Exploring STM32 System Bootloader: Part 2

Introduction

In the second part of our series on STM32 system bootloader, we delve into the communication sequence between the host and the microcontroller for firmware updates. We analyse the essential commands involved, such as baud rate synchronization, memory reading, chip identification, extended erase, and host write commands. Understanding these protocols is crucial for effectively utilizing the system bootloader in STM32 microcontrollers.

Bootloader Identification and Versions:

Firstly, lets talk about identification. The bootloader identification version of the STM32 device bootloader is coded on one byte in the 0xXY format. Here, X specifies the embedded serial peripheral(s) used by the device bootloader, and Y specifies the bootloader version.

STM32 series	Device	Supported serial peripherals	Bootloader ID		Bootloader (protocol) version
			ID	Memory location	
F4	STM32F401xD(E)	USART1/USART2/ DFU (USB device FS)/ I2C1/I2C2/I2C3/ SPI1/SPI2/ SPI3	0xD1	0x1FFF76DE	USART (V3.1) DFU (V2.2) SPI(V1.1) I2C (V1.1)
	STM32F410xx	USART1/USART2/ I2C1/I2C2/I2C4 SPI1/SPI2	0xB1	0x1FFF76DE	USART (V3.1) I2C (V1.2) SPI (V1.1)

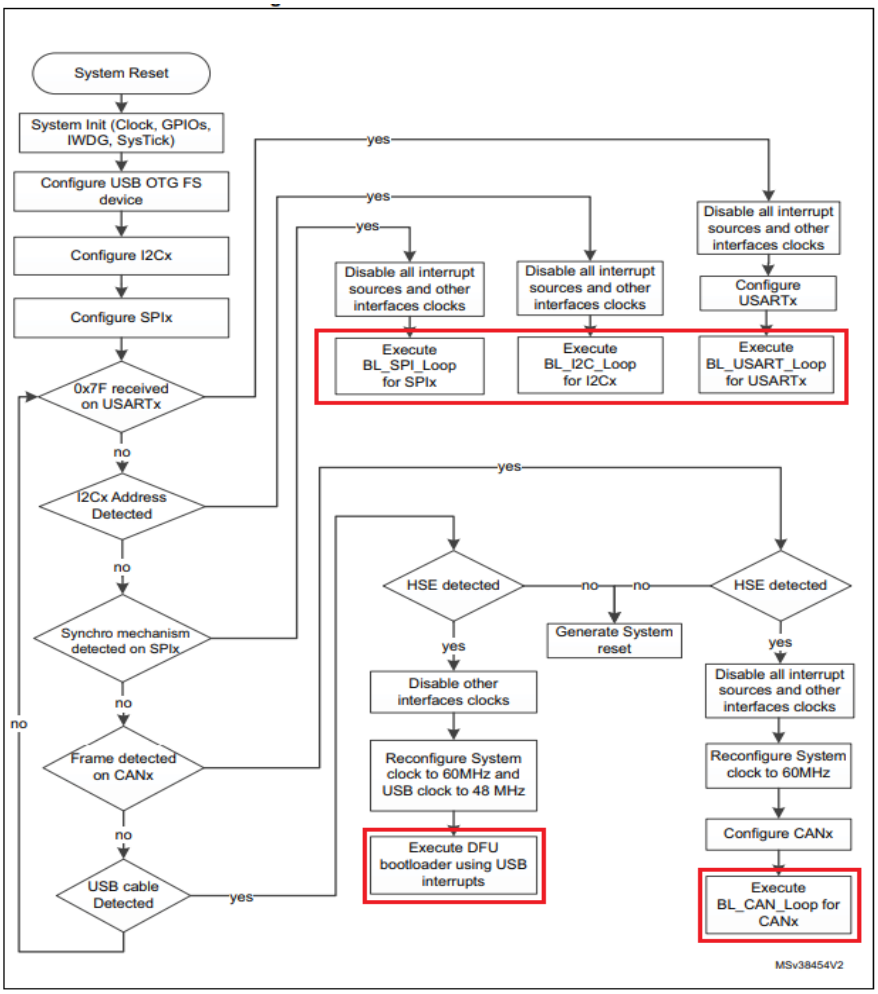
– **X** specifies the embedded serial peripheral(s) used by the device bootloader:
X = 1: one USART is used
X = 2: two USARTs are used
X = 3: USART, CAN and DFU are used
X = 4: USART and DFU are used
X = 5: USART and I²C are used
X = 6: I²C is used
X = 7: USART, CAN, DFU and I²C are used
X = 8: I²C and SPI are used
X = 9: USART, CAN (or FDCAN), DFU, I²C and SPI are used
X = 10: USART, DFU and I²C are used
X = 11: USART, I²C and SPI are used
X = 12: USART and SPI are used
X = 13: USART, DFU, I²C and SPI are used
X = 14: USART, DFU, I2C, I3C, FDCAN and SPI are used

