



AN02058: SPI slave boot

Publication Date: 2026/8/1
Document Number: XM-015648-AN v1.0.0

IN THIS DOCUMENT

1	Introduction	1
2	Installing the XTC SPI Boot Tools	1
3	SPI slave image file processing	3
4	Example application	7
5	Debugging	9
6	Further reading	11

1 Introduction

The XCORE devices may be booted with an image supplied by an external SPI-master device. This App Note provides the components required to process an XMOS `.xe` file containing an XCORE application and generate a SPI data stream.

In particular it provides:

- ▶ XTC SPI Boot Tools to convert a `.xe` file into a SPI slave image file
- ▶ Python scripts to extract data blocks from the SPI slave image file and transmit them to the XCORE (using an FTDI FT232H on a Windows host)
- ▶ Documentation showing how to install and use these tools
- ▶ Timing diagrams that show how a SPI master device must supply the image data blocks to the XCORE

1.1 Restrictions and limitations

- ▶ The example has been validated on the XK-EVK-XU316 board
- ▶ The example has been validated on Windows 11 Pro hosts using an FTDI FH232H SPI Master with SPI clock rate of 15 MHz
- ▶ Loading of LPDDR and SWMEM are not supported

2 Installing the XTC SPI Boot Tools

This section assumes that the [Xmos XTC 15.3.1 Tools](#) have been downloaded and installed. The required version is specified in the accompanying **README**.

Installation instructions can be found [here](#).

Special attention should be paid to the section on [Installation of Required Third-Party Tools](#).

2.1 Extracting the XTC SPI Boot Tools



Windows

Extract the contents of the file **tools\Microsoft_xboot_1_0_1__XTC_15_3_1.zip** provided in this App Note into the existing XTC Tools 15.3.1 installation directory tree. This zip file contains:

- ▶ a **bin** directory; the files in this must be extracted to the **bin** subdirectory of the XTC Tools 15.3.1 installation
- ▶ a **target** directory tree; the files in this must be extracted to the **target** subdirectory tree of the XTC Tools 15.3.1 installation

The term <AppNoteFullPath> is used below to refer to the full path of the location in which this App Note has been installed.

The extraction of the *XTC SPI Boot Tools* into XTC Tools 15.3.1 can be carried out with the tool 7Zip with the following commands:

```
cd "c:\Program Files\XMOS\XTC\15.3.1"
7z x "<AppNoteFullPath>\tools\Microsoft_xboot_1_0_1__XTC_15_3_1.zip"
```

Note

The XTC Tools 15.3.1 installation typically resides at **c:\Program Files\XMOS\XTC\15.3.1**. Administrator privileges are required to write files to this location.

macOS

Extract the contents of the file **tools/MacOSX_xboot_1_0_1__XTC_15_3_1.tgz** provided in this App Note into the existing XTC Tools 15.3.1 installation directory tree. This tgz file contains:

- ▶ a **bin** directory; the files in this must be extracted to the **bin** subdirectory of the XTC Tools 15.3.1 installation
- ▶ a **target** directory tree; the files in this must be extracted to the **target** subdirectory tree of the XTC Tools 15.3.1 installation

The term <AppNoteFullPath> is used below to refer to the full path of the location in which this App Note has been installed.

The extraction of the *XTC SPI Boot Tools* into XTC Tools 15.3.1 can be carried out with the tool *tar* with the following commands:

```
cd /Applications/XMOS_XTC_15_3_1
tar xzf "<AppNoteFullPath>/tools/MacOSX_xboot_1_0_1__XTC_15_3_1.tgz"
```

Run the commands:

```
chmod a+x /Applications/XMOS_XTC_15_3_1/bin/xboot
xattr -rd com.apple.quarantine /Applications/XMOS_XTC_15_3_1/bin/xboot
```

Linux

Extract the contents of the file **tools/Linux64_xboot_1_0_1__XTC_15_3_1.tgz** provided in this App Note into the existing XTC Tools 15.3.1 installation directory tree. This tgz file contains:

- ▶ a **bin** directory; the files in this must be extracted to the **bin** subdirectory of the XTC Tools 15.3.1 installation
- ▶ a **target** directory tree; the files in this must be extracted to the **target** subdirectory tree of the XTC Tools 15.3.1 installation



The term <AppNoteFullPath> is used below to refer to the full path of the location in which this App Note has been installed. The term <XTCFullPath> is used below to refer to the full path of the location in which the XTC Tools 15.3.1 have been installed.

