

## 7 Device Manufacture and Test

---

### In This Chapter

- ▶ Introducing the XMOS flash file format
  - ▶ Booting a device from flash memory
  - ▶ Producing field-upgradeable programs
  - ▶ Deploying the AES Security Module
  - ▶ Programming the XCore OTP memory
- 

The XMOS tools support the deployment of systems in external flash memory or internal one-time programmable (OTP) memory.

An XMOS device can be made to boot standalone from either external flash (typically the default boot mode of a development board unless reset by a debugger) or from OTP memory by writing a boot register.

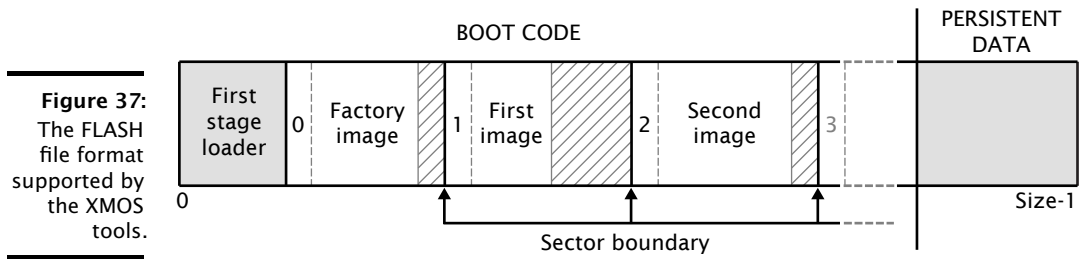
An AES Security Module can also be programmed into the OTP memory to enable programs to be encrypted on flash memory. This helps prevent programs residing on external flash memory from being reverse engineered or tampered with.

### 7.1 The XMOS Flash File Format

The XMOS tools support the FLASH file format shown in Figure 43. The flash device is logically split between boot and persistent storage, with independent read/write access to each area.

The boot area consists of a first-stage boot loader followed by a set of program images. The first image is referred to as the “factory image” and is numbered 0. Each image starts with a unique number (greater than 0) and is followed by a header that contains table of code/data segments for each core used by the program. Each image also contains a CRC. Except for the factory image, all other images are guaranteed aligned on sector boundaries.

The first-stage loader loads the image with the highest number that validates against its CRC.



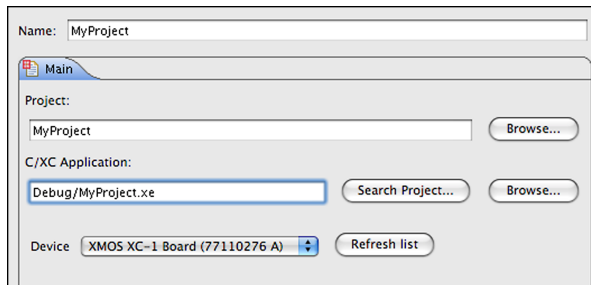
## 7.2 Loading a Program into Flash Memory

To load a program onto a flash device that is connected to an XMOS device on your development board, follow these steps:

1. Select your project in the *Project Explorer*.
2. Select **Run ► Run Configurations**.  
The Run Configuration dialog box opens.
3. Double-click **Flash Programmer**.

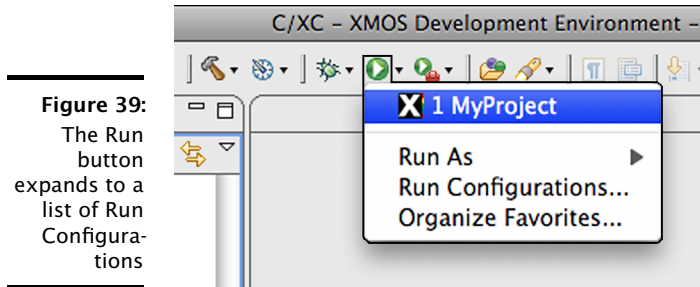
XDE creates a new configuration and opens the *Configuration Properties* window. The name, target project and target executable should be automatically configured based on the selected project. If not, click **Browse** and select your project from the *Project Selection* dialog box, and click **Search Project** and select the executable file.

**Figure 38:** Use the Configuration Properties window to configure your target platform.



4. Select your target board from the *Board* list.  
If your board does not include flash memory, its name will appear greyed out.
5. If available, select the **Boot after Flash** check box to instruct XDE to reset the development board and boot from flash memory upon successfully flashing the device.  
Not all development boards support this option.
6. Click **Apply** to save the configuration settings and then click **Close** to close the dialog box.

