The Inner Workings of the Micro 3D Printer

By donovan6000

For Imee,
Thanks for having such an inspiring name
— donovan6000

CONTENTS

FOREWORD

Introduction

A while ago, it was brought to my attention that very few, if any, people understand the Micro 3D printer's software, hardware, and firmware as well as I do. This stems from the fact that a lot of the information regarding such details is carelessly scattered over a variety of places making it difficult to actually extract the valid pieces of information that one is searching for. As a result, some of the capabilities and limitations of the Micro 3D printer are quite misunderstood despite my best efforts to clarify such discrepancies. Over the last few years, as I've been developing software compatible with this type of printer, I've also noticed how little my own software is understood by the general public which has resulted in many not utilizing their printers to their fullest potential. These concerns have raised the need for some kind of documentation that details my complete work regarding the Micro 3D printer. So as I set forth and begin writing it, I want to explicitly point out that I intend for this document to appeal not just to those curious about the inner workings of the Micro 3D printer but also to those developing software the printer. Hence I'll try to be as detailed as possible to answer as many questions as I can.

I'd also like to mention, as some of you have already noticed, that this entire document is purely based on my own opinions and beliefs that I've developed over the course of my time reverse engineering the Micro 3D printer. I'm not going to put anything even remotely close to an 'About the Author' section in this document mainly because I feel like those who take the time to boast arrogantly about their mundane, often unrelated, accomplishments when appealing to others are blinded by their own hubris and rarely ever prove their credibility. "True knowledge is knowing that you know nothing", so I don't intend on trying to pretend like I know more that I know. And the few of you who do know a little bit about me would probably find it humorous for someone in my line of work to gloat about being creditable in this field. So feel free to take everything your about to read with a grain of sand if you see fit.

Acknowledgments

Although no one but myself currently knows that I’m writing this, I’d still like to thanks those who’ve contributed to the progress made in reverse engineering the Micro 3D printer up to this point. Even though the Micro 3D printer scene looks pretty dead nowadays, a few years ago it was booming with those eager to utilized and contribute to alternative software methods that would allow them to take full advantage of their underwhelming printers. It’s all thanks to to that passionate community that I was able to complete such ambitious projects. So, first off, I’d like to thank Bankyf72 for selling their printer on eBay the day they received it from the Kickstarter campaign. I would have never gotten involved with the Micro 3D printer if it weren't for them. I’d also like to thank those who worked on reverse engineering the Micro 3D printer before I got started developing for it. Specifically Loswave and tomasf whose work proved to me that there was a large part of the community who desired more from their printers. Huge thanks to ra100, MrBeta, TonyH, Muele, and drschlaumeier for everything they’ve done to assist in furthering the progress of my software, Nocturnal for mapping out where most of the printer’s microcontroller’s pins go, and mikeayles for taking some high resolution X-ray images of the printer’s motherboard. Also a huge thanks to anyone else who tested and contributed to M33 Linux, M33 Fio, and iMe. The impressive progress that’s been made wouldn’t have happened if it weren’t for all of those involved. You all have my undeniable gratitude.

PREFACE

What this document covers

This document is divided into 3 parts. The first will discuss the Micro 3D printers hardware, the second will discuss the printer's bootloader, and the third will discuss the printer's firmware.

The sections regarding the printer's hardware give an overview about everything that the printer physically consists of. This section will include how the printer' peripherals, like the heater, fan, motors, etc., work on a software level, and it will include where all the printer's microcontroller's pins go to.

The printer's bootloader section will detail all of the commands and responses that the bootloader can receive and send. This will include details about the printer's boot sequence, how updating the printer's firmware is accomplished, how to manage the printer's EEPROM, how to verify the validity of the printer's firmware, and how the bootloader can be leveraged to obtain the firmware decryption lookup table.

The section about the printer's firmware will discuss all the commands and responses that both M3D LLC's firmware and iMe firmware can receive and send. The differences between the those two firmwares will be explained.

The appendix contained at the end of this document will have one section. Appendix A will contain the addresses, length, type, and format of all the data stored in the printer's EEPROM.

