USB Bootloader Description and Standards

Version 1.0

The USB interface used to connect XS1-L devices to the XMOS toolchain and referenced in this document is under current development and subject to improvements and changes—the latest version is described in the XMOS USB Device Layer Library Guide.

Publication Date: 2010/07/22

1 Introduction

This document explains how the XMOS toolchain can boot an XS1-L1 over USB. It is used, for example, by the debugger (xgdb) to debug systems that are built out of one or more XCores. The document explains how the bootloader works, the port layout, the schematics, and the protocol to use the bootloader over USB.

This document does not discuss the USB *Device Firmware Upgrade (DFU)* protocol, which is discussed in a separate document (*USB Firmware Upgrade on XMOS* available soon). *Firmware upgrade* enables any USB device to upgrade firmware stored in flash memory using a standard USB-protocol. *USB boot* enables a specific USB device to boot over USB without using flash.

This document specifies a set of the standards that must be followed for a design to be compatible with the XMOS toolchain. These standards are:

USB-BOOT-1: USB enumeration The USB descriptors to use when enumerating as a bootloader.

USB-BOOT-2: Firmware upload over USB protocol The protocol to use over the two USB endpoints.

USB-BOOT-3: Hardware design The schematics to use when designing a system that uses a USB bootloader.

USB-BOOT-4: USB serial number assignment The meaning of serial numbers.

USB-BOOT-5: USB Serial number storage How the serial number is stored.

The use of the XMOS USB Vendor ID (VID) and the allocated Product Identifier (PID) are allowed provided that these standards are followed.

Figure 1 shows a block diagram of a device that supports bootloading over USB. It comprises a USB-PHY and an XS1-L1 device that is both controlling the PHY and being booted over the PHY.

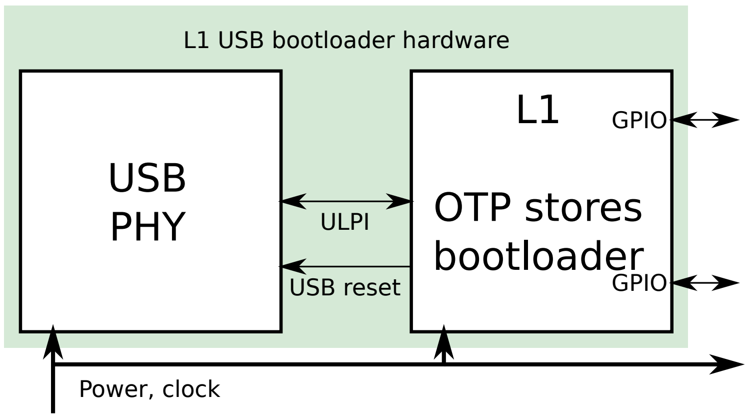


Figure 1: Block diagram of booting over USB

The bootloader works as follows:

1. The L1 boots from OTP and runs code that enumerates on the USB bus and presents itself as a device that accepts programs to execute.
2. xgdb (or other program) scans the USB bus and tries to find devices that are a bootloader (determined by the VID, PID and version number of the device), and that have a serial number that indicates that the device is compatible. For example, xgdb requires a serial number that indicates that the L1 has hardware that is compatible with the debugger.
3. xgdb (or other program) loads the new firmware into the device. When a device receives new firmware, bootcode copies its serial number to the 16 bytes at address 0x1B000 to 0x1B00F prior to executing the new code.
4. On boot, the new firmware picks up the serial number to use from address 0x1B000, and then enumerates as a device with that serial number.

1.1 PID and VID

USB uses a PID and VID to indicate the product and vendor. These must be identical for the bootloader and new firmware if one wants to avoid Microsoft Windows requiring multiple driver installations. The values of PID and VID are specified in Standard USB-BOOT-1 below.

1.2 Version numbers

The version number is used to differentiate between bootloader (major version 0) and subsequent firmware (major version not equal to 0) if identical PID, VID and serial numbers are used.

1.3 Serial numbers

The serial number must uniquely identify the hardware. The same serial number must be used for both the bootloader and the new firmware (again in order to avoid windows requiring multiple drivers). The serial number is used to decide what capabilities this device has, and must adhere to the specification in Standard USB-BOOT-4 below.

A serial number can be programmed in by loading the programming firmware into the device (using the bootloader).

2 Standard USB-BOOT-1: USB Enumeration

The descriptor should contain a device class, subclass and protocol that are all equal to 0xff, and have one configuration only. The device qualifier should indicate a max packet size of 64 bytes.

The single interface should have two endpoints, interfaceclass, subclass and protocol should all be 0xff. For each endpoint the max packet size should be set to 512, and the interval should be set to 1.