7. Click the arrow next to the **Run** button and select your new Run Configuration from the drop-down list.
8. Click the arrow next to the **Run** button, as shown in Figure 8. and select your new Run Configuration from the drop-down list.



XDE writes your program to flash memory, outputting its progress in the *Console* view.

If you did not check the Boot after Flash check box, manually reset your development board to boot the program from flash.

## 7.3 In-Field Upgrades

Developers who need to provide in-field upgrades to the flash can access the libflash library from their programs.

The program in Figure 40 shows how to use the libflash API with standard I/O operations to retrieve an upgrade boot image. Typically, this image would be obtained from a network or a removable storage device such as an SD card.

The function `upgrade` performs the following operations:

- Calls the function `f1_connect`, which opens a connection between the XCore and flash device.
- Calls the function `f1_getFirstBootImage`, which obtains a handle to the factory image and writes it to the variable `f1_BootImageInfo`.
- Calls the function `f1_addBootImage`, which writes the image in `BufferHandle_t` to the memory at the next sector boundary following the factory image.

The function `f1_addBootImage` takes as arguments the number of bytes to write and a function pointer to the callback function `getData` that provides the data in 256-byte chunks.

- Calls the function `f1_disconnect`, which closes the connection between the XCore and flash device.

```

#include <stdio.h>
#include <stdlib.h>
#include <flashlib.h>

typedef struct {
    void *dataPtr;
    int pos;
} bufferHandle_t;

unsigned int getData(void *img, unsigned n, unsigned char *dstBuf) {
    /* Call-back function that provides the next n-bytes of data
    bufferHandle_t *b = img;
    memcpy(b->dataPtr+pos, dstBuf, n);
    b->pos += n;
    return n;
}

int upgrade(char img[], int nbytes) {

    bufferHandle_t b = { img, 0 };
    fl_BootImageInfo bii;

    /* Connect to the flash chip */
    fl_connect();

    /* Locate the factory boot image and obtain handle to bii */
    fl_getFirstBootImage(&bii);

    /* Add the upgrade image immediately after the factory image */
    fl_addBootImage (&bii, nbytes, 0x1, &getData, (void*)b)

    /* Disconnect from the flash chip */
    fl_disconnect();

    return 0;
}

```

**Figure 40:**  
A simple  
program that  
field-  
upgrades its  
own flash  
memory

Note that if the writing operation fails, the XCore will load the factory image the next time the XCore is booted, otherwise it will load the upgrade image. It is up to the developer to determine the policy of where to write the upgrade image in the boot address space. The developer is also responsible for providing a mechanism for reverting to the factory boot image, for example by providing a reset button.

## 7.4 Securing Flash-Based Systems

XMOS devices are provided on-chip with one-time programmable (OTP) memory that can be blown during or after device manufacture testing. Additional security bits can be blown in the OTP, transforming an XCore into a secure island

in which all computation, memory access, I/O and communication are under exclusive control of the program running on the core.

When set, these bits:

- Force boot from OTP to prevent bypassing
- Disable JTAG access to the XCore to prevent reading
- Stop further writes to OTP to prevent updates

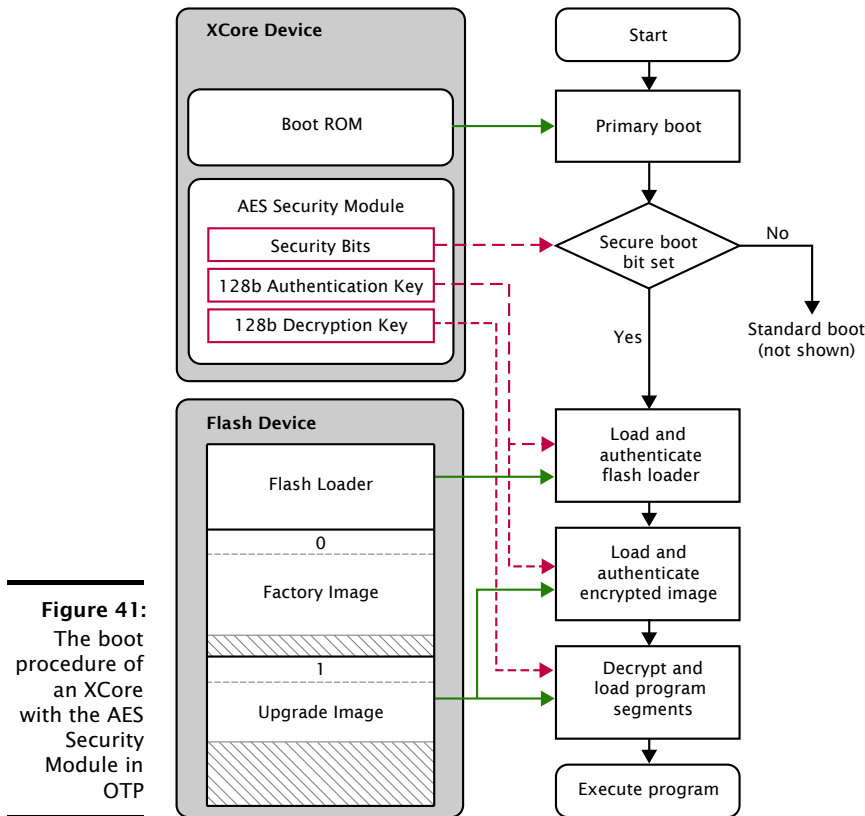
The XMOS tools include an *AES Security Module* that can be programmed into the on-chip OTP memory and provides:

