The Jasmine OpenSSD Platform

Version 1.2

FTL Developer's Guide

Revision history

Date	Author	Description	Rev
2011-04-27	Sangphil Lim (Sungkyunkwan University)	Initial release	1.0
2011-05-20	Sangphil Lim (Sungkyunkwan University)	Section 2.2 update	1.1
2012-01-12	Sangphil Lim (Sungkyunkwan University)	Section 2.2.3 update	1.2
2015-08-22	Preethika Kasu, Donghyun Kang, Heerak Lim (Ajou University)	Translated to English	1.2

Contents

The Jasmine OpenSSD Platform: FTL Developer's Guide

Preface

About this Document

This manual guides to develop a FTL for Jasmine OpenSSD Platform which is based on the controller of Barefoot™

Contents

Further Reading

For better understanding refer OpenSSD and RealViewDebugger

Chapter 1.
Getting Started to Develop an FTL

This chapter introduces the development environment before starting the FTL development for Jasmine OpenSSD platform. The following are the key contents in this chapter:

- ✓ Hardware and Software requirements for FTL development
- ✓ Development environment building guide

1.1. Development Environment

1.1.1. Requirements Specification

To develop a new FTL firmware on the Jasmine OpenSSD platform, the following are the recommended requirements:

H/W requirement

- Test PC : 1
  - Communicates with the Jasmine Board
- Client PC : 1
  - Debugs using RVD(RealViewDebugger) or UART
- Jasmine Board
  - Jasmine board with the flash module attached, SATA and RS232 cable
- RealView In-Circuit Emulator(ICE) equipment(*optional*)
  - ICE body, Ethernet/USB cable, JTAG connection cable

S/W requirement

- O/S: Windows XPSP2
- Firmware compiler software
  - RVDS(RealView Development Suite) 3.0 or more
  - Or GNU Compilation toolchain (The Sourcery G++ Lite Edition)
- Microsoft Visual Studio8
  - Add PATH environment variable: C:\Program Files\Microsoft Visual Studio8\VC\
- Jasmine OpenSSD platform firmware
- using UART, Hyper terminal program is needed

1.1.2. Development Environment Setup

Build the development environment referring Figure (1) meeting the requirements in 1.1.1

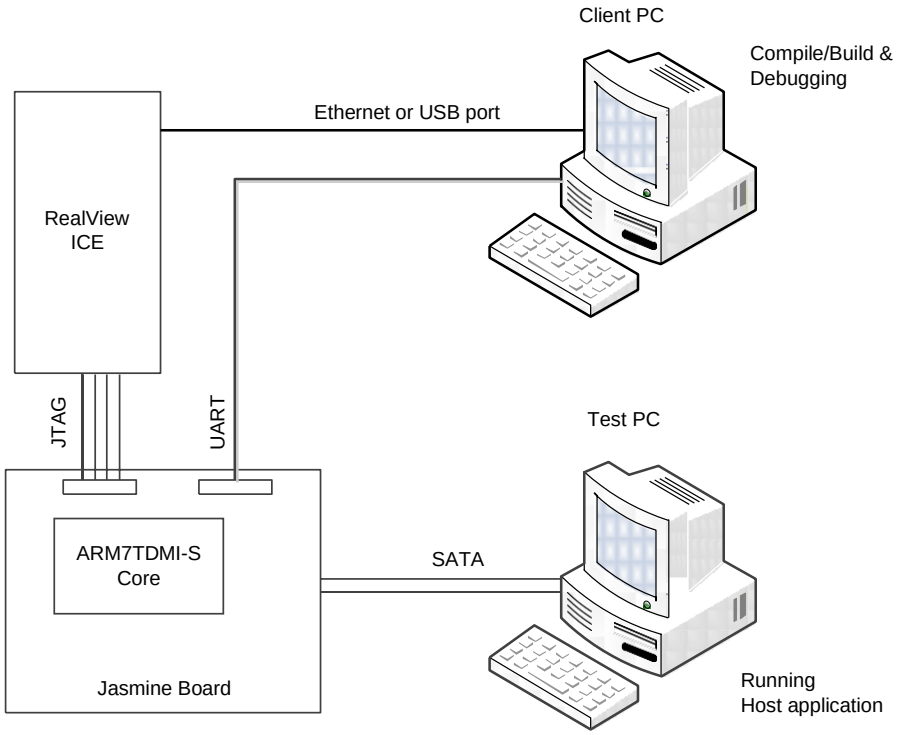


Figure (1) FTL Development Environment

- Connect Jasmine board to power and Test PC with SATA cable
- For debugging, connect RVICE device to Ethernet or USB port, connect UART interface of Jasmine board to Test PC with RS232 serial cable
- Install RVDS, RV ICE software and SSD firmware to Client PC
  - evaluate RV ICE and Jasmine board with RV debugger

NOTE: Client PC is not required when host application is used for bench mark or UART port is used to debug i.e., when RV debugger is not used

---

Chapter 2.
FTL Porting

This chapter explains how to port new FTL to Jasmine OpenSSD platform

2.1. Porting Guide

While porting, FTL reads and writes data from DRAM or NAND Flash memory. The below are the points to be considered when accessing the hardware component:

2.1.1. Implementation

Together with the FTL protocol API, the following functions should be implemented before porting

