Application Note: AN00153

Programming OTP memory via SPI boot

This application note describes how to create a binary image which can be used to program the xCORE tile OTP memory via SPI boot. This is the XMOS recommended process for a production programmer which would be used to program application data into the OTP memory in volume production.

The code associated with this application note provides a simple example which demostrates building an application which then, using the XMOS development tools, produces a SPI flash image that can be used to boot the xCORE, program the OTP and then signal that the device has been programmed correctly via GPIO ports on the xCORE device.

With the development flow presented here it is possible to design and manufacture a programmer that could be used to program xCORE devices with custom firmware images as required.

Example code is provided to allow a simple example to be built into the SPI programming image for the xCORE OTP.

Required tools and libraries

- xTIMEcomposer Tools - Version 13.2

Required hardware

This application note is designed to run on any XMOS xCORE multicore microcontroller that is able to boot from SPI and has OTP memory available for programming.

The example code provided with the application has been implemented and tested on an XMOS general purpose sliceKIT (XK-SK-L2-ST) but there is no dependancy on this board and it can be modified to run on any board containing an XMOS xCORE processor.

Note that the application example can only be run once successfully on the target hardware as the OTP memory is one time programmable.

Prerequisites

- This document assumes familiarity with the XMOS xCORE architecture, the XMOS tool chain and the xC language. Documentation related to these aspects which are not specific to this application note are linked to in the *References* appendix.
- For descriptions of XMOS related terms found in this document please see the XMOS Glossary[1].
- The XMOS tools manual contains information regarding the use of xSCOPE and how to use it via code running on an xCORE processor[2].

[1] http://www.xmos.com/published/glossary
[2] http://www.xmos.com/published/xtimecomposer-user-guide

1 Overview

1.1 Introduction

XMOS devices contain secure on-chip one-time programmable (OTP) memory that can be blown during or after device manufacture testing. XMOS provides an Advanced Encryption Standard (AES) Module that authenticates and decrypts programs from external SPI flash devices. You can program the AES Module into the OTP of an xCORE device, allowing programs to be stored encrypted on flash memory.

This helps provide:

- Secrecy - Encrypted programs are hard to reverse engineer
- Program Authenticity - The AES loader will not load programs that have been tampered with or other third-party programs
- Device Authenticity - Programs encrypted with your secret keys cannot be cloned using XMOS devices provided by third parties

Once the AES Module is programmed, the OTP security bits are blown, transforming each tile into a "secure island" in which all computation, memory access, I/O and communication are under exclusive control of the code running on the tile.

When set, these bits:

- force boot from OTP to prevent bypassing
- disable JTAG access to the xCORE tile to prevent the keys being read stop further writes to OTP to prevent updates

The programming process proceeds as follows:

- The XMOS device boots from the Programming Image through the SPI pins
- The XMOS device executes the Programming Image, which blows the OTP target image into the OTP memory
- The XMOS device verifies that the OTP Target Image is now present in the OTP and signals status using GPIO

1.2 Block diagram

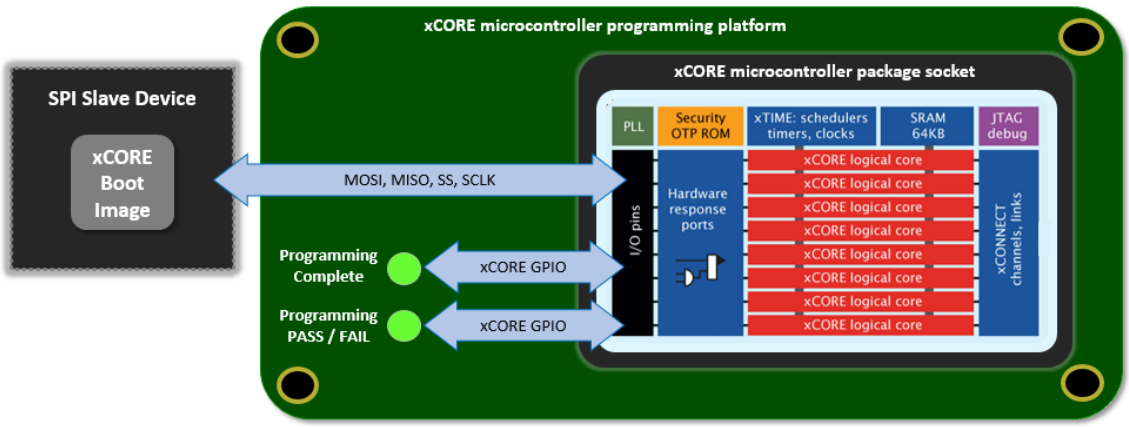

Figure 1: Block diagram of xCORE microcontroller programming platform

2 Programming OTP memory via SPI boot example

The example in this document does not have any external dependancies on application libraries other than those supplied with the XMOS development tools. It demonstrates how to generate an OTP programming image of an xCORE application which can signal its PASS / FAIL and completion status via the GPIO pins of an XMOS mulitcore microcontroller. This allows an external programming rig to automatically detect if the device under test has had the OTP memory programmed correctly.

