Design and manufacture systems with flash memory

IN THIS DOCUMENT

- Boot a program from flash memory
- Generate a flash image for manufacture
- Perform an in-field upgrade
- Customize the flash loader

xTIMEcomposer can be used to target xCORE devices that use SPI flash memory for booting and persistent storage. The xCORE flash format is shown in Figure 1.

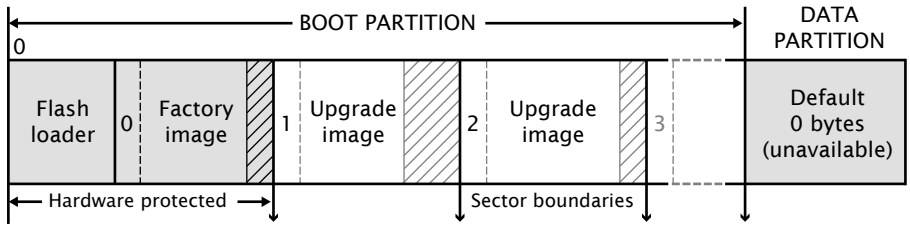

Figure 1: Flash format diagram

The flash memory is logically split between a boot and data partition. The boot partition consists of a flash loader followed by a "factory image" and zero or more optional "upgrade images." Each image starts with a descriptor that contains a unique version number, a header that contains a table of code/data segments for each tile used by the program and a CRC. By default, the flash loader boots the image with the highest version with a valid CRC.

1 Boot a program from flash memory

To load a program into an SPI flash memory device on your development board, start the command-line tools (see XM-000950-PC) and enter the following commands:

1. `xflash -l`

   XFLASH prints an enumerated list of all JTAG adapters connected to your PC and the devices on each JTAG chain, in the form:

   ```
   ID        Name        Adapter ID        Devices
   --        ----        ----------        -------
   ```

2. `xflash --id ID` *`program`*`.xe`

Publication Date: 2013/11/11 REV D

XFLASH generates an image in the xCORE flash format that contains a first stage loader and factory image comprising the binary and data segments from your compiled program. It then writes this image to flash memory using the xCORE device.

The XN file used to compile your program must define an SPI flash device and specify the four ports of the xCORE device to which it is connected (see XM-000929-PC).

2 Generate a flash image for manufacture

In manufacturing environments, the same program is typically programmed into multiple flash devices.

To generate an image file in the xCORE flash format, which can be subsequently programmed into flash devices, start the command-line tools (see XM-000950-PC) and enter the following command:

- `xflash` *program*`.xe -o image-file`

XFLASH generates an image comprising a first stage loader and your program as the factory image, which it writes to the specified file.

3 Perform an in-field upgrade

xTimeComposer and the libflash library let you manage multiple firmware upgrades over the life cycle of your product. You can use XFLASH to create an upgrade image and, from within your program, use libflash to write this image to the boot partition. Using libflash, updates are robust against partially complete writes, for example due to power failure: if the CRC of the upgrade image fails during boot, the previous image is loaded instead.

3.1 Write a program that upgrades itself

The example program in Figure 2 uses the libflash library to upgrade itself.

The call to `fl_connect` opens a connection between the xCORE and SPI devices, and the call to `fl_getPageSize` determines the SPI device's page size. All read and write operations occur at the page level.

The first upgrade image is located by calling `fl_getFactoryImage` and then `getNextBootImage`. Once located, `fl_startImageReplace` prepares this image for replacement by a new image with the specified (maximum) size. `fl_startImageReplace` must be called until it returns 0, signifying that the preparation is complete.