Source file	Function	Description
./installer/installer.c	ftl_install_mapping_table	This function is called when the firmware is installed and records early metadata to NAND flash memory
./ftl_[scheme]/ftl.c	ftl_open	This function initializes the FTL - loads and initializes the metadata form NAND Flash memory -call format function when VBLK #0 is not in marked state
	format	This function erases all the blocks apart from VBLK #0 and FTL metadata field - records format mark
	ftl_read	This function reads the user data
	ftl_write	This function writes the user data
	ftl_flush	This function flushes the metadata to each SATA idle/ stand by time on a regular basis

2.1.2. DRAM Host Buffer Management

SATA interface buffers the user data at DRAM host buffer (i.e. SATA Read / Write buffer). When a read request is triggered, the read ATA command is sent form the event queue, then FTL sends FCP command to flash memory controller to read the data from NAND flash. Finally, the data is transferred to the assigned SATA read buffer. When a write request is triggered, the data is stored at the NAND flash.

DRAM host buffer is managed by SATA and FTL by adjusting the pointer in the hardware buffer manager. Buffer pointer can be clashed due to the difference in the I/O performance of Flash memory and SATA bandwidth. To overcome the clash, FTL should be implemented in ftl_read / write functions such that the buffer pointer should be incremented to avoid the rotational queue way of the host buffer

---

NOTE: This is applicable only if the commands are passed directly to the Flash controller by adjusting the FCP. It is not applicable if FTL is implemented using LLD library as LLD adjusts the DRAM host buffer pointer

---

2.1.3. DRAM Access Limitation

Barefoot controller uses hardware ECC engine to improve the reliability of DRAM data. The data in the memory can be lost or invalid data can be read when CPU directly reads or writes DRAM data due to the ECC parity data. Therefore, for the reliable DRAM data, DRAM data should be accessed by hardware Memory Utility (`./include/mem_util.h`). For example, the metadata in DRAM can be accessed using `read_dram_xx or write_dram_xx` library respectively to read the data from DRAM to SRAM and to write the data from SRAM to DRAM (`mem_copy` can also be used)

The performance can be improved with `mem_copy` utility, if the frequently used DRAM data is cached in SRAM

The following are the constraints to be considered before working with memory:

1. 4 Byte ECC parity information per 128 Byte would be added
2. Available DRAM field is 64MB*128/132 = 65075200Bytes
3. Hardware Memory utility must be used while copying data between DRAM-to-SRAM and SRAM-to-DRAM
4. The size of the data should be equal to the `DRAM_ECC_UNIT` (128 Bytes) while copying the data from DRAM-to-SRAM
5. Starting address of the metadata has to be declared at DRAM and should be a multiple of `DRAM_ECC_UNIT`or`BYTES_PER_SECT` size.

2.1.4. NAND configuration

Barefoot controller uses VBLK#0 in flash memory to store firmware binary images and metadata (e.g. scan list). FTL uses the other blocks (apart from VBLK#0) to store user data and metadata

Many FTL firmware's proposed many ways to store FTL metadata to spare area of Flash memory considering the POR/SPOR. However, Barefoot controller can't use spare area or Flash memory, so FTL logs metadata in separate blocks assigned in advance.

2.1.5. Flash Command

The process that FTL passes I/O request to flash memory is briefly explained. First, FTL sets flash command to FCP which redirects it to the WR (Waiting Room). Then that command waits in WR for a while. NAND Flash controller checks the bank status of the flash command and if the bank is idle then deliver the flash command to BSP (Bank Status Port).

NOTE: Refer the technical manual for FCP, WR and BSP

As the queue depth of WR is '1', it must check weather a command already exists in the queue before passing a new flash command to WR. If the FTL issues a new flash command while a command already exists in the WR queue, then the existing command may not work properly. Therefore, it should be carefully implemented

NOTE: Refer the flash issue cmd function of ./target_spw/flash.c

Auto-Select Mode *(ADVANCED)*

In a write operation, hardware can use Auto-select mode which can maximize I/O parallelism by selecting automatically bank of idle state. But before passing FCP_BANK register of FCP should be set to 0x3F

2.2. Porting Example – GreedyFTL

FTL, Greedy FTL (. /ftl greedy/ ftl.c) and page mapping FTL that executes simple garbage collection operation is implemented in Jasmine firmware. Along with garbage collection operation, Greedy FTL can also implement normal power-on/off using Power-off Recovery (POR) function. This chapter explains the functions that should be used during porting while implementing Greedy FTL

2.2.1. FTL Initialization

Jasmine board first initializes the hardware component. Hardware component then calls the ftl open function to initialize the FTL. Initialization process of Greedy FTL is described below:

1 Set the bad block bitmap table by reading scan list from the VBK #0 by calling build bad blk list function

2 Execute format operation until format mark is written in VBLK #0

- The format function is responsible for initializing the metadata of SRAM/DRAM and deleting the scope except in VBLK #0
- Write initialized metadata in meta scope of NAND flash and format mark in VBLK #0.
- If it is not booting for the first time i.e., no need to execute format, then load the FTL metadata from NAND flash

3. Initialize the FTL buffer pointer

2.2.2. SRAM Metadata

SRAM manages the frequently referenced FTL metadata which is smaller in size. The Greedy FTL manages the writing page index pointer on the FTL Meta scope and user data scope, free block count, statistical information etc. in SRAM.