Any bootloadable USB device must enumerate using the following details:

- Use VID (Vendor ID) 0x20B1.
- Use PID (Product ID) 0xF7D1.
- Enumerate with major version number 0

- Enumerate with a serial number of 16 digits where the first character indicates the class; the serial numbers are defined in Section 5 - Standard USB-BOOT-4.

  The serial number must be read out from OTP; first the word on address 2040 must be read, if it is '0' then serial number "XXXXXXXXXXXXXXXX" (16 Xs) shall be used. If word 2040 does not equal 0, then the entire serial number should be read out from word 2040 using differential mode. The least significant byte of the word at address 2040 is the first byte of the serial number. The most significant byte of the word at address 2046 is the sixteenth byte of the serial number.

USB bootloader firmware must be programmed into either flash or OTP. The bootloader must relocate itself to address 0x1B000 or higher prior to execution, and execute from that address. If stored in OTP, the bootloader must be no larger than 8160 bytes, including the relocation code and any error correcting code, leaving 32 bytes empty for the serial number.

3 Standard USB-BOOT-2: Firmware upload over USB protocol

The bootloader must reside at at address 0x1B000 or above in memory, allowing programs of up to 44K of memory to be loaded into the device. The protocol of the USB bootloader requires two endpoints (in addition to endpoint 0) that are used as described below. Communication is synchronous: for every OUT transaction on Endpoint 0x01, the host must issue an IN transaction on Endpoint 0x82 to verify that the operation has completed.

3.1 Out Endpoint 1 (0x01)

Commands are received on this endpoint. A command comprises at most 512 bytes, and consists of a single word command, and up to 508 bytes of payload.

LOADER_CMD_WRITE_MEM — 1 Carries an address (bytes 0..3), a length (bytes 4..7), and *length* bytes of data. The length must be a multiple of 4. After writing the data, the USB loader will send back a LOADER_CMD_WRITE_MEM_ACK, see section 3.2. No writes should be requested to addresses 0x1B000 - 0x1FFFF inclusive.

LOADER_CMD_JUMP — 5 Carries an address (bytes 0..3) only; it must have a payload of exactly 4 bytes. The USB loader will send back a LOADER_CMD_JUMP_ACK (see section 3.2) and then jump to the specified address prior to shutting down all resources.

3.2 IN Endpoint 2 (0x82)

On this endpoint the firmware responds to commands. Packets are up to 12 bytes long, where the first word contains the response; there are up to 8 bytes of payload. The last 4 bytes of payload indicate whether another command can be issued: (0) means that another command can be issued, (-1) indicates that no other commands can be issued.

LOADER_CMD_WRITE_MEM_ACK — 2 Has a payload of 4 bytes, indicating the success state only (0).

LOADER_CMD_JUMP_ACK — 6 Has a payload of 4 bytes containing -1 indicating that the device will detach itself from the bus. On receiving this response, the host should wait for at least one millisecond and then issue a USB-reset.

4 Standard USB-BOOT-3: Hardware design

4.1 Clock frequencies

The L1 must run at 400MHz derived from a 13 MHz Crystal.