Bootloader ID, Peripheral's compatible, Version

System Bootloaders is designed in BareMetal programming with one of the loop gets executed **(BL_USART_Loop**, BL_I2C_Loop, BL_SPI_Loop,

Written By
Narendra Chowdary Chundru

BL_CAN_Loop, DFU USB). The bootloader approach to firmware updates offers a standardized and streamlined method, irrespective of the communication protocol employed. This approach ensures consistency and compatibility throughout the update process, regardless of whether the firmware update is carried out using protocols such as USART, I2C, SPI, CAN, or DFU USB.



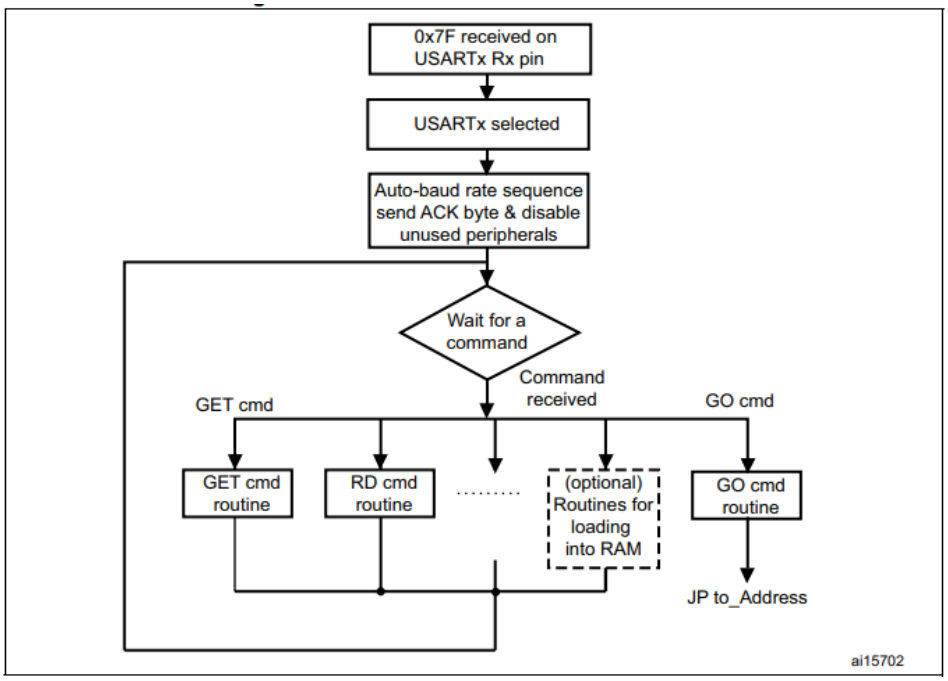
Flow Diagram of System Bootloaders

Lets explore on how STM32CubeProgrammer(Host) is talking to STM32F401x(D)E(Device) through USART by analysing the AN3155 STM Application Note.