2.2.3. Address Mapping

Unlike FTL, the Greedy FTL's address mapping is implemented by fixed bank mechanism to LPN i.e., a certain LPN always make read/write operation to occur in a certain bank. The target bank under LPN is calculated by modular operation using NUM_BANKS and mapping information only manage the VPN information.

The performance of parallelism degrades for a random IO access and the performance increases for a sequential IO access, especially in case of sequential read operation than FTL.

2.2.4. DRAM Metadata

```
DRAM reserves a separate memory for the metadata, if the size of
the metadata managed by FTL increases more than the available
space in SRAM. In this case, the 'DRAM segmentation' of ftl.h
should be revised.
```

The following table contains the list of DRAM Meta data managed by Greedy FTL:

Table (1) DRAM metadata for Greedy FTL

Metadata	Description
BAD_BLK_BMP	The bitmap table on bad block list that is obtained from scan list.
PAGE_MAP	The size of PAGE_MAP that is needed to map physical page index to logical page index can be calculated from [the number of total logical block X block per the number of page X 4B].
VCOUNT	Manages the number of available pages in a block that are responsible for selecting a block to be sacrificed in garbage collection. The number of VCOUNT can be calculated from [the number of bank X bank per the number of virtual block X 4B].

2.2.5. NAND Configuration

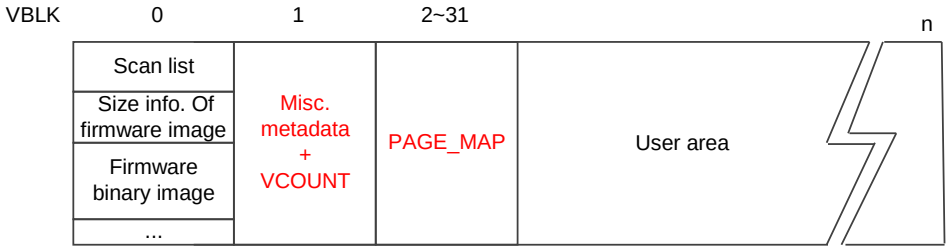


Figure (2) NAND configuration in Greedy FTL

The Greedy FTL manages NAND flash as in Figure (2). A part of FTL meta logging scope is reserved for POR and the remaining scope is used to record the user data. The FTL metadata is sequentially recorded in meta blocks.

The blocks in user scope have same structure as in Figure (3). If the number of pages in a block is 'm', pages from 0 to m-2 records the user data and page m-1 records the LPN information of pages from 0 to m-2. The meta information is separately recorded because the firmware can't use separate secondary scope in a page, due to the property of Barefoot controller. LPN information is used to discriminate the validity of the user data for further garbage collection

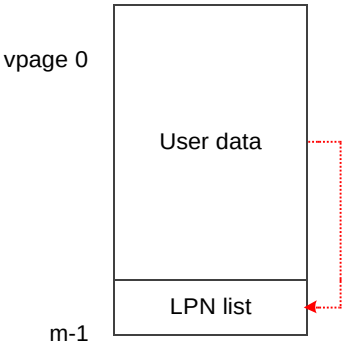


Figure (3) Block structure of user area in Greedy FTL

2.2.6. GarbageCollection

Garbage collection of the Greedy FTL is invoked if there is no more space available to write the new data. A block containing the least number of available pages is selected as a sacrifice block i.e., get_vt_vblock() of ./ftl_greedy/ftl.c, because of the lowest cost for the copying page operation in garbage collection.

NOTE: Refer the below thesis for the garbage collection policy.

```
Atsuo Kawaguchi and Shingo Nishioka and Hiroshi Motoda, A Flash-
Memory Based File System, USENIX Winter, pp. 155-164, 1995.
```

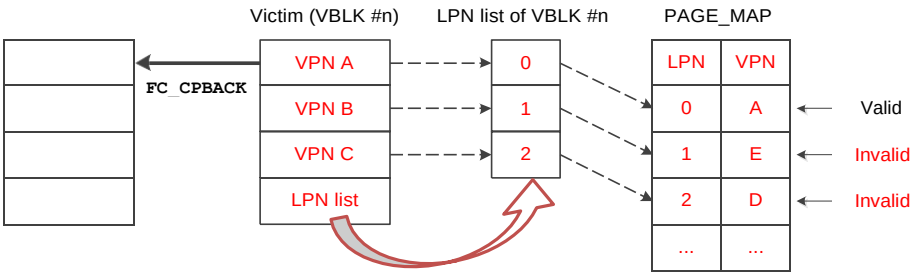


Figure 4) Garbage Collection in Greedy FTL

First, search the block having the lowest VCOUNT value from the VCOUNT meta table of DRAM, the selected block is called the victim block. LPN list is read from the last page of the block. The validity of the data in the victim block can be verified by comparing the PAGE_MAP information, whether the LPN list information maps to the VPN that have the latest information of the given LPN. From the Figure (4), the victim block VPN A having the user data is mapped to LPN 0 which is the same in PAGE_MAP table i.e., LPN o mapped to VPN A. So, the data is the latest. Whereas VPN B and VPN C are mapped to 1 and 3 respectively which is not the same in PAGE_MAP table i.e., LPN 1 maps to E and LPN 2 maps to D. So, the data is invalid.