In the example, the Programming Image is supplied to the XMOS device through a SPI slave interface on a flash memory that is itself programmed using the XMOS development tools via JTAG. In a production system the SPI slave interface may be provided directly from the programming equipment.

The following diagram shows the programming flow and the steps taken during the image generation process.

2.1 Programming Flow:

The following diagram describes the programming flow where the Programming Image is provided through a flash memory. If an alternative approach is used, for exmaple a SPI slave delivering the Programming image from an FPGA, then the SPI flash programming step will have to be modified accordingly.

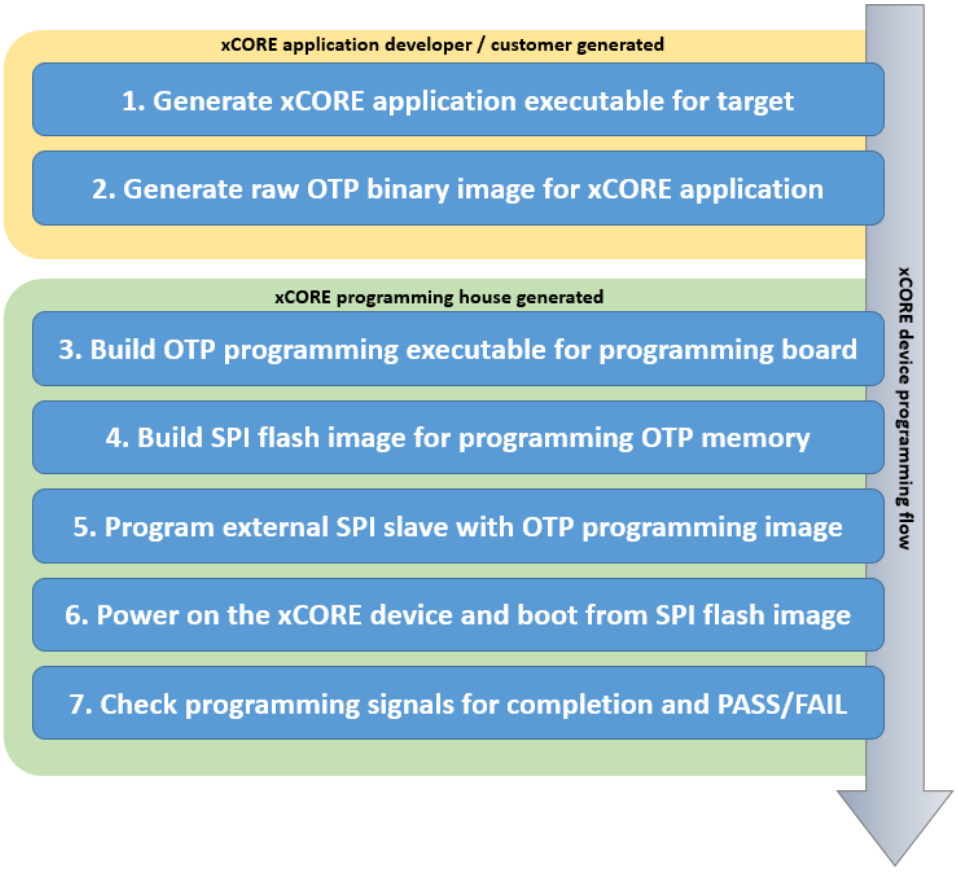

Figure 2: Programming flow for OTP memory via SPI flash

2.2 Building the xCORE application target image

This is the first step in the programming process and is normally done by the application developer or customer requiring the OTP to be programmed on the xCORE device. An xCORE application is built in the standard way using an xmos makefile and source code.

To demonstrate the approach, this example uses a simple main function which operates in a loop as follows.

```
#include <xs1.h>

out port leds = XS1_PORT_4E;

void drive_leds(void) {
  for (int i = 0; i < 5; i++) {
    delay_milliseconds(200);
    leds <: 0x0;
    delay_milliseconds(200);
    leds <: 0xf;
  }
}

int main (void) {
  drive_leds();
  return 0;
}
```

In this example you can see we are simply calling a function to flash an LED 5 times and then exiting the program.

To compile this application, startup a command prompt which uses the XMOS development tools, go to the top level of the application note directory structure and enter the command:

```
xmake all
```

2.3 Building the xCORE application OTP binary image

From the previous step we have a standard XMOS executable file which can be used with the XMOS development tools to run on the target hardware. To be able to program this code into OTP memory via a programming platform we need to generate the binary image of the data that would be programmed into the OTP, independantly of the hardware platform itself.