The extraction of the *XTC SPI Boot Tools* into XTC Tools 15.3.1 can be carried out with the tool *tar* with the following commands:

```
cd <XTCFullPath>
tar xzf "<AppNoteFullPath>/tools/Linux64_xboot_1_0_1__XTC_15_3_1.tgz"
```

3 SPI slave image file processing

The tool **xboot** converts an XMOS **.xe** file to an XMOS proprietary SPI slave image file. This file must be processed by the customer's system (either off-line by a PC for example, or by the SPI master processor) to extract the data that must be sent to the xcore.

3.1 Data blocks

The following **data blocks** are sent over the SPI slave interface to boot the XCORE. The **Delay after block send** field defines the time-delay that must be inserted (with SPI Slave Select de-asserted) after the block has been sent.

These blocks and their delays are extracted from the SPI slave image file generated by the **xboot** tool.

Block description	Data block name	Delay after block send (ns)
Secondary loader	SEC- ONDARY_LOADER	SECONDARY_LOADER_POST_DELAY
Secondary header data	SEC- ONDARY_HEADER	SECONDARY_HEADER_POST_DELAY
Switch setup program	SWITCH_SETUP	SWITCH_SETUP_POST_DELAY
Tile 1 image	TILE_1_IMAGE	TILE_1_IMAGE_POST_DELAY
Tile 0 image	TILE_0_IMAGE	

The **Data block name** and **Delay after block send** fields are referred to in the tables below.

The SPI slave image file is a binary file and its format (schema) is defined in the next section. All values are little-endian. This image file must be processed to extract the data blocks and inter-block delays defined in the table above.

All delay values extracted from the SPI slave image file are in units of nanoseconds.

3.2 Data block extraction

The SPI slave image file comprises the following contiguous data sections, with the data blocks to be transmitted and the associated delays embedded in these data sections.

Secondary loader section

The first section of the file contains the secondary loader which is defined below. The size of this block is variable and the SPI image file must be read in two steps; the first step extracts size of the data, the second step reads the data and CRC.



SEC- ONDARY_LOADER field	Description	Size (bytes)
SEC- ONDARY_LOADER size	Size of the SECONDARY_LOADER data	4
SEC- ONDARY_LOADER data	The SECONDARY_LOADER data bytes	SECONDARY_LOADER size x 4
SEC- ONDARY_LOADER CRC	CRC32 value of SEC- ONDARY_LOADER data	4

The three fields above must be concatenated to form the **SECONDARY_LOADER** data block which is the first data block trasmitted over the SPI interface to the XCORE.

Header section

The next section of the file contains the **SECONDARY_HEADER** data block. This block must be read in two stages - the first read is required to obtain the value of **SEC-
ONDARY_HEADER size** and the second to read in the entire **SECONDARY_HEADER** data block based on the size obtained by the first read.

SECONDARY_HEADER field	Description	Size (bytes)
Reserved		16
Number of tiles	Number of tiles in image	4
Switch table offset	Offset in header to switch table	4
Tile table offset	Offset in header to first tile table	4
SECONDARY_HEADER size	Size of this entire header block	4
SEC- ONDARY_LOADER_POST_DELAY	See the Data blocks section above	4
SEC- ONDARY_HEADER_POST_DELAY	See the Data blocks section above	4
Boot delay		4
Reserved		4
Variable-size header data	Size based on number of tiles	SECONDARY_HEADER size - (12 x 4)

The **Variable-size header data** field of the **SECONDARY_HEADER** data block is structured as follows:



Variable-size data field	header	Description	Size (bytes)
Reserved			Switch table offset
SWITCH_SETUP_POST_DELAY		See the Data blocks section above	4
SWITCH_SETUP size		Size of SWITCH_SETUP which follows this section	4
Reserved			4
TILE_1_IMAGE_POST_DELAY		See the Data blocks section above	4
TILE_1_IMAGE size		Size of TILE_1_IMAGE which follows SWITCH_SETUP	4
Reserved			4
Reserved			4
Reserved			4
TILE_0_IMAGE_POST_DELAY		See the Data blocks section above	4
TILE_0_IMAGE size		Size of TILE_0_IMAGE which follows TILE_1_IMAGE	4
Reserved			4
Reserved			4

The next section of the file contains the **SWITCH_SETUP** data block (which is defined in the **Data blocks** section above). The size of this block (SWITCH_SETUP size) is provided in the above Variable-size header data.