The valid page using this discrimination method is copied to an empty block that is already assigned by using copy-back function of LLD library and then the existing block will be deleted. For reflecting changed FTL meta information, the garbage collection is ended by renewing PAGE_MAP and VCOUNT meta table. After that, the new data is written to that block.

2.2.7. POR (Power-Off Recovery)

If the Jasmine board is normally ended, the Greedy FTL supports the POR. If there are no IO requests from the host, thus in SATA idle/standby state, the whole FTL metadata is logged in the NAND flash at the FTL meta scope as shown in Figure 2 (See the ftl flush).

If the Jasmine board is normally ended and booted, it doesn't perform format and is ready to perform the user IO request by loading FTL meta data logged in FTL meta scope (See the load_metadata function).

2.3. How to verify FTL Operations?

2.3.1. FTL logic test

After a write operation, the FTL movement can be verified by comparing two buffers data that are read from the same scope. FTL movement verification can be done by the following methods:

1 Using a special verifying program or verifying by inserting codes that operate read-after-write in host application (e.g. IOmeter).

2. Using the FTL test code of firmware.
   - Set the OPTION_FTL_TESW of ./include/jasmine.h to 1 to test the FTL code. If this option is set, after initializing Jasmine, FTL code is verified by passing SATA and by calling ftl_test function. This function first executes ftl_write function and then calls ftl_read function to execute comparison operation for data validity.
   - To test in various occasions, implement various test codes in firmware code to verify the data.

2.3.2. POR Test

POR feature of the FTL can be tested by setting the OPTION_FTL_TEST to 1 and the ftl_test is executed by the booting firmware. The POR code is executed explicitly by turning the power on and off so as to execute ftl_test again.

2.4. Setting up to build

To build the FTL mechanism that is porting to Jasmine firmware, create a folder (./ftl_xxx) and add the relevant header file to FTL and source code file.

The firmware source file list that needs to be compiled is drafted in the file_list.via of the build folder (./build_rvds or ./build_gnu). If there are any source files apart from ftl.c, they should be added to the file_list.via to compile together.

The compile option for the relevant header file of the Jasmine should be revised. The sort of NAND flash chip, composition of banks, 2-plane mode, size of DRAM, clock working speed etc., could be determined by this setting, in addition, the FTL test mode, assert verification, UART debugging, SATA 2.0/1.0, SATA NCQ could be activated or deactivated. Once the build setting is finished, build the firmware and install firmware binary image in Jasmine board as explained in Chapter 3.

Chapter 3.
Compile, Build & Install Firmware

This chapter explains the process of compiling, building and installing firmware in Jasmine board.

3.1. Compile & Build Firmware

First, download the latest firmware source file of the Jasmine OpenSSD platform to the Client PC.

The Jasmine OpenSSD platform can be compiled in two methods. First method uses RV ICE equipment and RVDS. Second method uses GNU tool-chain.

3.1.1. Build firmware using RVDS tool-chain

RVDS 3.0 should be installed in the Client PC to build the firmware using ARM RVDS.

NOTE: If the licensed formal version is not available, then registering at ARM Ltd., website allows to download the evaluation version of ARM RVDS 4.1 Professional which expires in 30 days.

Execute the firmware build from the command window as below:

```
>cd ./build_rvds
> build.bat [tutorial | greedy]
```

Once the firmware build is executed the firmware .bin i.e., firmware binary image is created at the pertinent folder.

NOTE: If the below error is encountered during the build, halt the vaccine program and execute.

mt.exe : general error c101008d: Failed to write the updated manifest to the resource of file...

Build firmware using GNU tool-chain

The Jasmine OpenSSDflatform can also build firmware by using GNU compile tool.

First, download latest version of the Sourcery G^{++} Lite Edition from CodeSourcery and install at Client PC.

The Makefile is forced to build Tutorial FTL. Therefore, modify the first line of the Makefile to build firmware by using ported new FTL mechanism.

```
FTL = new_scheme
...
```

And execute the below commands in the command window.

```
>cd ./build_gnu
> build.bat
```

Once the firmware build is executed the firmware .bin i.e., firmware binary image is created at the pertinent folder.

3.1.2. Compile Firmware Installer

Execute install.exe to install the firmware. For executing the install.exe file, first, install the Visual C++ 2010 Express Free Edition.

For the earlier version, build the Visual C++ 2005 solution file (./installer/installer.sln) to create install.exe file in pertinent path. Copy the created firmware installer to the FTL build path (e.g. if you try to install Tutorial FTL, ./ftl_tutorial/).

NOTE: Execute the installer after the rebuild if the channel/way configuration of Jasmine board and the BANK_BMP of ./include/jasmine.h are modified.

3.2. Install Firmware

Install the firmware binary image created in chapter 3.1 on the Jasmine board using install.exe. The firmware installation process is as follows:

1. Booting Jasmine board with 'Factory mode'.
   - For booting Factory mode, J2 jumper of Jasmine board should be set as shown in the below image and connect the power and SATA cable. Then turn on the power switch.

