# **Flashing Protocol**

Abstract

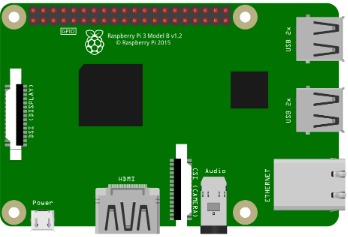
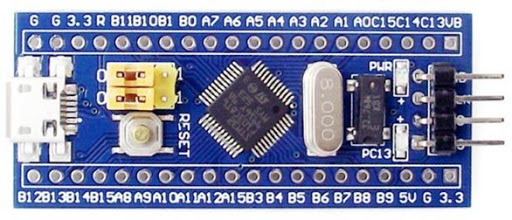
This Document describes the communication process between raspberry pi and the bootloader of the microcontroller during flashing new application process. This document presents communication protocol used and physical Layer description, Frames description and Flashing process.

Table of contents

1.0 Communication protocol:

The main communication protocol used between raspberry pi and the microcontroller is UART (Universal Asynchronous Transmitter Receiver) with baud Rate 115200 Fps. Additional external GPIO pin is used to send a triggering signal from the raspberry pi to the microcontroller to start the flashing process.

* 1. **Physical Layer diagram**



Interrupt GPIO Pin

UART Pins

Common ground

* 1. **Pins Description** 
     1. **Pins Connection Table**

|  |  |  |
| --- | --- | --- |
| Raspberry Pi Pin | MicroController Pin | Connection description |
| Pin #5 (GND Pin) | GND Pin | Common ground |
| Pin#8 (TXD) | Pin A10 (Rx-Pin) | Raspberrypi → microcontroller UART pin |
| Pin#10 (RXD) | Pin A9 (Tx-Pin) | microcontroller → Raspberrypi UART pin |
| Pin#15 (GPIO 22) | Pin A0 (GPIO pin) | Start Flashing signal pin |

* + 1. **Pins Configuration**

|  |  |
| --- | --- |
| Raspberry Pi Pin | Configuration |
| Pin #5 (GND Pin) | No Configuration needed. |
| Pin#8 (TXD) | No Configuration needed. |
| Pin#10 (RXD) | No Configuration needed. |
| Pin#15 (GPIO 22) | Output GPIO pin. |

|  |  |
| --- | --- |
| STM32f103 Pin | Configuration |
| GND Pin | No Configuration needed. |
| Pin A10 (Rx-Pin) | * Activate GPIO –A Clk. * Input Pull-up Resistor. * Input Speed. |
| Pin A9 (Tx-Pin) | * Activate GPIO –A Clk. * Alternative Function Push/pull. * O/P suitable speed. |
| Pin A0 (GPIO pin) | Configured as EXTI\_0 interrupt source pin. |

1. Frames

The communication between Raspberry Pi and target microcontroller consists of many frames each frame has its own functionality, for the target microcontroller it takes different actions like receive frames from raspberry pi and send response according to the frame sent to it, and these Frames are:

1- Flash new app.

2- Flash write sector.

3- Respond frame.

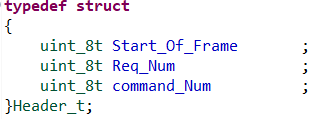
4- End flashing.

**Note:** Each frame consists of its own header used to describe it

* 1. **Frame Header**
     1. **Description**

Each frame consists of header which is struct type consist of three members to describe the frame.

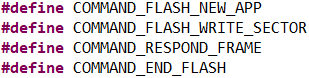
* + 1. **Contents**



1. **Start\_Of\_Frame:** it’s just to notify the micro controller that   
    the communication will start and the raspberry pi   
    want to send frames, the microcontroller checks   
    this element with defined macro.



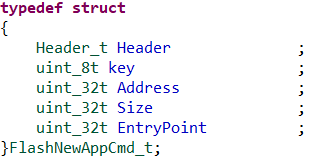
1. **Req\_Num:** it’s like counter increments for each frame sent and it’s   
    used for debugging, if frame sent wrong and we need to   
    know which frame has the problem we can check this   
    element.
2. **Command\_NUM:** as we said before the micro controller take   
    specific action for each frame so this element   
    used to acknowledge the micro controller which   
    frame sent, and according to the macro the micro   
    controller will take action, these macros are:



* 1. **Flash new Application Command Frame**
     1. **Description**

This frame is sent at the beginning of the flashing process. It contains main information about the flashed application.