SWITCH_SETUP field	Description	Size (bytes)
SWITCH_SETUP	The data block SWITCH_SETUP	SWITCH_SETUP size x 4

The next sections of the file contain the TILE data blocks (which are defined in the **Data blocks** section above).

The size of these blocks (TILE 1 IMAGE size and TILE 0 IMAGE size) is provided in the above Variable-size header data.

Field	Description	Size (bytes)
TILE 1 IMAGE	The data block TILE 1 IMAGE	TILE 1 IMAGE size x 4
TILE 0 IMAGE	The data block TILE 0 IMAGE	TILE 0 IMAGE size x 4

3.3 Data transmission

Bytes are sent *least significant bit* first. Each byte of the data blocks extracted from the SPI image file must be bit-reversed before being sent over the SPI interface to the XCORE.

The data must be transmitted over the SPI interface according to the timing diagrams below.

The diagram above shows the first data block send, where the secondary stage loader is sent to the XCORE. This includes detailed signal timing information.

The diagram below shows the full sequence of sending five data blocks, that will boot the application.



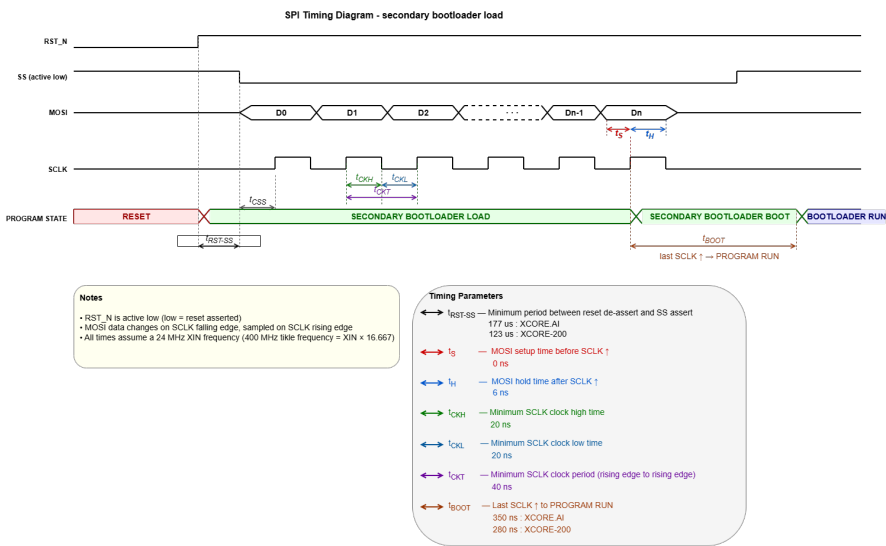


Fig. 1: Timing diagram - SPI signal timing when sending secondary loader

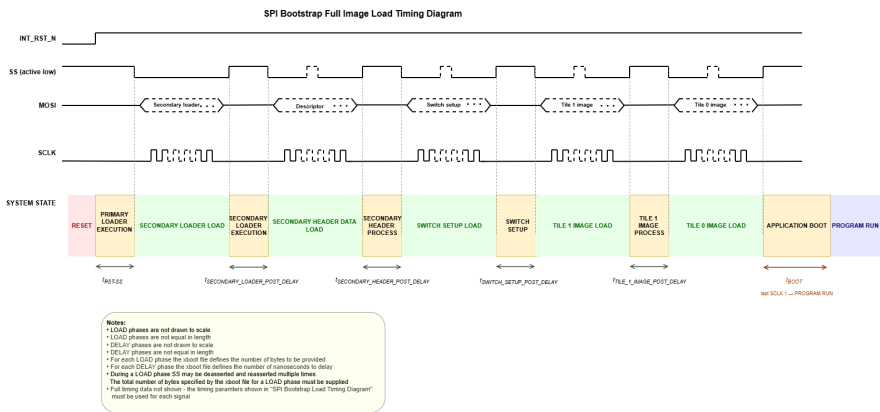


Fig. 2: Full SPI sequence to load application



The host SPI master may have a limit on the data that may be transferred in a single transaction (with SPI SS continuously asserted). It may break up a data transfer request by a software into a sequence of transactions, with SS de-asserted between these transactions.