Factory Mode

   - Once Jasmine board is booted in Factory mode, Jasmine board is accepted to 'YATAPDONG BAREFOOT-ROM in Device Manager-Disk Drive of host PC.

2. Installing firmware in Jasmine board by executing the installer (install.exe).

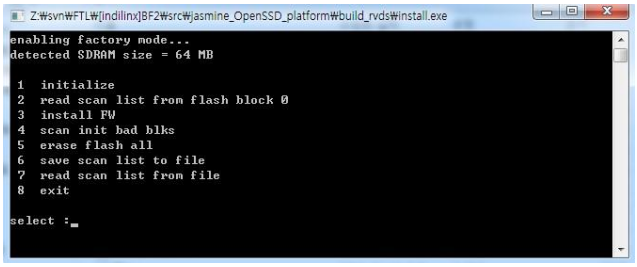


Figure (5) Jasmine Firmware Installer

- The options are explained below:
  1. `initialize`
     Initializes the Jasmine board.
  2. `read scan list from flash block 0`
     Loads bad block list installed in block 0 of the bank zeroth NAND flash chip mounted on the Jasmine board.
  3. `Install FW`
     Installs firmware in Jasmine board.
  4. `scan init bad blks`
     Writes the bad block list scanning the NAND flash memory.
  5. `erase flash all`
     Deletes all the block in the NAND flash chip. Utmost care should be taken as the firm ware installed in block 0 and the bad block list are deleted at the same time.
  6. `save scan list to file`
     Saves the bad block scan list from the option 2 or 4 to the working PC as a file.
  7. `read scan list from file`
     Writes bad block list by reading scan list file that is saved in option 6.
  8. `exit`
     Exits the installer.
- The scan list is already installed in NAND flash memory of Jasmine board during release time. Therefore, before installing firmware, the backup scan list in PC should be foregone. This should be done in the order of 1-2-6-3 options.
- If the scan list installed in block 0 is corrupted, an error encounters in executing the option 2. Therefore, the firmware should be installed after saving the scan list file in the PC. This should be done in the order of 1-7-3 options.
- If the new NAND flash module is installed in Jasmine board, install the firmware after saving the scan list by scanning bad blocks in the NAND flash memory. This should be done in the order of 1-4-6-3 options.

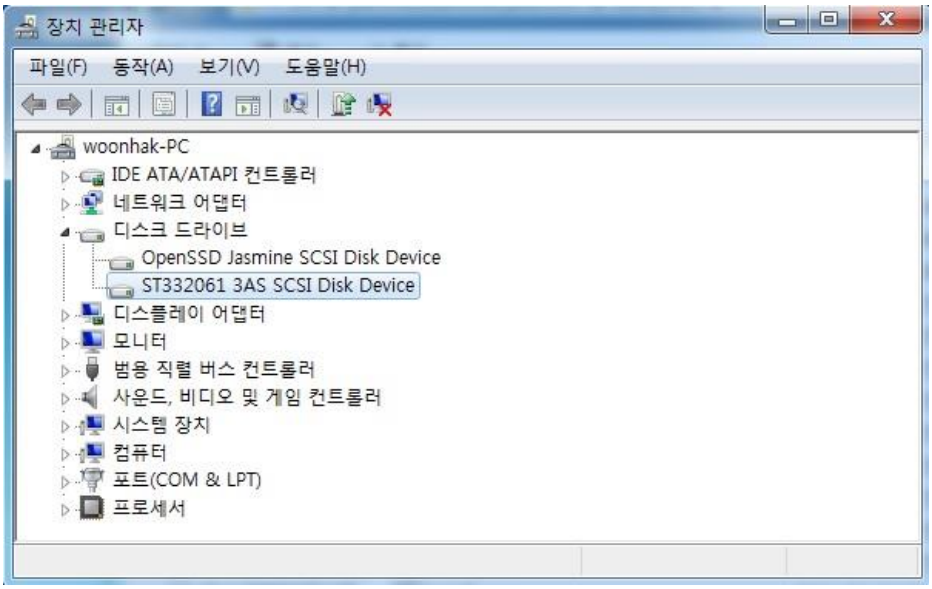
If the firmware installation is successful, you should set J2 jumper of Jasmine board with Normal mode as in the below image:

Normal Mode (Default)

Pull the SATA cable before turning on the power switch. After successfully executing the FTL format, turn on the LED in D4 location. Once the SATA cable is connected, the Jasmine board is ready to execute the SATA command from a host.

NOTE: Once the power of the Jasmine board is turned on, the ftl_open function is called internally and the pertinent operation might take long time for the implementation. Due to this the response time-out error may occur which turns on the LED at D4 location and ends the ftl_open function. Connect the SATA cable when the LED at D4 location is on.

If the firmware is installed successfully in the Jasmine board, the 'OPENSSD Jasmine' is perceived in the Device-Manager disk drive as below and then it can be used as a complete flash SSD.



Chapter 4.
Debugging Tips

The Jasmine OpenSSD platform outputs the message through the UART interface and a real time firmware debugging can be done using the ICE equipment and the RV Debugger. In this chapter, these two methods are accounted to verify the FTL mechanism that is installed in the Jasmine board.

4.1. Debugging with UART

This chapter describes the debugging process with the information printed on the terminal window through the UART interface.