The function `fl_writeImagePage` writes the next page of data to the SPI device. Calls to this function return after the data is output to the device but may return before the device has written the data to its flash memory. This increases the amount of time available to the processor to fetch the next page of data. The function `fl_endWriteImage` waits for the SPI device to write the last page of data to

```
#include <platform.h>
#include <flash.h>

#define MAX_PSIZE 256

/* initializers defined in XN file
 * and available via platform.h */

fl_SPIPorts SPI = { PORT_SPI_MISO,
                    PORT_SPI_SS,
                    PORT_SPI_CLK,
                    PORT_SPI_MOSI,
                    XS1_CLKBLK_1 };

int upgrade(chanend c, int usize) {

  /* obtain an upgrade image and write
   * it to flash memory
   * error checking omitted */

  fl_BootImageInfo b;
  int page[MAX_PSIZE];
  int psize;

  fl_connect(SPI);

  psize = fl_getPageSize();
  fl_getFactoryImage(b);
  fl_getNextBootImage(b);

  while(fl_startImageReplace(b, usize))
    ;
  for (int i=0; i page[j];)
    fl_writeImagePage(page);

  fl_endWriteImage();

  fl_disconnect();

      return 0;
}

int main() {
  /* main application - calls upgrade
   * to perform an in-field upgrade */
}
```

Figure 2: C program that uses libflash to upgrade itself

its flash memory. To simplify the writing operation, XFLASH adds padding to the upgrade image to ensure that its size is a multiple of the page size.

The call `fl_disconnect` closes the connection between the xCORE and SPI devices.

3.2 Build and deploy the upgrader

To build and deploy the first release of your program, start the command-line tools (see XM-000950-PC) and enter the following commands:

1. `xcc` *file*`.xc -target=`*boardname* `-lflash -o first-release.xe`

   XCC compiles your program and links it against libflash. Alternatively add the option -lflash to your Makefile.

2. `xflash first-release.xe -o manufacture-image`

   XFLASH generates an image in the xCORE flash format that contains a first stage loader and the first release of your program as the factory image.

To build and deploy an upgraded version of your program, enter the following commands:

1. `xcc` *file*`.xc -target=`*boardname* `-lflash -o latest-release.xe`

   XCC compiles your program and links it against libflash.

2. `xflash --upgrade` *version* `latest-release.xe -o upgrade-image`

   XFLASH generates an upgrade image with the specified version number, which must be greater than 0. Your program should obtain this image to upgrade itself.

If the upgrade operation succeeds, upon resetting the device the loader boots the upgrade image, otherwise it boots the factory image.

4 Customize the flash loader

xTIMEcomposer lets you customize the mechanism for choosing which image is loaded from flash. The example program in Figure 3 determines which image to load based on the value at the start of the data partition.

The xCORE loader first calls the function `init`, and then iterates over each image in the boot partition. For each image, it calls `checkCandidateImageVersion` with the image version number and, if this function returns non-zero and its CRC is validated, it calls `recordCandidateImage` with the image version number and address. Finally, the loader calls `reportSelectedImage` to obtain the address of the selected image.

To produce a custom loader, you are required to define the functions `init`, `checkCandidateImageVersion`, `recordCandidateImage` and `reportSelectedImage`.

The loader provides the function `readFlashDataPage`.

```
extern void *readFlashDataPage(unsigned addr);

int   dpVersion;
void *imgAdr;

void init(void) {
  void *ptr = readFlashDataPage(0);
  dpVersion = *(int *)ptr;
}

int checkCandidateImageVersion(int v) {
  return v == dpVersion;
}

void recordCandidateImage(int v, unsigned adr) {
  imgAdr = adr;
}

unsigned reportSelectedImage(void) {
  return imgAdr;
}
```

Figure 3: C functions that customize the flash loader

4.1 Build the loader

To create a flash image that contains a custom flash loader and factory image, start the command-line tools (see XM-000950-PC) and enter the following commands:

1. `xcc -c` *file*`.xc -o loader.o`

   XCC compiles your functions for image selection, producing a binary object.

2. `xflash` *bin*`.xe --loader loader.o`

   XFLASH writes a flash image containing the custom loader and factory image to the specified file.

4.2 Add additional images

The following command builds a flash image that contains a custom flash loader, a factory image and two additional images:

- `xflash` *factory*`.xe --loader loader.o --upgrade 1 usb.xe 0x20000 --upgrade 2 avb.xe`

The arguments to `--upgrade` include the version number, executable file and an optional size in bytes. XFLASH writes each upgrade image on the next sector boundary. The size argument is used to add padding to an image, allowing it to be field-upgraded in the future by a larger image.