<b>Secrecy</b>	Prevents a program stored in flash memory from being reverse engineered
<b>Program Authenticity</b>	Prevents a program stored in flash memory from being tampered with, and prevents third-party IP from being run on a device.
<b>Device Authenticity</b>	Prevents a program stored in flash memory from being loaded onto an XMOS device provided by a third-party.

XBURN is used to program the security module into the OTP memory of a XMOS device.

The AES Security Module authenticates and decrypts programs from SPI flash devices. The secure boot procedure, which is shown in the diagram in Figure 41, proceeds as follows:

- The XCore loads the primary bootloader from its ROM, which detects that the secure boot bit is set in the OTP and then loads and executes the AES security module from OTP.
- The AES Security Module loads the flash bootloader into RAM over SPI.
- The Security Module authenticates the bootloader using the CMAC-AES-128 algorithm and the 128-bit authentication key. If authentication fails, boot is halted.
- The Security Module places the authentication key and decryption key in registers and jumps to the flash bootloader.
- The flash bootloader selects the image with the highest number that validates against its CRC.
- The flash bootloader authenticates the selected image header using its CMAC tag and authentication key. If the authentication fails, boot is halted.
- The flash bootloader authenticates, decrypts and loads the table of program/-data segments into memory. If any images fail authentication, the boot halts.
- The flashbootloader starts executing the program.



### 7.4.1 Enabling the AES Security Module

The security module can be loaded directly into OTP using XBURN, or it can be generated as a separate secure bootloader image and passed to a third party consolidator to load using XRUN—see [Figure 42](#)

#### Loading the security module directly to OTP

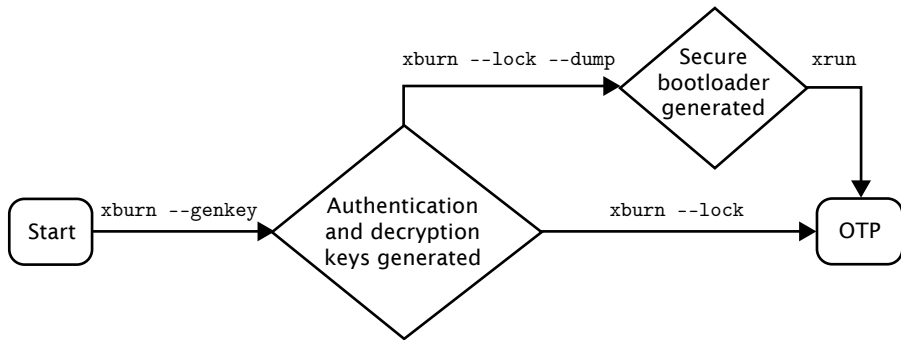
Follow these steps:

1. Start the command-line tools. (See [Chapter 1](#).)
2. Generate the authentication and decryption keys. Type:
 

```
xburn --genkey keyfile
```

Two keys are written to `keyfile` as 128-bit values on two lines. The algorithm for generating a random key uses the open-source library `crypto++`. If you prefer, you can create this file yourself and provide your own keys.

**Figure 42:**  
The secure  
bootloader  
image  
generation  
procedure



- Identify the device to program. Type:

```
xburn -l
```

A list of devices connected to the host system is displayed in the form

```
Available XMOS Devices
```

```
-----
```

```
0 - XMOS XTAG          (78450068A)
```

- Identify the number of the target device and type:

```
xburn --device id --lock keyfile -targetfile target.xn
```

The *target.xn* file must reflect the correct device (for example XS1-G4B-FB144) and the ports the SPI device is connected to.