4.1.1. Debugging Setting

First, the UART port of Jasmine board (P1) is connected to the serial port of Client PC through the RS232 cable. In order to use the UART interface, the on-board switch of the Jasmine board (SW2, 3, 4) should be set as below.

- SW2: No. 1, 2, 3, 4 (ON)
- SW3: No. 1, 2, 3, 4 (OFF)
- SW4: No. 1, 2, 3, 4 (OFF)

Once the Jasmine board settings are done, a serial port of the terminal program should be set as below.

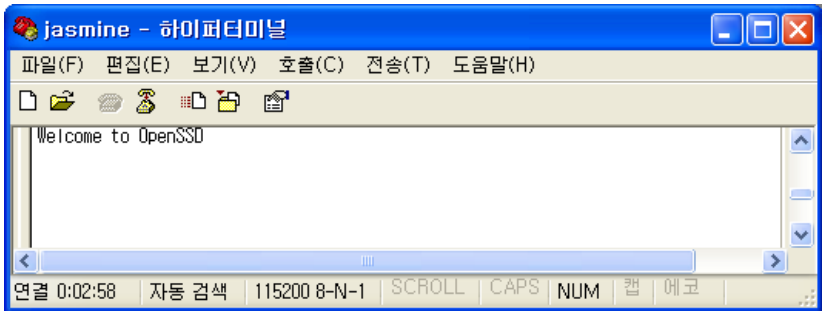
- bit/sec (Baud rate):115200
- Data bit: 8
- Parity: nothingness
- Stop bit: 1
- Flow control: hardware (or Yes)

To receive a output message from UART interface, firmware must set `OPTION_UART_DEBUG`. Set `OPTION_UART_DEBUG` (`./include/jasmine.h`) to 1.

4.1.2. Debugging by Printing Message

If the UART port is activated, dump a specific memory field (if some error occurs) or debug the FTL firmware performance by printing the debugging message using UART message print function i.e., uart_print, ./target_spw/uart.c.

If the UART port of the Jasmine board is initialized normally and the Jasmine board is booted in the Normal mode, the following message should be printed.



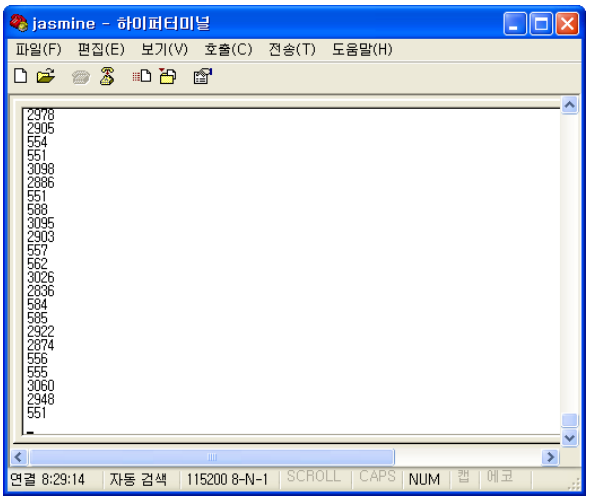
4.1.3. Measuring Response Time

Jasmine firmware also provides some functions to measure the FTL performance using the Timer. Using the timer functions FTL read/write or garbage collection overhead can be measured, response time can be checked or can debug the errors by printing the messages to the UART port.

The following code explains the usage of Timer functions (`ptimer_start, ptimer_stop, _uart_print;./target_spw/misc.c;`) to check the response time when `ftl_write` is performed

```
ptimer_start();
ftl_write(lba, num_sectors);
ptimer_stop_and_uart_print();
```

The following figure shows the response time (unit: us) for the above code when inserted in the FTL test function (`ftl_test ; ftl.c`).



4.2. Debugging with RVD

This chapter explains the firmware debugging process using RealView debugger and RealView ICE.

NOTE: If the FTL firmware operation is suspended explicitly with a break point while the FTL is processing the I/O command issued by the host to the Jasmine board, because of the busy waiting the OS may stop. So, it is recommended to use a separate client PC for debugging.

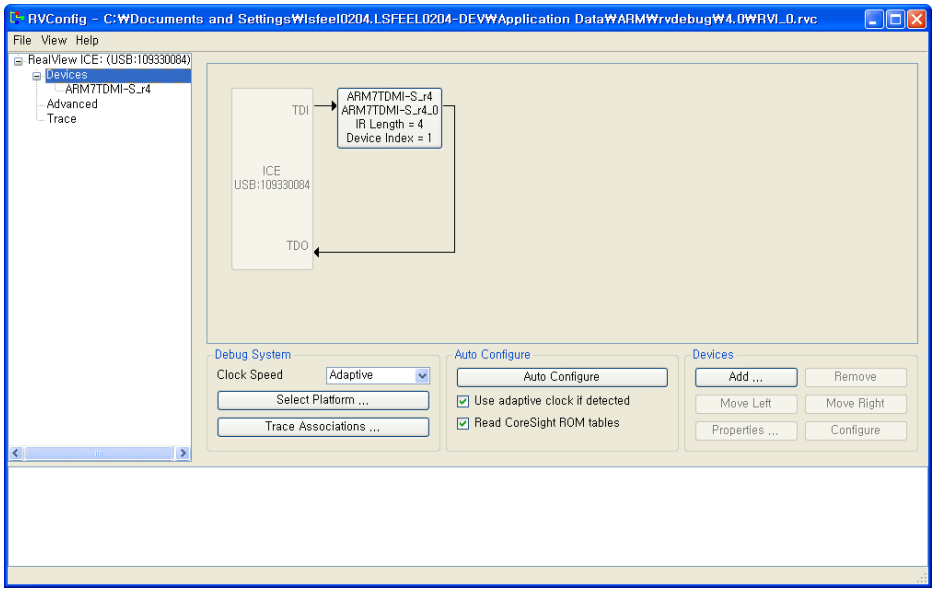
4.2.1. Debugging Setting

Firstly, edit the Compile environment file to perform line-by-line debugging.

