TC | APPLIED TECHNOLOGIES

Firmware Load Specification

Date: June 20, 2008

Document History

Version	Date	Name	Comment
0.1.0.0	July 25, 2007	M Lave	Document created
3.0.1.0	Dec. 14, 2007	M Lave	Added Test/Debug interface desc.
3.2.0.0	June 20, 2008	M Lave	Added capabilities progress and atomic erase/program. This addition id fully backward compatible.

Table of Contents

1 Introduction

This specification specifies the Firmware Load protocol for DICE products. The implementation is part of the application. The new firmware is first loaded to ram from the host. The device then performs a checksum calculation which the host will match to a local check. If everything is fine the host will instruct the firmware to delete the old image and create a new.
The device is sensitive to power off during the erase and create process. If the firmware is lost the device can recover through the serial port. We have not had any such incidents in the field and we believe it is highly unlikely to happen.

2 Communication model

The communication model is based on a private memory space. This memory space implements a command interface with a data buffer which should be filled before the command is issued.

Address	Parameter name	Size	Att
FFFF E010 0000_{16}	FIRMWARE_LOAD_SPACE	See below	RW

Offset	Parameter	Size	Attribute	Function
00_{16}	VERSION	32bit	RO	Not used
04_{16}	OPCODE	32bit	RW	Writing op-code with execute bit set initiates execution.
08_{16}	RETURN_STATUS	32bit	RO	Return value from previous operation. A value of zero indicates successful execution.
$0C_{16}$	Progress	32 bit	RO	Contains progress information for lengthy commands
10_{16}	Capabilities	32 bit	RO	Contains capability bits
14_{16}	Reserved	6x32bit	RO	Reserved for future use
$2C_{16}$	Param/Return data	Variable	RW	Buffer for command parameters to be written and return values to be read.
$FD8_{16}$	testDelay	32bit	RW	Part of test and debug interface.
FDC_{16}	Testbuf	32x32bit	RO	Part of test and debug interface.

Definition of OPCODE bitfield

Bit	Name	Meaning
0..11	OPCODE_ID	The op-code identifier, see table below for a complete list of op-codes.
31	EXECUTE	The host writes an op-code with this bit set. This will initiate execution and the bit will be cleared by the device when execution is completed.

Definition of OPCODE_ID's

Val	Name	Meaning
0000_{16}	GET_IMAGE_DESC	Returns the RedBoot FIS image information for a given image. This command takes the image number as parameter.
0001_{16}	DELETE_IMAGE	Deletes a given image identified by name. This command takes the image name as parameter.
0002_{16}	CREATE_IMAGE	Create a new image based on the data uploaded and the information passed as parameter to this command. The parameters passed are length, exec. Addr, entry addr and image name.
0003_{16}	UPLOAD	Upload part of an image. The parameters passed are index, length and up to 1024 bytes.
0004_{16}	UPLOAD_STAT	Calculates the checksum of the uploaded image. The parameter passed is the length of the total image.
0005_{16}	RESET_IMAGE	Force a reset of the device
0006_{16}	TEST_ACTION	Test actions, see section 9.
$000A_{16}$	GET_RUNNING_IMAGE_VINFO	Get information on the running image.

Definition of Capabilities bitfield

Bit	Name	Meaning
0	FL_CAP_AUTOERASE	If this bit is set, auto erase is supported. In that case the CREATE_IMAGE opcode will automatically delete the old image if needed. This is backward compatible so the DELETE_IMAGE opcode will still work.
1	FL_CAP_PROGRESS	If this bit is set, the protocol supports progress information for lengthy operations. Further description of progress can be found below.

Definition of Progress bitfield

Bit	Name	Meaning
0..11	PRGS_CURR	This field indicates the current progress value
12..23	PRGS_MAX	This field indicates the max progress value.
24..27	TOUT	This field indicates the max expected seconds for a progress step to complete.
31	PRGS_FLAG	This bit is set if the current command supports progress information and if the other fields are valid.

Currently the DELETE_IMAGE and CREATE_IMAGE functions will provide progress information. When polling for the opcode to complete it is possible to read Opcode, ReturnStatus and Progress as one 3 quadlet block read. While polling for completion the progress information can be used to update a progress bar so the user can track progress of these lengthy commands.

2.1 GET_IMAGE_DESC

This command returns the image by ID. This can be used to list the images in the flash. A firmware loader would normally not need to use this command. By default DICE firmware is in an image called DICE and the serial number is in an image called SETUP.