The SECONDARY_LOADER data block **must** be transferred in a single transaction, with SPI SS asserted. This data block is small in size so the single transaction requirement can be met by hosts.

Subsequent data blocks may be broken into multiple transactions.

4 Example application

The example application can be built for the XCORE.AI device on the XK-EVK-XU316 board or for the XCORE-200 device on the XK-EVK-XU216 board

The example application is built using the [xcommon-cmake](#) build system, which is provided with the XTC tools and is based on [CMake](#).

The example application will flash the LEDs on the XK-EVK-XU316 board and toggle the IOs X1D_14, X1D_15, X1D_20, X1D_21 (tile 1 port 4C).

An example **host** application is provided to unpack the SPI image file, and send the SPI data blocks to the XCORE.

4.1 XTC Tools 15.3.1 Command Prompt

In the steps below, an XTC Tools 15.3.1 Command Prompt must be used.

Check the **xboot** tool is available by running:

```
xboot --version
```

If this tool is not available, go back to the section [Extracting the XTC SPI Boot Tools](#) to extract the XTC SPI Boot Tools into the XTC 15.3.1 installation.

4.2 Building the example

The **an02058** software ZIP package should be downloaded and extracted to a chosen working directory.

XCORE.AI

To configure the build, the following commands should be run from the XTC command prompt:

```
cd an02058
cd app_an02058
cmake -G "Unix Makefiles" -B build
```

XCORE-200

To configure the build, the following commands should be run from the XTC command prompt:

```
cd an02058
cd app_an02058
cmake -G "Unix Makefiles" -B build -DAPP_HW_TARGET=XCORE-200-EXPLORER
```

All required dependencies are included in the software package. If any dependencies are missing, they will be retrieved automatically during this step.

The application binaries should then be built using **xmake**:



```
xmake -j -C build
```

Binary artifacts (.xe files) will be generated under the appropriate subdirectories of the **app_an02058/bin** directory – one for each supported build configuration.

For subsequent builds, the **cmake** step may be omitted. If **CMakeLists.txt** or other build files are modified, **cmake** will be re-run automatically by **xmake** as needed.

4.3 Running the example

The example code can be loaded via an XTAG and run to ensure it functions. From an XTC command prompt, the following command should be run from the **an02058/app_an02058** directory:

```
xrun bin\app_an02058.xe
```

4.4 Generating a SPI image

From an XTC command prompt, the following command should be run from the **an02058/app_an02058** directory.

XCORE.AI

Run the **xboot** tool with the command:

```
xboot -o bin\app_an02058.xboot --target-file XK-EVK-XU316-xboot.xn bin\app_an02058.xe
```

This step creates the SPI slave image file **app_an02058.xboot**.

When using the **xboot** tool to generate an xboot image from an **.xe** program, the file **XK-EVK-XU316-xboot.xn** *must* be supplied using the **--target-file** option.

XCORE-200

Run the **xboot** tool with the command:

```
xboot -o bin\app_an02058.xboot --target-file XCORE-200-EXPLORER-xboot.xn bin\app_an02058.xe
```

This step creates the SPI slave image file **app_an02058.xboot**.

When using the **xboot** tool to generate an xboot image from an **.xe** program, the file **XCORE-200-EXPLORER-xboot.xn** *must* be supplied using the **--target-file** option.

4.5 Booting the XCORE device

Python example scripts are provided to boot the XCORE device, using an FTDI FT232H device in the adapter board [Adafruit FT232H Breakout](#) acting as a SPI master, connected to a Windows host.

A Command Prompt must be used to run the Python example scripts. Python 3.12 must be on the PATH of the Command Prompt. This Python installation must have the modules **usb** and **pyftdi**. These are typically installed by running the commands:

```
pip install usb
pip install pyftdi
```

Setting up the FTDI device

The FTDI FT232H must be setup to use the **libusb-win32 (v1.4.0.0)** driver. This is done using the **Zadig** 3rd party driver management tool.



1. Download the [Zadig](#) executable from the [Zadig](#) website
2. Run the downloaded [Zadig](#) executable
3. Check the menu option **Options -> List All Devices**
4. Scroll down the device list and select the **USB Serial Converter** device option
5. In the **Driver** field select the **libusb-win32 (v1.4.0.0)** option

The above process must be repeated when the FTDI device is plugged into a different USB Type A socket.