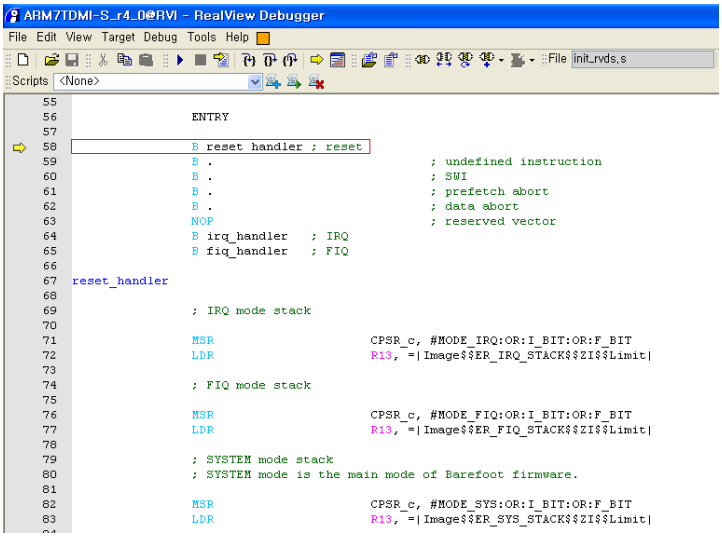
Edit the Compile environment file (`./build_rvds/armcc_opt.via`) as following and build the firmware

```
-O3 // modifying to -01
-Otime // delete
```

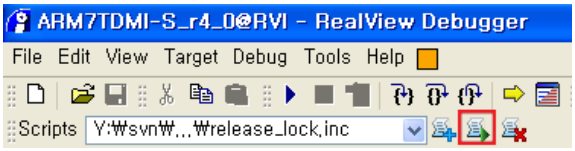
As Figure1, after executing the RVD setting, check connecting the ICE device with the Jasmine board normally through a setting in the RV debugger as below:



Once the connection is complete, load the firmware image built at Target-> LoadImage in the options. If the image is normally loaded, a firmware startup code (`./target_spw/init_rvds.s`) can be seen as below.



Finally, adding and executing the ./`release_lock.inc` script as below opens the JTAG port of the Jasmine board completing the debugging setting.



4.2.2. Debugging tip #1 – "Use a break point statement"

Once the Jasmine board is powered on, the init_jasmine function of `./target_spw/initialize.c is first called by the startup code. This function initiates various hardware components and SRAM/DRAM scope and waits for the user IO request after calling the ftl_open function.`

Let's suppose a bug is encountered in the firmware before calling a function that waits for the user IO request after turning on the power. For example, let's suppose there is a bug that writes a data on a memory scope in the ftl_open fucntion that can't be referred.

```
void ftl_open(void)
{
    *(UINT32*)0xFFFFFFFE = 10; // occur data abort
    ...
}
```

The above mentioned case is an example of the data interrupt occurred when executing a pertinent code as soon as the Jasmine board is powered on. It is difficult to find the location of the code where the interrupt has occurred though RV debugger is used.

Thus, to resolve this problem, start debugging by explicitly stopping the firmware operation just before the occurrence of the interrupt. This method inserts a dummy while statement as below:

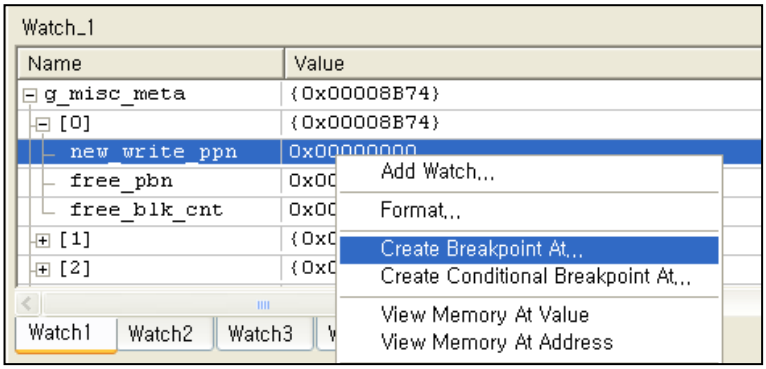
```
volatile UINT32 g_barrier;

voidinit_jasmine(void)
{
    ...
    g_barrier = 0;
    while (g_barrier == 0);

    ftl_open();
    ...

}
```

