Application Note: AN00238

SPI Slave Boot Loader

xTIMEcomposer tools support booting over JTAG or from a SPI flash, optionally with encryption and authentication using on-chip OTP. xCORE-200 devices also have a SPI slave boot mode. With a simple custom boot loader, it is possible to boot a two-tile XU216/XL216 device as SPI slave, with another device, such as applications processor, supplying the image to boot, and taking care of any extra features such as firmware upgrade.

This document describes how to use a custom boot loader that was developed for this.

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].

[1] http://www.xmos.com/published/glossary

1 Version history

1.0.0

- Initial version

1.0.1

- Add an overview paragraph
- Split setup steps into install and configure
- Remove xTIMEcomposer project files (this version command line only)
- Add a note about loader's USB setup code on XL parts

1.0.2

- Add a demonstration using eXplorerKIT boards
- Explicitly say which directories to go to for compile
- Clarify supported xTIMEcomposer versions
- Add a timing diagram of SPI transaction

2 Overview

Building and using the boot loader is a two stage process. The first stage is to build the boot loader. This stage creates 3 files (loader_pre_crt.o, loader_init.o and libloader.a) that are then imported, in the second stage, into the application to allow it to boot from an external SPI master processor. The use of the boot processor in the end application does not require any modifications to the application code, only modification of the makefile to ensure that the boot loader is installed.

3 Usage

3.1 Install loader

1. Go in the *loader* directory and compile loader (optional step as binaries are included):

```
make
```

2. Copy loader binaries to the directory where your application makefile is. This app note has an *example* directory:

```
loader_pre_crt.o
loader_init.o
libloader.a
```

3.2 Configure application

1. Build your application with extra build options. This app note has a makefile with these in the *example* directory to illustrate:

```
XCC_EXTRA_MAP_FLAGS = -L .. -lloader -Xmapper --first -Xmapper ../loader_pre_crt.o ../loader_init.o
```

2. Generate SPI image for your host processor. You can find the script in the *loader* directory and the default binary produced in the *example* directory will be *a.xe*:

```
loader/generate_spi_slave_image img.bin example/a.xe
```

3.3 Wiring

Connect SS to pin X0D00 (note this is different from SPI master where SS is X0D01), SCLK to pin X0D10 and MOSI to X0D11. MISO is not used (high-impedance with weak internal pull down).

These are ports 1A, 1C and 1D on tile 0 that are free to use in application (you can have other drivers on those I/O as long as host can still drive SPI during boot).

Remember to select the correct boot mode by adding a 3k3 pull-up on X0D05, leaving X0D04 and X0D06 unconnected (internal pull-down).

3.4 Host configuration

Send the image from your host as one continuous transaction, with SS asserted once at the beginning and deasserterted once at the end.

Least significant bit first. SPI is often MSb first, but xCORE's built-in ROM requires data to be LSb first.

SPI mode 0, toggle on falling edge, xCORE will sample on rising edge.

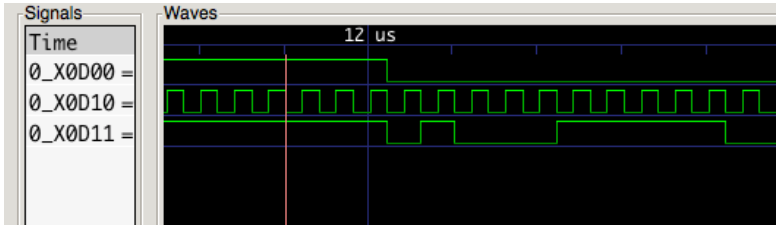

Figure 1: Start of SPI slave boot. First 4 bytes of data is image size in words. Here E2 03, 994 words.

xCORE is ready to receive image around 1ms after released from reset. There is no indication of xCORE ready in SPI slave mode, but if you switch it to SPI master temporarily (remove X0D05 pull-up), SPI activity should begin in around 30us. SPI slave will be ready in a similar amount of time.

Tested with clock speeds from 100kHz to 5MHz. Should be able to go faster.

3.5 xCORE-200 eXplorerKIT demonstration

If you need a working setup to debug against, you can use a pair of eXplorerKIT boards.

3.5.1 Slave

Example application that comes with this app note will run on the eXplorerKIT board and flash its RGB LED. This will be the SPI slave bootee. You can first try running it over JTAG. From the *example* directory:

```
make && xrun a.xe
```

To set boot mode on the slave, X0D05 is on test point TP24 as well as pin 2 of QSPI flash U2. You can solder a resistor to 3V3 somewhere nearby such as the flash's pin 8. For SS, SCLK and MOSI connections, use test points TP27, TP28 or TP30. SCLK is also on flash pin 6, same as the SPI master SCLK.