Booting the XCORE device

Carry out the following steps to boot the XCORE using its SPI slave interface:

1. Open a Command Prompt which has Python 3.12 on its PATH
2. Change directory to **an02058/app_an02058**
3. Run the command:

```
python ..\tools\spi_master\send_from_ftdi.py bin\app_an02058.xboot
```

XCORE.AI example test system

The XK-EVK-XU316 and the [Adafruit FT232H Breakout](#) should be connected as shown in the schematic below.

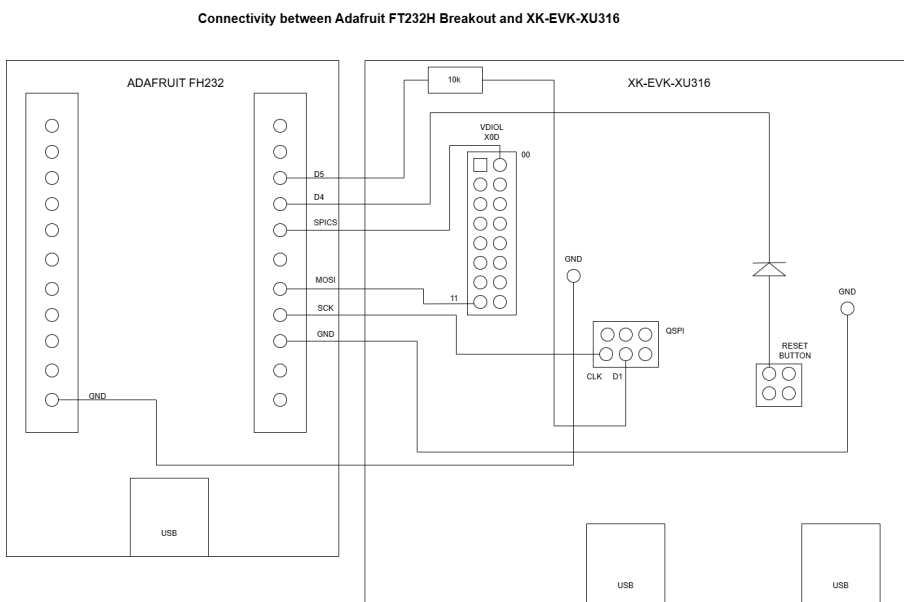


Fig. 3: Test system connectivity

The full SPI sequence is shown in the diagram below.

5 Debugging

If the device fails to boot the tool **xrun** may be used to inspect the XCORE:





Fig. 4: Full SPI sequence

```
xrun --dumpstate
```

If the PC of tile 0 contains an address with the upper three nibbles set (**0xfff**....) then it is in the ROM's primary loader.

This will occur if the SPI slave bootmode (binay value 010) is not set on the signals X0D06, X0D05 and X0D04 when **RST_N** is de-asserted.

It will also occur if the SECONDARY_LOADER data block has been rejected by the primary loader due to a CRC check error, or because SS was deasserted during the transmission of this block.

If the PC of tile 0 contains an address of the form (**0x000f**...) then it is running the secondary loader and a subsequent data block may have been rejected due to a CRC error check, an interblock delay is too short or insufficient data has been transmitted.



6 Further reading

- ▶ XMOS XTC Tools Installation Guide
<https://www.xmos.com/xtc-install-guide>
- ▶ XMOS XTC Tools User Guide
<https://www.xmos.com/view/Tools-15-Documentation>
- ▶ XMOS application build and dependency management system; *xcommon-cmake*
<https://www.xmos.com/file/xcommon-cmake-documentation/?version=latest>
- ▶ IO timings for XCORE.AI
<https://www.xmos.com/documentation/XM-014231-AN/html/rst/index.html>



Copyright © 2026, All Rights Reserved.

XMOS Ltd. is the owner or licensee of this design, code, or Information (collectively, the "Information") and is providing it to you "AS IS" with no warranty of any kind, express or implied and shall have no liability in relation to its use. XMOS Ltd makes no representation that the Information, or any particular implementation thereof, is or will be free from any claims of infringement and again, shall have no liability in relation to any such claims.

XMOS, XCORE, VocalFusion and the XMOS logo are registered trademarks of XMOS Ltd. in the United Kingdom and other countries and may not be used without written permission. Company and product names mentioned in this document are the trademarks or registered trademarks of their respective owners.