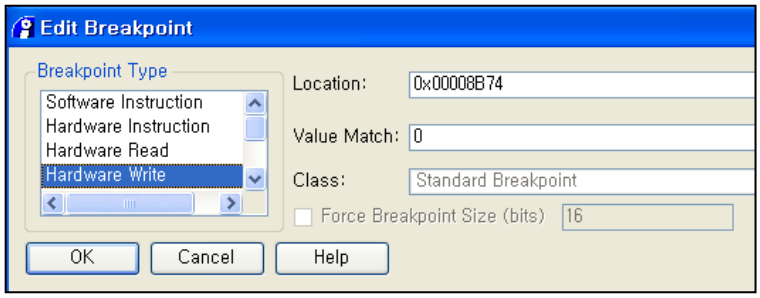
After stopping the firmware flow with the dummy while statement before executing the ftl_open function, proceed with the line-by-line debugging executing the RV debugger that modifies g_barrier value to 1.

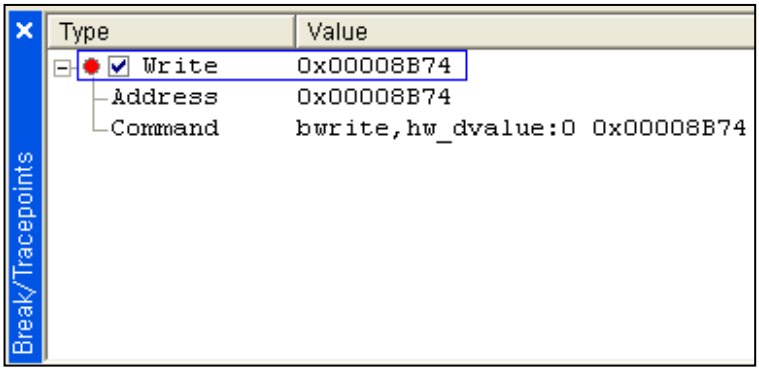
4.2.3. Debugging tip #2 – "Use a H/W breakpoint"

If there is a logical error in the FTL code that reconfigures the value of the metadata and the memory buffer. In this case, the debugging can be done using a 'hardware breakpoint ' of the RV debugger.

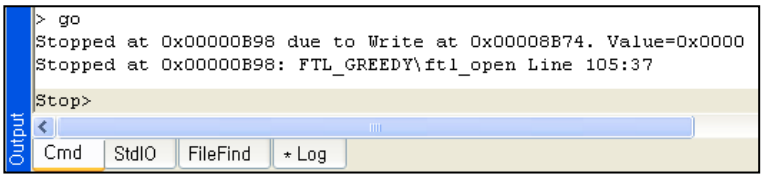
Stop the firmware flow when the metadata value is changed to '0' and register 'hardware write' breakpoint at a pertinent memory address as shown in the below image.







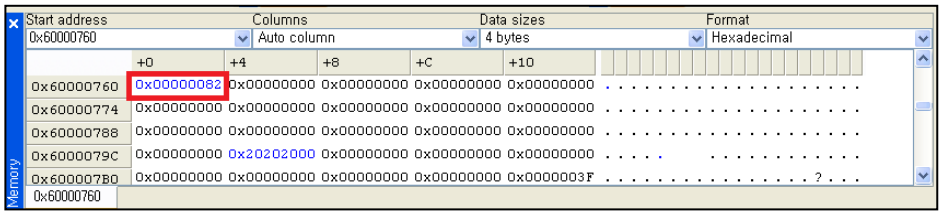
After registering the breakpoint, a bug can occur if the debugging is reactivated because the firmware flow is stopped at the line where there is a '0' at the memory address which is an observation point.



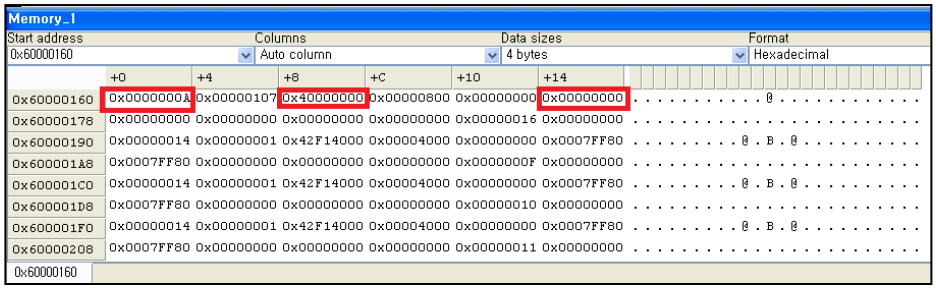
4.2.4. Debugging tip #3 – "Watch status registers"

The latest executed flash command is written in the BSP, in case of an interrupt, the details of the interrupt is stored in the BSP_INTR register which helps FTL to debug.

Navigate to Memory window from **View->Memory** of the RV debugger, BSP interrupt information can be found when the memory address of the BSP_INTR register is entered. `FIRQ_DATA_CORRUPT(0x82)` interrupt can occur at bank0 as below:



Also target bank, block, page number and buffer address can be known from the flash command remained in the BSP. The below image depicts an interrupt when a page read operation is executed i.e., `FC_NORMAL_READ_OUT (0x0A)`.



NOTE: Ensure the memory value as a little endian. Refer the Technical Manual for the BSP register, flash command macro and DRAM memory map.