To proceed with the example, open a command prompt setup to use the XMOS development tools. Then change into the bin directory from the top level of the application note directory structure

First it is required to generate the program image for the specfied xCORE tile we are going to program. This is done with the following command sequence:

```
> xobjdump --split app_test_program.xe
> xobjdump --add 0,0,image_n0c0_2.elf otp_tile_image.xe
> xobjdump --add 0,0,..\src\platform\platform_def.xn otp_tile_image.xe
```

The binary data for the OTP image can now be produced.

In order to do this the xburn tool is used:

```
> xburn -o otp_tile_image.bin otp_tile_image.xe --target-file ..\src\platform\platform_def.xn
```

The output from this stage of the process is a binary image of the OTP contents to be programmed. This

is contained within the file otp_tile_image.bin.

2.4 Building the OTP programming application for the programming rig

At this stage in the process there is normally a handover of the binary image for the OTP memory between the customer and programming house. In this example both parts of the process are being performed on the same development board.

Once we have the binary image for the OTP from the last step an OTP Programming Image can be generated which will perform the actual writing and verification of data into the OTP memory. This is done again using the xburn tool.

For this step we need to link in some extra code into the programming executable in order to perform the signalling of the programming completion and also the PASS / FAIL status of writing to the OTP memory.

This is done by passing the following code to xburn on the command line.

```
#include <platform.h>

on tile[0] : out port complete_signal = XS1_PORT_4E;

void _done() {
  set_port_use_on(complete_signal);

  complete_signal <: 0xc;

  while (1);
}

void _exit(int exitcode) {
  set_port_use_on(complete_signal);

  if (exitcode != 0) {
    complete_signal <: 0xe;
  } else {
    complete_signal <: 0xc;
  }

  while (1);
}
```

From this source code you can see the following,

- A GPIO is declared to report status complete and the status of PASS / FAIL
- Standard xCORE GPIO ports are used and these can be changed depending on where the detection of these signal is connected
- The standard runtime function _done() is overloaded by this code to report the status that programming the OTP has completed and been a success. This function is called by the OTP programming application when there has been no error in programming
- The standard runtime function _exit is overloaded by this code to report the status that programming the OTP has completed and has either passed or failed. This function will be called by there OTP programming application when there has been an error in programming the OTP.
- Both of the above functions wait forever after signalling the status of the programming operation

In order to generate the OTP programming executable the following xburn command line is used:

```
> xburn --make-exec otp_programmer.xe --extra-file ..\src\signalling\portsignal.xc otp_tile_image.bin -target=
  ↪ XK-1A
```

At this point in the process a decision has to be made about various security options which are required to be passed to xburn as command line options, the developer requesting OTP programming would have specified these options,

- Enable boot from OTP
- Enable JTAG access
- Enable access to plink registers from other cores
- Enable participation of device in global debug
- Enable OTP master lock

2.5 Building the SPI flash boot image for programming

Once the OTP Programming Image is produced from the previous step it now needs to be translated into a binary image which can reside in a SPI salev device (in this case a flash memory) and boot the xCORE device. The act of booting this device will load the programming code from SPI slave device into the xCORE SRAM memory and then execute the code which will in turn program the application image we built earlier into the OTP memory of the xCORE device.

The OTP programming application is converted to a SPI flash image by using the `xflash` tool supplied with the XMOS development tools.

The command line to convert the OTP programming image using `xflash` is as follows:

```
> xflash -o otp_programmer.bin otp_programmer.xe -target=SLICEKIT-L16 --noinq --boot-partition-size 0x10000
```

This command produces a binary image which is in the correct format to be programmed into a SPI flash that will boot an xCORE device via its boot ROM.

This image can be used with an external programmer to load into the SPI slave device.

APPENDIX A - Example Hardware Setup

This example is designed to run on an XMOS general purpose sliceKIT (XK-SK-L2-ST) but the target can be changed by editing the Makefile supplied to select a different target board. The hardware for running the example is shown below.