XBURN writes the AES security module and authentication/encryption keys into the OTP memory. The secure boot bit is set, JTAG access to the XCore is disabled and the OTP is locked to prevent further modifications.

To retain JTAG access to the locked device, you must pass the following options at the same time as `--lock`:

```
--enable-jtag --disable-master-lock
```

This allows the program to be debugged but exposes the keys to the debugger. JTAG access can be disabled later using:

```
xburn --device n --disable-jtag -targetfile target.xn
```



XFLASH must be used to program the FLASH before JTAG is disabled. XFLASH cannot be used after JTAG has been disabled. An external programmer can still be used to reprogram the FLASH device directly.

### Outputting a secure bootloader image

The `--dump` option instructs XBURN to output a bootloader image instead of loading it into the OTP. To create a secure bootloader image follow these steps:

- Generate the authentication and decryption keys. Type:

```
xburn --genkey keyfile
```

2. Output the secure bootloader image  

```
xburn --lock keyfile --dump-xe loader.xe --targetfile target.xn
```
3. The secure bootloader image can now be loaded using XRUN:  

```
xrun --device id loader.xe
```

#### 7.4.2 Encrypting Programs on Flash

XDE uses a *Run Configuration* to save the settings used when loading a program onto SPI memory. To encrypt and load a program binary into SPI memory, follow these steps:

1. Select your project in the *Project Explorer*.
2. Select **Run ► Run Configurations**.  
The Run Configuration dialog box opens.
3. Double-click **Flash Programmer**.  
XDE creates a new configuration and opens the *Configuration Properties* window.  
The name, target project and target executable should be automatically configured based on the selected project. If not, click **Browse** and select your project from the *Project Selection* dialog box, and click **Search Project** and select the executable file.
4. Select your target board from the *Board* list.
5. Click the **Browse** button next to the AES Keys text box and select your key-file.
6. Click **Apply** to save the configuration settings and then click **Close** to close the dialog box.
7. Click the arrow next to the **Run** button and select your new Run Configuration from the drop-down list.  
XDE writes your program to flash memory, outputting its progress in the *Console* view.

## F XRUN Command Line Usage

XRUN is the XMOS loader. It loads XE files generated by the compiler toolchain onto a network of one or more devices via a JTAG interface.

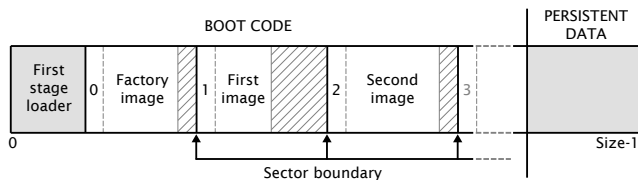
- `--listdevices`  
Prints (on standard output) an itemised list of the available XMOS devices connected to the development system and their serial numbers.
- `--device n`  
Specifies the device to target.  
The device number `n` can be found by first typing `xrun --listdevices`.
- `--serial id`  
Specifies the device to target by its serial number.
- `--core name`  
Loads the program on the core determined by the XC core variable `name`, for example `stdcore[0]`. This option is only permitted for single-core programs.
- `--core n,c`  
Loads the program on core `c` on node `n`. This option is only permitted for single-core programs.
- `--io`  
Runs an I/O server after loading the program that communicates with the device to enable system calls. XRUN terminates when the program performs a call to `exit`.
- `--attach`  
Attach to a running program and run an I/O server that enables system calls. XRUN terminates when the program performs a call to `exit`.
- `--noreset`  
Disables the automatic reset on connection. By default, the device is reset prior to loading the XE file.

- `--jtag-speed` Sets the divider for the JTAG clock. If unspecified, the default value is 0 (6MHz). Matt to update.
- `--dumpstate` Prints (on standard output) the thread, register and stack contents for all processors in the target system.
- `--verbose` Prints (on standard output) additional information about the program when loaded onto the target system.
- `--help`  
`-h` Prints (on standard output) a description of the supported command line options.
- `--version`  
`-v` Displays the version number and copyrights of the invoked XRUN.

## G XFLASH and the libflash Library

XFLASH is the XMOS Flash Programming Tool. It loads XE files generated by the compiler toolchain onto SPI FLASH memory attached to an XMOS device. Once loaded, the XMOS device can be booted standalone from FLASH memory. The XMOS tools support the FLASH file format shown in Figure 43.

**Figure 43:**  
The FLASH file format supported by the XMOS tools.



The FLASH device is logically split between boot and persistent storage, with independent read/write access to each area.