Written By
Narendra Chowdary Chundru

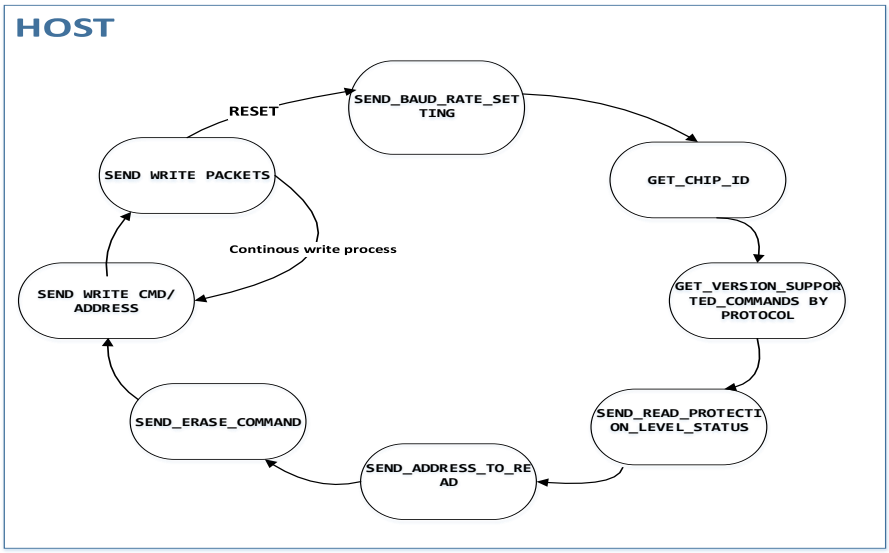
Command[1]	Code	Description
Get[2]	0x00	Gets the version and the allowed commands supported by the current version of the protocol.
Get Version[2]	0x01	Gets the protocol version.
Get ID[2]	0x02	Gets the chip ID.
Read Memory[3]	0x11	Reads up to 256 bytes of memory starting from an address specified by the application.
Go[3]	0x21	Jumps to user application code located in the internal flash memory or in the SRAM.
Write Memory[3]	0x31	Writes up to 256 bytes to the RAM or flash memory starting from an address specified by the application.
Erase[3][4]	0x43	Erases from one to all the flash memory pages.
Extended Erase[3][4]	0x44	Erases from one to all the flash memory pages using two-byte addressing mode (available only for USART bootloader v3.0 and higher).
Special	0x50	Generic command that allows to add new features depending on the product constraints, without adding a new command for every feature.
Extended Special	0x51	Generic command that allows the user to send more data compared to the Special command.
Write Protect	0x63	Enables the write protection for some sectors.
Write Unprotect	0x73	Disables the write protection for all flash memory sectors.
Readout Protect	0x82	Enables the read protection.
Readout Unprotect[2]	0x92	Disables the read protection.
Get Checksum	0xA1	Computes a CRC value on a given memory area with a size multiple of 4 bytes.

UART Commands



Bootloader for STM32 with USART

Written By
Narendra Chowdary Chundru



Host Communication

Let's discuss the communication sequence between the host (PC) and the microcontroller (MCU) device. As part of my reverse engineering efforts, I captured the data packets exchanged between the host and the MCU using a logic analyzer. I have attached the analyzer files below for your analysis once you complete reading this article.

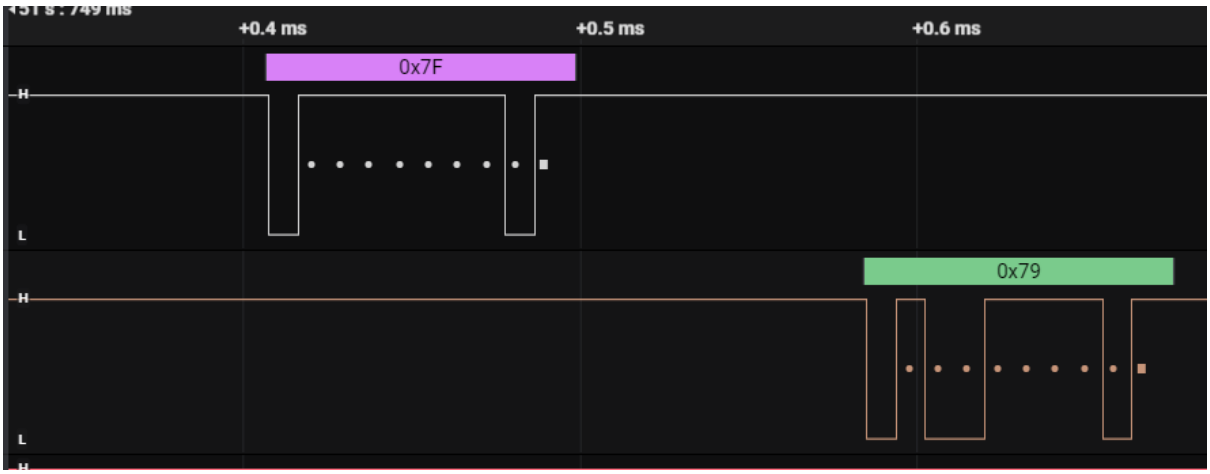
BL_USART_LOOP:

Synchronization of Host and Device Baud rate:

When the device enters the system bootloader based on its configuration, the MCU begins scanning the RX line for a specific data frame. This data frame consists of 8 bits with even parity and one stop bit (8E1) and has a value of 0x7F. The baud rate detection can be performed using hardware (auto baud rate) or software methods.

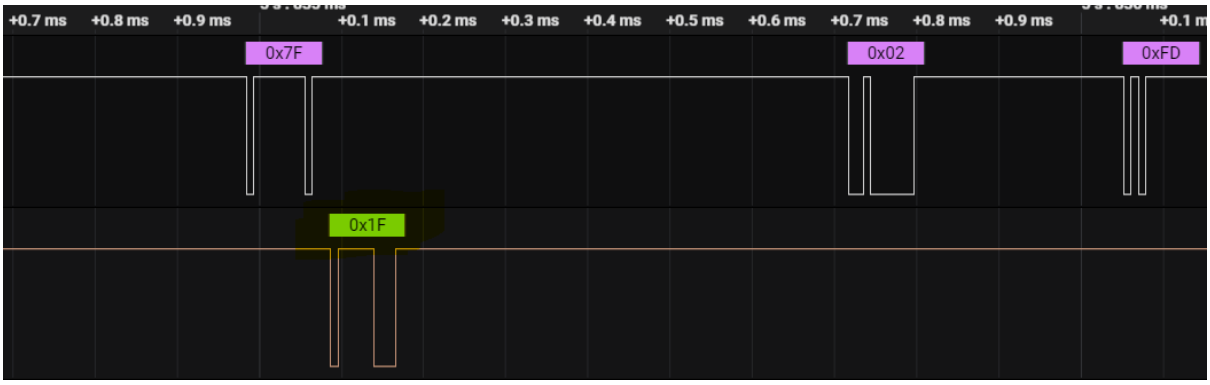
To determine the baud rate, the duration of the received data frame is measured using the Systick timer. The count value of the timer is then used to calculate the corresponding baud rate factor relative to the current system clock. Once the baud rate is calculated, the code initializes

the serial interface with the determined baud rate. An acknowledge byte (0x79) is sent back to the host, indicating that the STM32 device is ready to receive commands.



Baud Rate Command(0x7F) and its ACK Response

During baud rate setting, the host receives an ACK from the device, indicating successful synchronization. This ACK is considered only during the initial communication. After baud rate setting, the device sends a NACK to confirm the configuration is complete. This NACK indicates a healthy communication and allows further commands without explicit acknowledgments.

