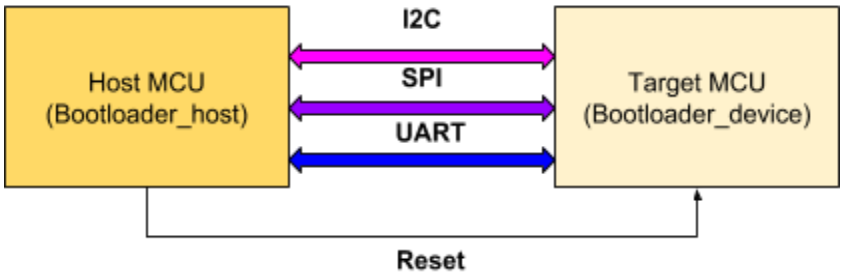
Introduction

This document describes the generic bootloader host system. It can be easily ported to any microcontroller having one of these hardware interfaces which are UART, I2C, and SPI. You can even modify it to work as Firmware Over the Air update by modifying the transport layer of the bootloader host system. The host system will reside on a microcontroller which will update the target microcontroller.

Figure 1: Host and Target Connection Scheme



Assumptions:

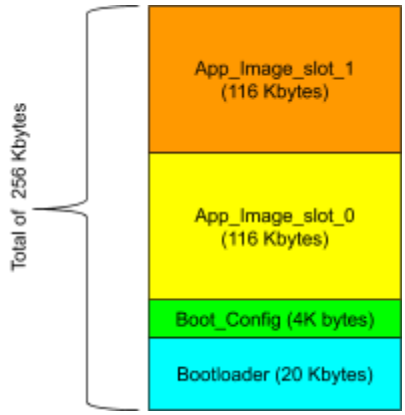
1. Host can receive new firmware image with any of these medium.
    a. A SD card attached to it having new firmware.
    b. Interfaced With PC to get new firmware and can act as protocol translator for a hardware combination of host and target.
2. Power pins are omitted for simplicity.
    a. Host can provide the power to the target or host and device can be powered separately with common ground path between them.
3. The target microcontroller is having 256 KBytes of Flash available. The whole flash has been divided into four parts which are as follows:
    a. Bootloader:
        i. Bootloader target code, which is responsible for
            1. Receiving new firmware.
            2. Booting from particular image slot based on Boot_Config Information.
            3. Checking the firmware integrity.
            4. Roll back to previous image if firmware checksum fails.

    b. Boot_Config: It stores the following information:
        i. Active image information
            1. Active Image 0/1.
            2. Firmware version and checksum of both slots.
            3. Bootloader and protocol version.

4. Timestamp of the last image update.
5. Firmware update counter.

c. App_Image_Slot_0:
   i. Application firmware slot 0.
d. App_Image_Slot_1:
   i. Application firmware slot 1.

Figure 2: Flash Layout



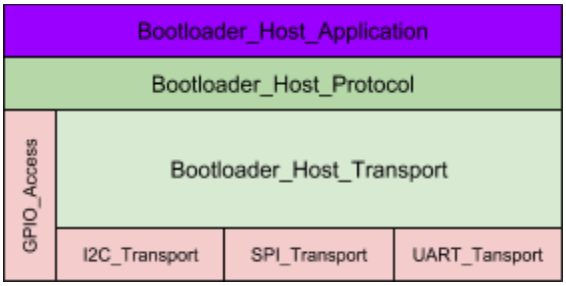
Bootloader host side firmware components:

- **Bootloader_Host_Application:** Hardware independent Application layer which inherits the host protocol layer in order to package the data frame and commands.

- **Bootloader_Host_Protocol:** This layer implements the actual bootloader command set and the firmware validation functionality. This layer can be modified and changed to support new bootloader features without affecting the transport and application layer.

- **Bootloader_Host_Transport:** This layer abstracts the hardware dependency of the whole stack, it consists of generic functions to read and write over any communication protocol such as I2C, SPI and UART.

Fig 3: Bootloader Host Software Stack



Bootloader command set:

The supported commands are listed in Table 1. Each command is described in this section.

Table 1: Bootloader Commands

Command	Command Code	Command Description
0x00	Get Config	Gets the boot configuration data
0x01	Set Config	Sets the boot configuration data
0x02	Erase	Erases fixed no of bytes from the selected address
0x04	Write Memory	Writes up to 256 bytes of memory starting from an address specified by the application.
0x08	Read Memory	Reads up to 256 bytes of memory starting from an address specified by the application.
0x10	Go	Jumps to user application code located in the internal Flash memory.

Communication Safety(optional):

All communication from the programming tool (PC) to the device is verified by:

- checksum: received blocks of data bytes are XOR-ed. A byte containing the computed XOR of all previous bytes is added to the end of each communication (checksum byte). By XOR-ing all received bytes, data plus checksum, the result at the end of the packet must be 0x00.
- For each command the host sends a byte and its complement (XOR = 0x00).
- Each packet is either accepted (ACK answer) or discarded (NACK answer):
    - ACK = 0xAA
    - NACK = 0x55