The boot area consists of a first-stage boot loader followed by a set of program images. The first image is referred to as the “factory image” and is numbered 0. Each image starts with a unique number (greater than 0) and is followed by a table of code/data segments for each core used by the program. Each image also contains a CRC. Except for the factory image, all other images are guaranteed aligned on sector boundaries.

The first-stage loader loads the image with the highest number that validates against its CRC.

XFLASH uses the libflash library (see X) to create an XE program that it loads onto an XCore via JTAG. This XE file uses system calls to read the image data from the host development platform and write it to FLASH.

## G.1 XFLASH Command Line Options

- `--listdevices`  
Prints (on standard output) an itemised list of the available XMOS devices connected to the development system and their serial numbers.
  
- `--device n`  
Specifies the device to target. The device number `n` can be found by first typing `xflash -listdevices`.
  
- `--dump-image file`  
`-o file` Place the image data in `file` instead of writing it to FLASH. The image contains a first-stage loader, and the code and data segments for all cores in the provided XE file.
  
- `--make-upgrade id`  
Produce `animage` with the specified `id`. The image contains the code and data segments for all cores in the provided XE file. This option is valid only with the `--dump-image` option.
  
- `--slimit address`  
Specifies the highest address in the FLASH that the first-stage loader searches for boot images.
  
- `--verbose`  
Prints (on standard output) additional information about the program when loaded onto the target system.
  
- `--help`  
`-h` Prints (on standard output) a description of the supported command line options.
  
- `--version`  
`-v` Displays the version number and copyrights of the invoked XFLASH.

## H XBURN Command Line Usage

XBURN is the XMOS OTP Programming Tool. It can be used to write a AES security module to the OTP of an XMOS device.

### H.1 Specifying an action

Exactly one of the following options must be passed to XBURN:

- `--genkey keyfile`  
Generates a pair of keys suitable for use with XBURN and XFLASH. The keys are written to the specified file.
- `--read`  
Displays (on standard output) the current contents of the OTP.
- `--burn file.xe`  
Writes an executable to the OTP as a bootable image.
- `--lock keyfile`  
Writes the AES security module to the OTP. The keys used for encryption and authentication are taken the specified key file.
- `-l|--listdevices`  
Lists the available XMOS devices on the machine.
- `-h|--help`  
Lists the commands supported by XBURN.
- `--version`  
Displays the version then exits.

### H.2 Specifying a target

A target must be specified whenever `--read` or `--lock` is used without `--flash`.

- `-target=platform`  
Specifies the target platform. The platform configuration must be specified in the file `platform.xn`, which is searched for in the paths specified by the `XCC_DEVICE_PATH` environment variable (see §C.10).
- `--targetfile platform.xn`  
Specifies the target platform. The platform configuration is read from the specified file.

### H.3 Security register options

XBURN provides options to set the OTP security register, enabling or disabling security features of the device. Every option has a positive (`--enable`) and negative (`--disable`) form. Only the positive form is listed below.

- `--enable-otp-boot`  
Enables boot from OTP.
- `--enable-jtag`  
Enables JTAG access. If JTAG access is disabled it will not be possible to gain debug access to the device or to read OTP via JTAG.
- `--enable-plink-access`  
Enables access to the plink registers from other cores. Disabling plink access restricts all access of the registers of each plinks to the core local to that plink.
- `--enable-global-debug`  
Allows the device to participate in global debug. If interaction with global debug is disabled it will not be possible to cause the cores to enter debug using the global debug pin.
- `--enable-master-lock`  
Enables the OTP master lock. Once the lock is enabled all further modification of the OTP is prevented.

When the state of a security feature is not explicitly set its value is determined by the default value for the action.

Feature	Action	
	<code>--lock</code>	<code>--burn</code>
OTP Boot	✓	✓
JTAG access	✗	✓
Plink access	✓	✓
Global debug	✗	✓
Master lock	✓	✗

Security register options may be passed to XBURN without an action. A target is required in this case. For example:

```
xburn --disable-jtag --targetfile platform.xn
```

XBURN will enable or disable just the features that are specified. All other features will remain unchanged.

### H.4 Other options

- `-d|--device n`  
Specifies the device to target.

- The device number `n` can be found by first typing `xburn --listdevices`.
- `-f|--force` Do not prompt before performing the specified action. By default XBURN asks for confirmation before performing an action which would modify the OTP.
  - `--flash file.xe` Flashes the specified file to the device using XFLASH prior to writing to the OTP. This option is only valid when use in conjunction with `--lock`.
  - `--dump-xe file.xe` The action is not performed on the OTP and instead an executable to perform the action is written to a file.