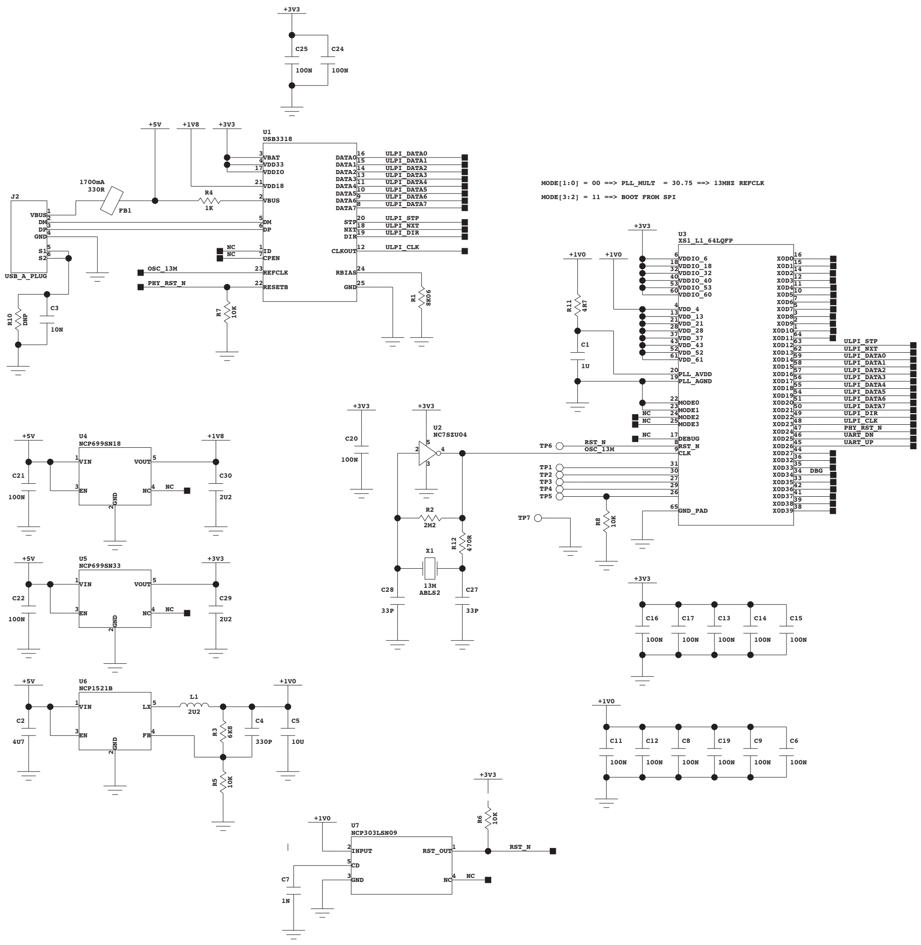
4.2 Port map

The L1 must use the following portmap. All pins labeled ULPI should be connected to the ULPI USB-PHY. Ports M and N should be declared as input ports (they must be tristated).

Pin	Port			Signal
	1b	4b	8b	
XD12	P1E0			ULPI_STP
XD13	P1F0			ULPI_NXT
XD14		P4C0	P8B0	ULPI_DATA[0:7]
XD15		P4C1	P8B1	
XD16		P4D0	P8B2	
XD17		P4D1	P8B3	
XD18		P4D2	P8B4	
XD19		P4D3	P8B5	
XD20		P4C2	P8B6	
XD21		P4C3	P8B7	
XD22	P1G0			ULPI_DIR
XD23	P1H0			ULPI_CLK
XD24	P1I0			ULPI_RST_N
XD35	P1L0			Declare as input
XD36	P1M0			Declare as input

Some ports are used internally when the ULPI is in operation—see the XS1-L Hardware Design Checklist for further information.

Developers are strongly encouraged to use the design in Figure 2 verbatim.



USB3318
XS1_L1_64LQFP
MODE[1:0] = 00 ==> PLL_MULT = 30.75 ==> 13MHZ REFCLK
MODE[3:2] = 11 ==> BOOT FROM SPI
NCP699SN18
NCP699SN33
NCP1521B
NC7SZU04
NCP303LSN09
USB_A_PLUG

Figure 2: Reference design of an L1 for USB boot loading. [TODO: delete components that are not USB related, ie, XLINK, UART, and JTAG]

5 Standard USB-BOOT-4: USB serial number assignment

The USB serial number indicates the type of device and its capabilities. Serial numbers are interpreted as follows:

- Identifiers starting with ’X’, ’x’, ’Y’, ’y’, ’Z’ and ’z’ are reserved by XMOS and shall not be used by any device not developed by XMOS.
- ’D’ and ’d’ are used to indicate that this hardware is compatible with the debugger. Serial numbers of this class are defined in a companion document: “*USB debugger description and standards*”.
- A serial number of all ’X’ is used to indicate that this device does not have an identifier programmed. They can be programmed according to Section 6 - Standard USB-BOOT-5.
- A serial number starting with ’R’ or ’r’ can be used freely.
- All other serial numbers are reserved for future device classes.

6 Standard USB-BOOT-5: USB serial number storage

The serial number is stored in differential mode in the top 32 bytes of the OTP. Blank serial numbers appear as a sequence of 0 and -1 words; any device with a blank serial number shall enumerate with “XXXXXXXXXXXXXXXX”.

Words 2040 to 2047 of the OTP should be programmed as follows:

- Word 2040 should contain the first 4 characters of the serial number—byte 0 should be stored in the least significant byte of the word, byte 3 should be stored in the most significant byte of the word.
- Word 2041 must be a copy of word 2040.
- Word 2042 should contain characters 4-7 of the serial number
- Word 2043 must be a copy of word 2042.
- Word 2044 should contain characters 8-11 of the serial number
- Word 2045 must be a copy of word 2044.
- Word 2046 should contain characters 12-15 of the serial number
- Word 2047 must be a copy of word 2046.

Each word is programmed twice to provide redundancy when reading out the serial number.

Document History

Date	Release	Comment
2010-07-22	1.0	First release

Disclaimer