Parameters: uint32 imageID

Return:

```
typedef struct
{
        char       name[MAX_IMAGE_NAME];
        uint32     flashBase;
        uint32     memBase;
        uint32     size;
        uint32     entryPoint;
        uint32     length;
        uint32     chkSum;

        uint32     uiBoardSerialNumber;
        uint32     uiVersionHigh;
        uint32     uiVersionLow;
        uint32     uiConfigurationFlags;

        char       BuildTime[64];
        char       BuildDate[64];

}FL_GET_IMAGE_DESC_RETURN;
```

RETURN_STATUS:

```
E_GEN_NOMATCH     = 0xff000000 //No image with that ID
NO_ERROR          = 0x00000000
```

2.2 DELETE_IMAGE

Parameters: char name[16] //zero terminated

Return: none
RETURN_STATUS:
E_FIS_ILLEGAL_IMAGE
E_FIS_FLASH_OP_FAILED
NO_ERROR

2.3 CREATE_IMAGE

Parameters:

```
typedef struct
{
      uint32     length;
      uint32     execAddr;
      uint32     entryAddr;
      char       name[16];
}FL_CREATE_IMAGE_PARAM;
```

The length is the length of the image in bytes, execAddr is 0x30000 and entryAddr is 0x30040. The name is a zero terminated string. Firmware images are named ‘dice’ and the setup image is named ‘setup’.

Return: none
RETURN_STATUS:
E_FIS_ILLEGAL_IMAGE
E_FIS_NO_SPACE
E_FIS_FLASH_OP_FAILED
NO_ERROR

2.4 UPLOAD

Parameters:

```
typedef struct
{
      uint32      index;
      uint32      length;
      uint32      buffer[256];
}FL_UPLOAD_PARAM;
```

Return: none
RETURN_STATUS:
E_DICE_BAD_INPUT_PARAM
NO_ERROR

2.5 UPLOAD_STAT

Parameters: uint32 length; // length of image in bytes
Return: uint32 checksum; // sum of all bytes

RETURN_STATUS:
NO_ERROR

2.6 GET_RUNNING_IMAGE_VINFO

Parameters: none
Return:

```
typedef struct
{
      uint32   uiVProductID;
      char     uiVendorID[8];
      uint32   uiVMajor;
      uint32   uiVMinor;
      uint32   user1;
      uint32   user2;

}FL_GET_VENDOR_IMAGE_DESC_RETURN;
```

RETURN_STATUS:
NO_ERROR

3 Issuing Commands

In order to issue a command the host should first load the parameters for the command into the parameter space. After that the host should write the command into the opcode field with the execute bit set. The host should then wait for the execute bit to be cleared and read the RETURN_STATUS. If the status indicates success the return data can be read if any.

4 Byte ordering

The protocol is assuming network byte ordering of all uint32 units. Due to an idiosyncrasy in the device strings and the firmware image is assumed to be byte streams on a little-endian processor. The result of this is that strings and firmware image must be 32 bit byte swapped before being issued when running on a big-endian processor such as the PPC.

5 Determining the version of the running firmware

Before loading the firmware it might be convenient to determine the version of the current firmware to make sure that the firmware to load has a version number higher than the one running.
The version of the firmware application can be obtained with the GET_RUNNING_IMAGE_VINFO command.

6 Loading the firmware

The firmware is uploaded in blocks of up to 1024 bytes using the UPLOAD command. Start with index=0 and then increment in the following calls. Keep a 32 bit checksum of the sum of all the bytes uploaded. Remember to byte swap the image data on big-endian hosts.
When the firmware is uploaded issue the UPLOAD_STAT command and compare the checksum to the one held locally.

Now delete the old image using the DELETE_IMAGE command with the parameter “dice”. The firmware image is always called “dice”.
After that issue the CREATE_IMAGE command with the following parameters:

```
Length = <length of image in bytes>;
execAddr = 0x30000;
entryAddr = 0x30040;
name[16] = “dice”;
```

Remember to swap the string on big-endian hosts.

The delete and create commands will take some time to complete and during that time the device is quite busy so it should not be overloaded with polling commands. It is suggested that the execute flag being polled every 300ms or so.

After completion of the create command the firmware is ready to run. This will require a reset of the device which can be performed with the RESET_IMAGE command. Please note that the reset command will never complete. The device will reset and issue a bus-reset resulting in a re-enumeration by the host.

7 Loading the Setup image

The device contains another small image on top of the firmware image. This image contains the serial number of the device. The serial number is used as part of the WWUID. The setup image is a text file with one line formatted as follows:

SERIAL_NO=n<cr>

The serial number can be either decimal or hex (using 0x as prefix).
This image is programmed the same way as the firmware but the name is “setup” and the execAddr and entryAddr are ignored.