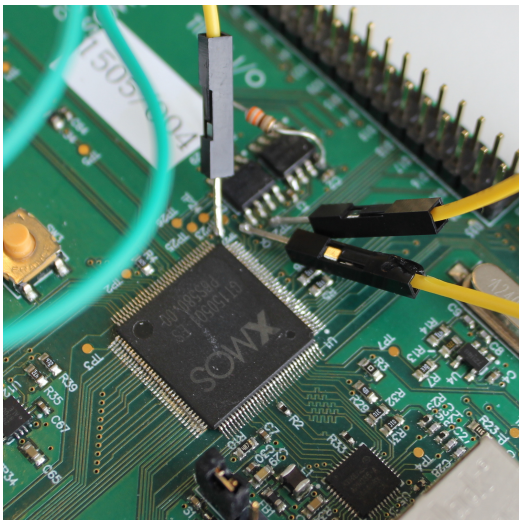

Figure 2: eXplorerKIT modified as SPI slave

Please refer to eXplorerKIT hardware manual for schematics and further details.[2]

Next, reset or power cycle the board and check the XU216 is now in SPI slave boot mode. You will see tile 0 of the XU216 in 0xfff005b8 for SPI master (out-of-the-box unmodified board). For SPI slave it will be 0xfff00734. Note that tile 1 might be listed first in the output of xrun:

```
xrun --dump-state-no-xe
```

Remember to reset the board again before starting the master because wires may have picked up noise and generated erroneous data on the SPI bus.

[2] https://www.xmos.com/published/xcore-200-explorerkit-hardware-manual

3.5.2 Master

Any board will do for the SPI master. Another eXplorerKIT is useful and has pin headers that we need. This app note comes with a programmer app that runs on an XMOS board, acts as a SPI master and sends out your image to the slave. Compile the app first, in the *programmer* directory:

```
make IMG=../img.bin
```

The app is configured to use port 1E for SS, 1F for SCLK and 1G for MOSI. So connect your slave SS to X0D12, SCLK to X0D13 and MOSI to X0D22.

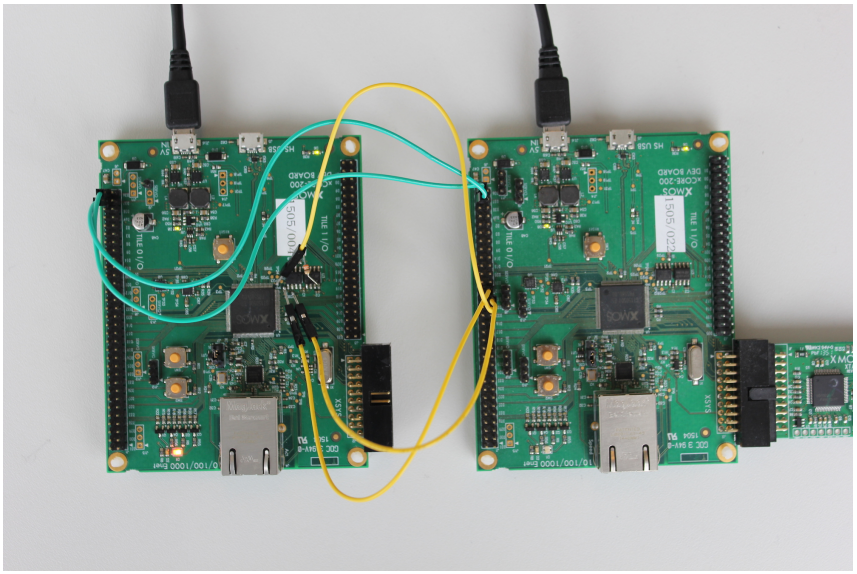

Figure 3: Connections between two eXplorerKIT boards set up as SPI slave and master

If you now run the programmer app, it will send the image and slave should boot and flash LEDs:

```
xrun --io bin/programmer.xe
```

3.6 XN file

No changes are required to the application's XN file.

You might want to remove references to SPI/QSPI flash to make it clear the program does not boot from flash. This is the Boot section from Node 0 and respective Device section from ExternalDevices.

3.7 USB

Loader includes some USB initialisation code that only applies to XU series parts. XL series parts can safely run the same code without any side effects.

4 Limitations

- Maximum combined image size 256KB
- XL216 and XU216 only
- Big PLL changes that require reboot are not supported
- Tools version 14.0 to 14.2.4
- No xflash features such as compression, firmware upgrade, multiple application images, or encryption

Please contact XMOS if you require support for older tools versions or encounter issues with future tools versions.