Who this document is for

This text is intended to be read by anyone who wants to know more about how the Micro 3D printer works on both a software and hardware level. Having a general understanding about 3D printers will help when it comes to understanding some of the more technical pieces of informations,

and it would benefit you to have some experience with a programming language and to understand how data is stored so that any portions of this document that contain code don't go over your head.

Conventions

The only textual conventions of this document that are worth mentioning are some of the code conventions. All code will be in a monospace font, like `int timer = 0`, all tokens consisting of compound words or phrases that aren't representing constant values will be stylized in camel case, like `colorOfThePrinter`, any constant values used in code will be written in all capitalized characters with underscores separating compound words or phrases, like `NUMBER_OF_SAMPLES`, the end of a statement will be signified by a newline, code will be indented to indicate what scope it's included in, all code will be presented in a pseudocode language similar to C. Data conventions in this document expect that a byte is 8 bits, a word is 2 bytes, a double word is 4 bytes, and a quadword is 8 bytes. A float refers to the iEEE 754 single-precision floating point number, and a double refers to the iEEE 754 double-precision floating point number.

Errata

Although I've tried my best to properly format and phrase the content in this document and provide information that I believe is accurate, I'm sure that there will be mistakes. So I'd like to apologize in advance if any of these overlooked errors hinder your enjoyment while reading this. I'd also be extremely grateful to anyone who takes the time to inform me of or fix any of these errors. This file is being distributed on the GitHub repository for the Micro 3D printer firmware iMe in Portable Document Format (PDF) and OpenDocument Text Document Format (ODT) to try to make it as accessible as possible, so I look forward to receiving pull requests from you guys letting me know of my blunders. I can also be contacted via email at donovan6000@exploitkings.com. And if you think there's something that you could expand on further or add to this text, then by all means do it. I'm quite lenient when it comes to merging pull requests, and I'm very supportive when it comes to helping those who want to share their knowledge. So don't by shy or nervous about contacting me.

HARDWARE

This section covers

- Blah
- Blah
- gBlah

In the beginning…

B lah

BOOTLOADER

This section covers…

- How the printer's boot sequence works
- All the commands and responses that the printer's bootloader can receive and send
- How to manage the printer's EEPROM
- How updating the printer's firmware is accomplished
- How to obtain the firmware decryption lookup table

Booting up

The bootloader section reset vector bit, `BOOTRST`, of the second fuse byte, `FUSEBYTE2`, in the ATxmega32C4 microcontroller determines if the reset vector loads the first address of the bootloader section, `BOOT_SECTION_START`, or the first address of the application section, `APP_SECTION_START`, into the microcontroller's program counter when the microcontroller is powered on or reset. This determines if the microcontroller starts off by running the code in the bootloader section of flash or the application section of flash when it starts up or gets reset, and the Micro 3D printer has that fuse's bit set to 0 which results in the former. So the bootloader is the first stage in the printers boot sequence, and the term 'bootloader mode' will be used from now on to refer to when the printer is running code in the bootloader section of flash. We'll also use the term 'firmware mode' to refer to when the printer is running code in the application section of flash.

Let me clarify how the ATxmega32C4 microcontroller flash is partitioned before going any further. In total the microcontroller has 36KB of flash. The first 32KB of which are called the application section and the last 4KB of is called the bootloader section. Those sections contain the firmware and bootloader respectively. The last 4KB of the application section is called the application table section, and it can be used to stored data that's avaliable for use by the other sections. If it's not used in that manner, like in the case of the Micro 3D printer, then it just contains application sections code. The accessibility for each of those 3 sections is controller by the microcontroller's lockbits, `NVM_LOCKBITS`, that determine when those sections can be read and written to. The locks bits can only be set to a more strict value by the microcontroller and can only be reset when all of the contents of the microcontroller are erased by an external programmer, so they provide a good way to prevent access to certain parts of flash.

Most Micro 3D printer have their lockbits set to 0xBC, but I've seen several with them set to 0x3C. The former prevents writing to the bootloader section and allows reading from the bootloader section, application section, and application table section. The later has the same restrictions except doesn't

allow reading from the bootloader section. They also both prevent reading and writing any of the microcontroller's contents using an external programmer.