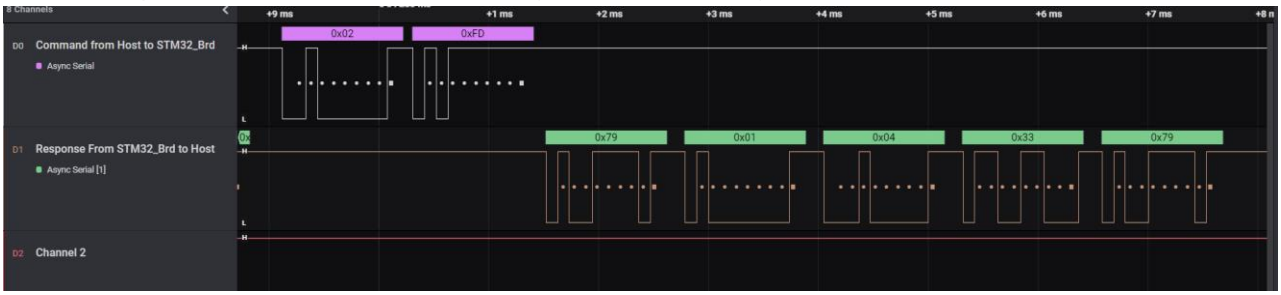


Baud Rate Command(0x7F) and its NACK Response

Chip ID(0x02) command

Written By
Narendra Chowdary Chundru

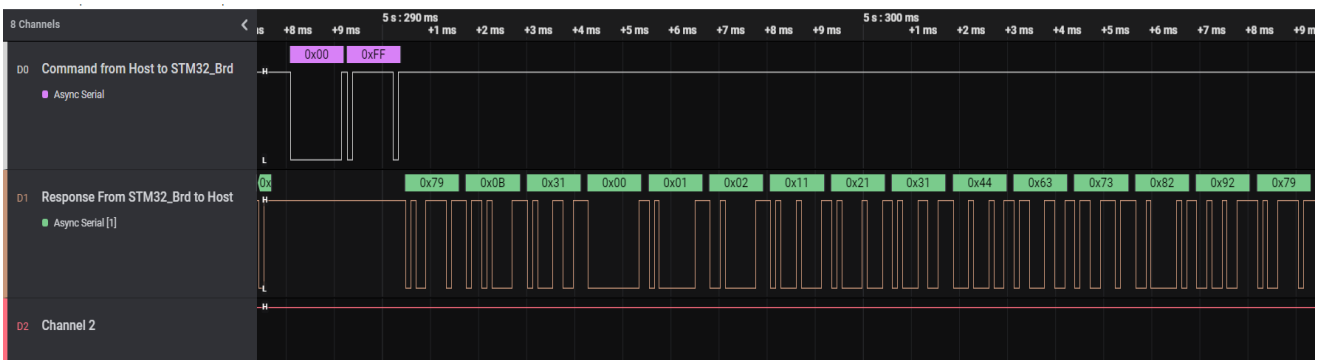
The host issues a command (0x02) to request the device's chip ID. The device responds with an acknowledgment (ACK) and provides the chip ID, along with the number of bytes and the product ID



Get ID Command(0x02) and its Response

Command Request(0x00)

The host sends a command (0x00) to the device, requesting information about the protocol version, the number of supported commands by the respective protocol. The device responds with an acknowledgment (ACK) and provides the requested information.



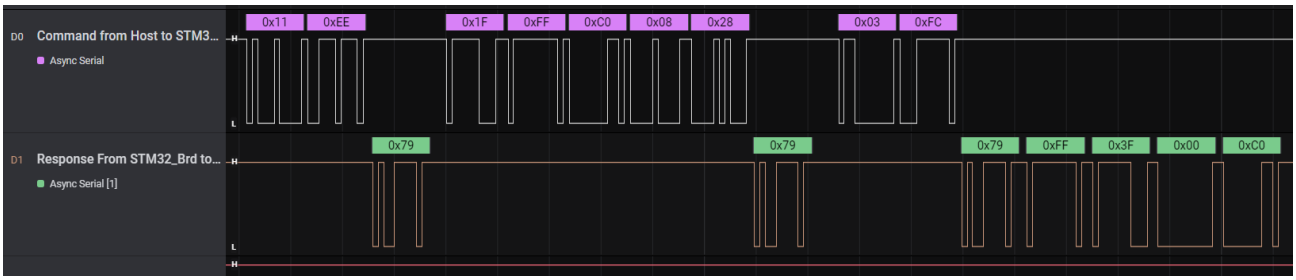
Get Command(0x00) and its Response

Read Memory (0x11) Command :

To read memory on the device, the host sends a read command (0x11) along with a checksum. If the read protection (RDP) is not active, the device acknowledges the command. Next, the host sends the memory