* + 1. **Contents**



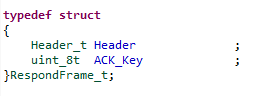
1. **Header:** Described previously.
2. **Key:** this key used just for verify the flow of frames that if the   
    raspberry pi sent write sector frame, first the micro   
    controller must check that if the raspberry pi sent flash new   
    app frame and the micro controller responded with   
    (RECEVIED\_OK) and the value of this key is macro :



1. **Address:** raspberry pi request to send the file in specific address   
    in micro controller flash and here it checks if it’s valid   
    or not.
2. **Size:**  this element contains the total size of the file which sent by   
    raspberry Pi.
3. **EntryPoint:** this element contains the address by which when the   
    microcontroller finished writing the application   
    successfully it will jump to this address to execute the   
    application .
   1. **Flash write Sector Frame**
   2. **Respond Frame**
      1. **Description**

The respond frame is sent from the microcontroller to the raspberry pi as acknowledge response for each frame is sent from the pi. The respond frame contains ACK\_KEY that indicates whether the frame - that was sent previously from the raspberry pi- was sent probably and made its functionality successfully or not.

* + 1. **Contents**



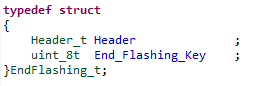
1. **Header:** Described previously.
2. **ACK\_Key:** Indicates the state of the previously sent command  
    whether it made its functionality probably or not.

Capture

* 1. **End Flashing Frame**
     1. **Description**

The last frame sent from the raspberry pi to the microcontroller. The Frame will be expected to be received from the raspberry pi by the microcontroller and it will check the End\_Flashing\_key. If it was received successfully, the communication between the raspberry pi and the microcontroller will be ended probably. After receiving the End flashing frame the microcontroller will change the Flash Marker and writes the Application Entry Point in its specific position in the Flash.

* + 1. **Contents**



1. **Header:** Described previously.
2. **End\_Flashing\_key:** This key will be checked by the microcontroller  
    to confirm the ending communication between  
    raspberry pi and the microcontroller.

end

1. Flashing process

The flashing process is occurred between raspberry pi and microcontroller by exchanging the previously described frames. Each frame is sent from the raspberry pi to the microcontroller is to control the state of the flashing process or to control the data flow.

* 1. **Flashing Process timeline Diagram**

Microcontroller

Raspberry-pi

The raspberry pi starts the flashing process by sending this frame with some initial information.

FlashNewAppCmd\_t

The microcontroller receives the flash new app frame, checks header, takes initial information then reply with respond frame.

RespondFrame\_t

FlashWriteSector\_t

The raspberry pi begins to send data block, size of the block, starting address of the block in this frame. The next block will not be sent until receiving respond frame.

The microcontroller gets the data block, checks the header, erases the page, flashes it in the desired address, verifies the flashing process is done correct then reply with respond frame .

RespondFrame\_t

FlashWriteSector\_t

RespondFrame\_t

EndFlashing\_t

The raspberry pi sends this frame to end the communication with the microcontroller.

The microcontroller checks the end flashing key, verifies it, changes the marker value, and stores the entry point, reply and reset.

RespondFrame\_t

* 1. **Flashing process stages description**

The flashing process can be divided into three main stages. Each stage related to one type of frames sent to the microcontroller by the raspberry pi. The stages are:

1. Flash new application command.
2. Flash data sectors (blocks).
3. End Flashing process.
   * 1. **First Stage: Flash new Application command**

This is the first stage of the flashing process. This stage starts by the raspberry pi sending FlashNewAppCmd\_t frame. The microcontroller starts with the bootloader that checks the marker (NO\_APP at the begining).

The microcontroller will checks the frame header, erase the marker, takes the total application size, flash key that indicates that a flash new app command was sent before the flash write sector frame, total application size, first location address to be flashed at the flash – for checking availability of the application region-. Then, replies with RespondFrame\_t with ACK\_Key equals RECEIVED\_OK.

* + 1. **Second Stage: Flashing data sectors (blocks)**

The Second stage will start after receiving RespondFrame\_t of the FlashNewAppCmd\_t frame. The raspberry pi begins to send data blocks to the microcontroller in FlashWriteSector\_t frame. The frame contains fixed size of data bytes (MAX\_DATA\_SIZE). The size of actual data from the fixed size data is determined by the size attribute.

The microcontroller erase the page then flashes data block based on the size attribute. After flashing it, it verifies the flashed block by comparing the written data in the Flash with the data received from the raspberry pi.

After verification, the microcontroller minus the data block from the total application size (received at the first stage) and replies with RespondFrame\_t with ACK\_Key equals RECEIVED\_OK. When the application saved size reaches zero, the application will know that all data application has been flashed and EndFlashing\_t frame is expected to be received from the raspberry pi.

* + 1. **Third Stage: End Flashing process**

The last stage is to end the communication between microcontroller and raspberry pi. The raspberry pi sends EndFlashing\_t frame, the microcontroller verifies the end flashing key, writes the marker and application entry point, and replies with RespondFrame\_t with ACK\_Key equals RECEIVED\_OK.

Finally it makes a software reset to start the bootloader. The bootloader checks the marker value at the beginning then jumps to the application entry point.