I'll also briefly mention the ATxmega32C4 microcontroller's fuse bytes just to get it out of the way. The microcontrller has 4 fuse bytes: FUSEBYTE1, FUSEBYTE2, FUSEBYTE4, and FUSEBYTE5. They configure things like the watchdog timeout period and brownout detection voltage. These bytes are set to 0x00, 0xBE, 0xE3, and 0xE9 for the Micro 3D printer, and the only things worth mentioning about them are that the reset vector jumps to the bootloader section of flash, the pin that triggers an external reset is disabled, and that the EEPROM is erased when an external programmer erases the microcontroller's contents.

That's all the topics that need to be covered for now, so lets get back to talking about the boot sequence. While in bootloader mode, the printer acts as a USB Communication Device Class (CDC) device accessible via the USB port on the back of the printer. Its USB descriptor's vendor ID (VID) is 0x03EB and product ID (PID) is 0x2404. Also, the bootloader's USB descriptor lacks a serial number entry.

While the printer is in bootloader mode, the microcontroller is capable of erasing the application section section of the microcontroller's flash, return the the CRC32 of the bootloader section of flash, return the CRC32 of the application section of flash, return the bootloader's version, write bytes to the application section of flash, return the first 0x300 bytes of the microcontroller's EEPROM, write to the EEPROM, and jump to the first address in the application section of flash which puts the printer into firmware mode. The printer can't process or run G-code commands while in bootloader mode, so it's only really useful for performing operations that verify, update, and launch the printer's firmware while in this mode. Switching the printer into firmware mode is the last step in the boot sequence, and once that happens the code in the bootloader section of flash is no longer being run. Before switching the printer into firmware mode, the bootloader disables all interrupts and disconnects from the USB bus.

Commands

There's only 9 commands that can be used to perform operations while the printer is in bootloader mode. However only the first 8 commands listed are implemented in retail units of the Micro 3D printer. It's believed that the last command is only available in bootloaders that M3D LLC uses during development.

• Launch firmware

Sending the byte 0x51 (Q) to the bootloader switches the printer into firmware mode. The printer does not send any response in reply to this command. As mentioned before, the bootloader disables all interrupts and disconnects from the USB bus before switching into firmware mode.

• Get CRC32

Sending the byte 0x43 (C) followed by a 0x42 (B) byte caused the printer to generate and respond with the 4 byte CRC32 of the microcontroller's bootloader section in little endian. If the byte 0x41 (A) is sent instead of the 0x42 (B) byte, then the printer will generate and response with a 4 byte CRC32 of the microcontroller's appication section in little endian.

• Set address

Sending the byte 0x41 (A) followed by a 2 byte address in little endian sets the destination offset for page command that's about the be sent. The printer responds with a 1 byte carriage return (0x0D) when the operation has completed successfully. This is only ever used to set the address to 0x0000 right before updating the printers firmware.

• Erase application sections

Sending the byte 0x45 (E) erases the application section of flash. When bytes of flash are erased they are set to 0xFF as a result, so the application section will contain 32KB of 0xFF bytes after this is performed. The printer responds with a 1 byte carriage return (0x0D) when the operation has completed successfully.

• Write page

Sending the byte 0x42 (B) followed by a 2 byte page size multiplied by 2 in little endian followed by that many encrypted bytes will cause the printer to decrypt and write those bytes to the page at the previously set address. After the write has been completed, the address will be incremented by that may bytes to allow the next write page command to write to the following page. The printer responds with a 1 byte carriage return (0x0D) when the operation has completed successfully.

• Get bootloader version

Sending the byte 0x4D (M) will cause the printer to respond with the printers bootloader version in ASCII. The response can vary between 2 and 4 bytes in length and always starts with a 'B' character. The known bootloader versions respond with the following: B001, B004, B5. Any bytes sent with the 0x4D byte are ignored, so sending an 'M115' command to the printer can assist I determining if the printer is in bootloader mode or firmware mode by checking if the first byte in the response is a 'B' or not.

• Read EEPROM

Sending the byte 0x53 (S) will cause the printer to respond with the first 0x300 bytes of the microcontroller's EEPROM followed by a 1 byte carriage return (0x0D) when the operation has completed successfully. The microcontroller's EEPROM is actually 0x400 bytes, but there's no way to read or write to the last 0x100 bytes while in bootloader mode.

