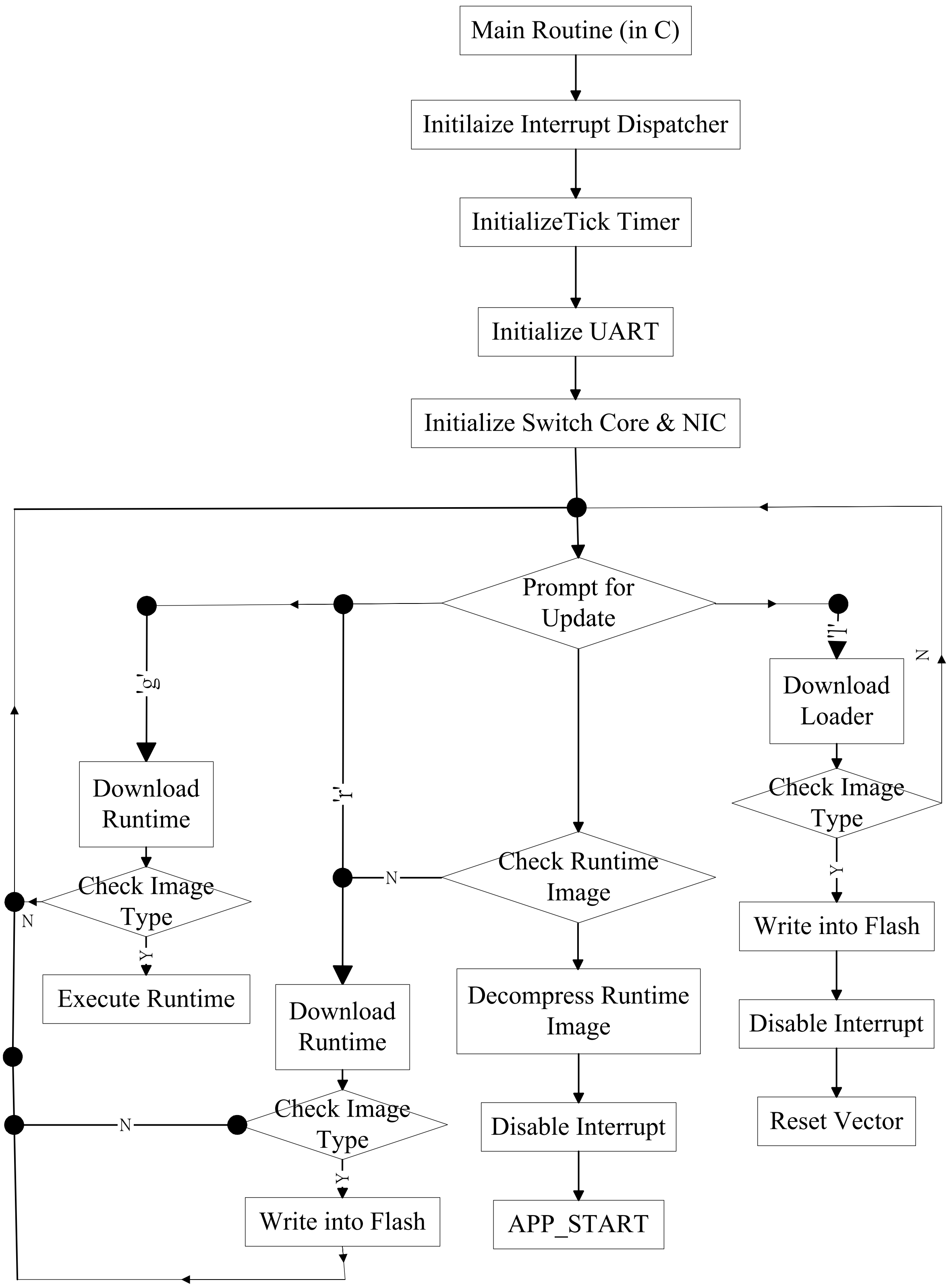

Figure 4.2. Loader Process: Transition to Runtime.

Figure 4.3. Loader Process: Download.

4.2. Update

When the UART and switch NIC drivers are ready, the program prompts user to update root directory, by pressing ‘d’, or runtime image, by pressing ‘r’. In fact, user can press ‘l’ to update loader itself. However, to use this option user must take extreme care. Besides, a useful debugging functionality is provided that user can press ‘g’ to download the runtime image and execute it directly, i.e. the flash is not updated. If user does not press any specified key, program shall test runtime image, including checking the magic number and checksum computation. If the test fails, a request for the user to download new image shall be made. The download method includes BOOTP/TFTP and XMODEM protocols. If the download succeeds and the image proved correct, the image will be updated into the flash.

4.3. File System

The loader incorporates a simple file system, which use root directory to decide the location of runtime image.

4.4. Memory Map

4.5. Version

The version number is displayed, in the format “xx.xx.xx”, behind logo messages on the console. Besides, an interface is provided to allow the runtime program read the version number of the loader. The memory section from 0x80000300 to 0x800003ff is reserved for communication between the loader and runtime program. The first word, i.e. 0x8000030[3:0], keeps the function pointer that returns the version number in the format of 32-bit integer. Each of the least significant three bytes equals one number in the version string, e.g. 0x00616263 indicates “97:98:99”. The following code is an example illustrating how to read version number of the loader in the runtime program:

```
#define LDR_COMMUN_BASE 0x80000300
    uint32 ver;
    uint32 (*getLdrVer)(void);
    uint8 buf[10];

    getLdrVer = (uint32 (*)(void)) *(volatile uint32*)LDR_COMMUN_BASE;
    ver = getLdrVer();
    sprintf(buf,"%02d.%02d.%02d",(ver>>16)&0xff,(ver>>8)&0xff,ver&0xff);
```