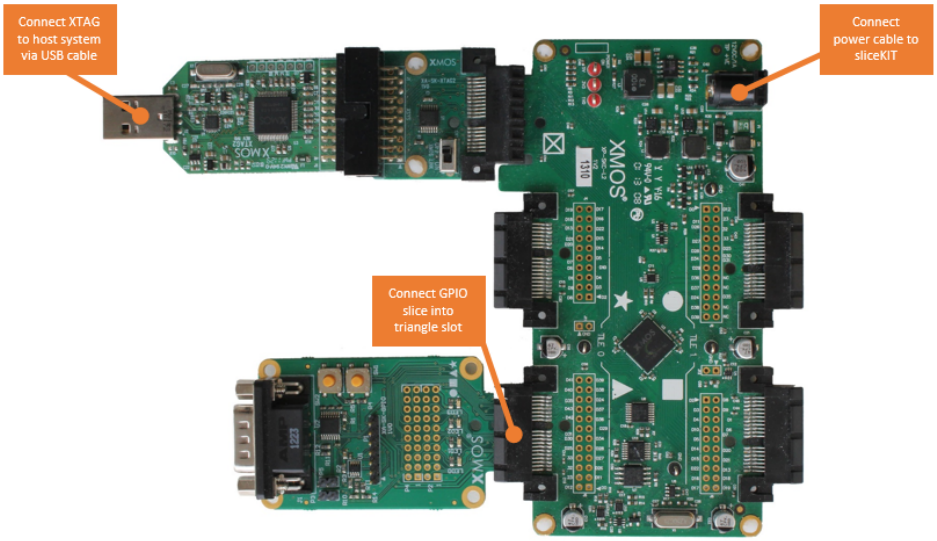

Figure 3: XMOS startKIT

The hardware should be configured as displayed above for this example :

- The XMOS XTAG should be connected to the host machine via a USB cable
- The GPIO slice card should be connected into the triangle connector on the sliceKIT
- The sliceKIT should have its power connector plugged in an powered on

APPENDIX B - Testing the completion and PASS / FAIL signalling

In order to check that our hardware is correctly signalling that programming is complete and if the status is a PASS or a FAIL we can run some simple code on the device using the GPIO signalling code we compiled into our programming binary. This will generate an executable which can be loaded onto the target.

The signalling of programming status is shown on the LED's of the GPIO slice connected as shown in the example hardware setup.

The completion status is reported on LED0 and the PASS/FAIL status on LED1.

We are using the XMOS command line tools, make sure you are running in a console with the XMOS development tools environment set.

From the top level of the application note directory change directory into the following location:

```
> cd src\signalling
```

In order to compile the pass and fail binaries from the XMOS command prompt then perform the following operations:

```
> xcc -o pass.xe pass.c portsignal.xc -target=SLICEKIT-L16
> xcc -o fail.xe fail.c portsignal.xc -target=SLICEKIT-L16
```

Once these are compiled we can run them on the board to check the PASS status:

```
> xrun pass.xe
```

The LED's on the GPIO slice should look as below.

Figure 4: Complete signalled by LED0 on and PASS signalled by LED1 on

And check FAIL status is being reported:

```
> xrun fail.xe
```

The LED's on the GPIO slice should look as below.

Figure 5: Complete signalled by LED0 on and FAIL signalled by LED1 off

APPENDIX C - Programming the application code into the device

C.1 Programming the SPI flash on the board

In this example we are using the XMOS development tools to program the SPI flash binary image into the flash device. The binary file we generated earlier would, in a production programmer, be used either with an external flash programmer, or with a programmable device that emulates a SPI flash.

In order to program the flash the `xflash` command is used with the binary image generated earlier:

```
> xflash --write-all otp_programmer.bin -target=SLICEKIT-L16
```

Once this command has completed the SPI flash will be loaded with our programming image.

C.2 Programming the OTP memory on the device

Now that the Programming Image is present in the programming hardware, the XMOS device now needs to be power cycled so that it can boot from the SPI slave device. In the example, this is done by removing and reinserting the power cable to the XMOS sliceKIT.

Once this is done the OTP will be programmed and the board will now be configured to always boot from OTP memory. You will no longer be able to boot the device from SPI or JTAG boot modes after this operation is complete

APPENDIX D - References

XMOS Tools User Guide

http://www.xmos.com/published/xtimecomposer-user-guide

XMOS xCORE Programming Guide

http://www.xmos.com/published/xmos-programming-guide

APPENDIX E - Full source code listing

E.1 xCORE source code for main.xc

```
#include <xs1.h>

out port leds = XS1_PORT_4E;

void drive_leds(void) {
  for (int i = 0; i < 5; i++) {
    delay_milliseconds(200);
    leds <: 0x0;
    delay_milliseconds(200);
    leds <: 0xf;
  }
}

int main (void) {
  drive_leds();
  return 0;
}
```

E.2 xCORE source code for portsignal.xc

```
#include <platform.h>

on tile[0] : out port complete_signal = XS1_PORT_4E;

void _done() {
  set_port_use_on(complete_signal);

  complete_signal <: 0xc;

  while (1);
}

void _exit(int exitcode) {
  set_port_use_on(complete_signal);

  if (exitcode != 0) {
    complete_signal <: 0xe;
  } else {
    complete_signal <: 0xc;
  }

  while (1);
}
```