Written By
Narendra Chowdary Chundru

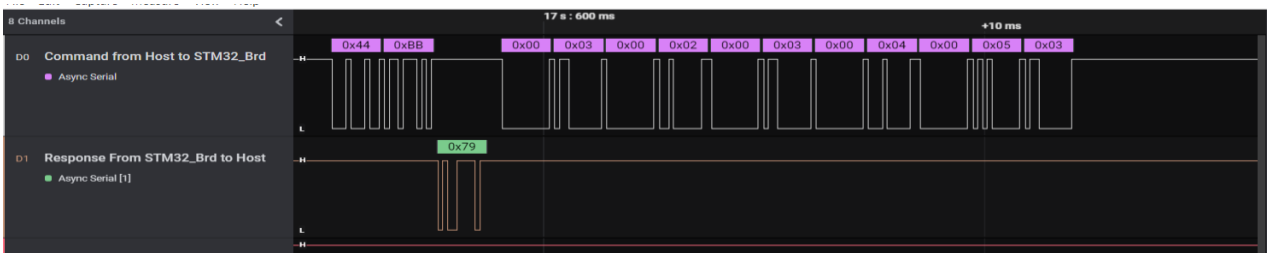
address along with a checksum. If the address and checksum are valid, the device acknowledges the request. The host then specifies the number of bytes to be read, and the device responds with an acknowledgment and sends the requested bytes from memory.



Get Read Memory (0x11) and its Response

Extended Erase (0x44) command

To perform an extended erase operation, the host sends an erase command (0x44) along with a checksum. The device acknowledges the command, and the host proceeds to send the number of pages to be erased (2 bytes) and the specific memory pages to be erased with their corresponding checksum. The device verifies the packets and erases the specified pages.

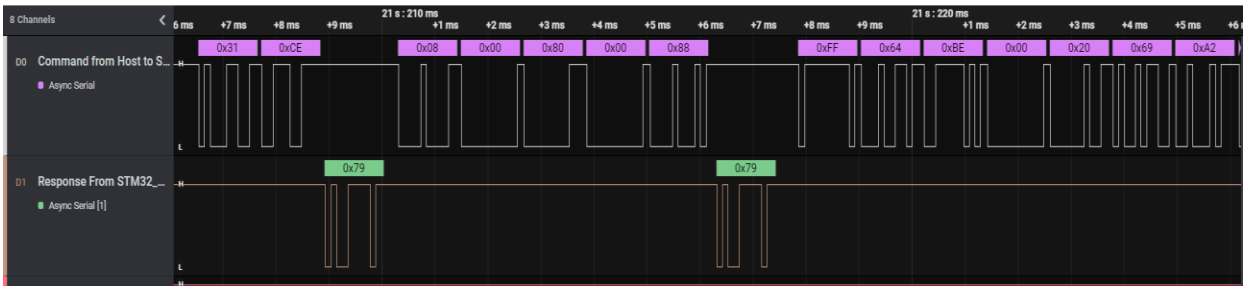


Extended Erase(0x44) Command and its Response

Host Write (0x31) command

For writing data into memory, the host sends a write command (0x31) along with a checksum. The device responds with an acknowledgment

(ACK), and the host proceeds to send the write memory address along with a checksum. The device acknowledges the address, and the host sends a single data packet containing 128 or 256 bytes. The device writes the received data into memory and sends an acknowledgment. The host continues the write process until it completes the full write operation.



Write Command(0x31) and its Response

Conclusion

In conclusion, we have discussed about communication sequence between HOST and device, The STM32 system bootloader is primarily used during the development phase for programming and updating firmware. However, in the end product, the In-Application Programming (IAP) bootloader is used for secure and efficient firmware updates.

References

AN2606 - STM32 microcontroller system memory boot mode
AN3155 - USART protocol used in the STM32 bootloaderRM0368 Reference manual
STM32F401xB/C and STM32F401xD/E advanced Arm®-based 32-bit MCUs

Logic Analyser's Application download link for viewing the packets

After_connecting.sal

Connection_establishment.sal

Written By
Narendra Chowdary Chundru