The WWUID is formed the following way:

24 bit OUI	4 bit category=0x4	10 bit product ID	22 bit serial no.

8 Handling serialization in production

Before production start a reference flash image is created with RedBoot, Setup and Application image. This flash image will have a known serial number (typically 0x3fffff). This image will be preprogrammed before mounting the flash.
When the device is connected to the test/serialization host it will always come up without asking for driver installation and the host can use the protocol above to change the setup image to contain the correct serial number. The host can also load the latest firmware replacing whatever was preprogrammed.

9 Test and Debug interface

If the firmware is compiled with the _FTM defined this interface exposes a number of functions to facilitate automated test and diagnostics. Those functions are used by the DICE tool. Some of the functionality also requires _AVSTRIG to be defined.

Most of the functions uses the command interface described above and adds functionality through a new op-code TEST_ACTION.
There is another part used for testing response to delayed split transactions using testDelay and TestBuf.

9.1 TEST_ACTION

Parameters:

```
typedef struct
{
      uint32      cmdID;
      uint32      lvalue0;
      uint32      lvalue1;
}FL_UPLOAD_PARAM;
```

Return:

```
typedef struct {
      uint32 data[100];
} FL_TEST_RTN;
```

RETURN_STATUS:
E_FAIL if cmdID not implemented, else NO_ERROR

There are several sub-tests available using cmdID. The commands and their parameters are listed in the following sub-sections.

9.1.1 TEST_CMD_POKE = 1

This command allows writing of any location in the memory space addressable by the ARM processor. It is typically used to set peripheral registers such as router tables.
Parameters:
lvalue0 = 32 bit linear address to write (must be 32 bit aligned)
lvalue1 = 32 bit value to write to the address.

9.1.2 TEST_CMD_PEEK = 2

This command allows reading of any location in the memory space space addressable by the ARM processor. It is typically used to read peripheral registers such as router tables.
Parameters:
lvalue0 = 32 bit linear address to read (must be 32 bit aligned)
Return
Data[0] = 32 bit value read from the address.

9.1.3 CMD_GET_AVS_CNT = 3

This command returns the AVS error counters. The number of counters differs from chip to chip. DICE II return 92 counters, DICE JR/Mini return 44.

Return
Data[ ] = counters.

9.1.4 CMD_CLR_AVS_CNT = 4

This command takes no parameters and returns no data. It will clear all the AVS counters.

9.1.5 CMD_SET_MODE = 5

This command sets the mode of the EVM boards. For EVM002 the mode change is immediate. For Classic EVM a reboot is required (might be changed in future impl.).
Parameters:
lvalue0 = mode to set, see EVM documentation for modes.

9.1.6 CMD_SET_MIDIBP = 6

This command will instruct the EVM to bypass all 1394 MIDI streams so all streams coming in will go back out on 1394.
Parameters:
lvalue0 = 0: No Bypass, 1: Bypass

9.1.7 CMD_GET_AVSPHASE = 7

Return the relative phase between PLL and AVS receiver.
Parameters:
lvalue0 = 0:avs1, 1:avs2
Return
Data[0] = signed phase offset in steps of 0.1us.
Data[1] = signed phase offset in %.

9.1.8 CMD_ENABLE_BNC_SYNC = 8

This function enables the use of the WCLK output of the DICE to be used as a trigger output pulsing when AVS errors occur or when the PULSE_BNC_SYNC command is issued.
A customer device can implement the targetBoard.c such that other signals are used for the trigger output. Both EVM002 and Classic EVM use the WCLK BNC output for this trigger.

9.1.9 CMD_PULSE_BNC_SYNC = 9

This function will pulse the sync output. The BNC sync must be enabled before this function will work. The purpose of this is to allow the DICE tool running on the host to signal a trigger condition to external hardware.

9.1.10 CMD_EMUL_SLOW_CMD

This dummy command will run a busy wait loop for as many ms as specified in the parameter. This will effectively stop the response to async. Communication until the time has passed.
Parameters:

lvalue0 = ms to wait

9.2 Testing Split transaction timeout

When a host accesses the DICE device with for example a read command a split transaction will occur. A response is immediately sent back to the host to indicate that the request was received. When the firmware is done processing the request a complete response with the read data is sent back.
In order to test how various hosts react to long response times, functionality has been added to simulate various response times.
The two memory variables TestDelay and TestBuf defined in section 2) are used for this purpose. Only split read transactions can be tested.
Whenever reading one or more (max 32) quadlets starting at TestBuf the split transaction response will be delayed for as many ms as specified in TestDelay. The wait is not busy, but no more transactions will be processed until the time has elapsed.