• Write EEPROM

Sending the byte 0x55 (U) followed by a 2 byte address in little endian followed by 2 bytes representing the number of bytes being written in little endian followed by that number of bytes will cause the printer to write those bytes to the microcontroller's EEPROM starting at the specified address. It will respond with a 1 byte carriage return (0x0D) when the operation has completed successfully.

• Read firmware

Sending the byte 0x52 (R) will cause the printer to response with the contents of its application section of flash. This command is only available in bootloaders used by M3D LLC during development, so it's not known if it responds with a 1 byte carriage return (0x0D) when the operation has completed successfully.

Errors

If the bootloader isn't able to understand your request or encounters an error with processing your request, then it will respond with one of the following. Not much is known about what triggers some of these errors, but it's common to receive an 'e1' response when you send an unknown command to the bootloader.

• e0

This response means that the firmware parser returned a parse error.

• e1

This response means that the firmware parser returned a not supported protocol error.

• e2

This response means that the firmware parser returned an ASCII buffer overflow error.

• e3

This response means that the firmware parser returned a number expected error.

• e4

This response means that the firmware parser returned a number overload error.

• e5

This response means that the firmware parser returned an unknown error.

- **e6**

This response means that the firmware interpreter returned an unsupported error.

- **e7**

This response means that the firmware interpreter returned an unknown error.

Managing EEPROM

Writing to the microcontroller's EEPROM while in bootloader mode can be a bit confusing, so I thought that I'd include some example code of how it's performed. This example just demonstrates a function that can be used to write a value to a specified address, and it doesn't provide any error checking for it sending data to the printer fails.

```
bool writeToEeprom(uint16_t address, uint8_t data[], uint16_t length) {

	// Send command, address, and number of bytes
	send('U')
	send((address >> 8) & 0xFF)
	send(address & 0xFF)
	send((length >> 8) & 0xFF)
	send(length 0xFF)

	// Send bytes
	for(uint16_t i = 0; i < length; ++i)
		send(data[i])

	// Return if response indicated successful
	return receive() == "\r"
}

// Write 4 bytes containing 0x12 0x34 0x56 0x78 to address 0x2F0
writeToEeprom(0x2F0, “\x12\x34\x56\x78”, 4)
```

Updating the firmware

Updating the printer's firmware is also a complicated process, so it seemed necessary to include example code on how to do that. So the following code demonstrates how to send the commands required to update the printer's firmware. Like the previous example code, this one doesn't provide any error checking for it sending data to the printer fails either.

```
bool updateFirmware(uint8_t flashPageSize, uint8_t data[], uint16_t length,
uint32_t firmwareVersion, uint32_t firmwareCrc32, uint8_t paddingByte) {

	// Check if length isn't an even number of bytes
	if(length % 2 != 0)

		// Append padding byte to data
		data += paddingByte
```

```
      ++length

// Erase application section
send('E')

// Return false if erasing application section failed
if(receive() != "\r")
      return false

// Set address to 0x0000
send('A')
send('\x00')
send('\x00')

// Return false if setting address failed
if(receive() != "\r")
      return false

// Set pages to write
uint16_t pagesToWrite = length / 2 / flashPageSize
if(length / 2 % flashPageSize != 0)
      ++pagesToWrite

// Go through all pages to write
for(uint16_t i = 0; i < pagesToWrite; ++i)

      // Send page write command
      send('B')
      send((flashPageSize * 2 >> 8) & 0xFF)
      send((flashPageSize * 2) & 0xFF)

      // Go through all data in the page
      for(uint16_t j = 0; j < flashPageSize * 2; ++j)

            // Check if data to be written exists
            uint32_t position = j + flashPageSize * i * 2
            if(position < length)

                  // Check if padding is required
                  if(position % 2 == 0 && position == length - 1)

                        // Sending padding byte
                        send(paddingByte)

                  // Otherwise
                  else

                        // Send value
                        send(firmware[position + (position % 2 ? -1 : 1)])

            // Otherwise
            else

                  // Sending padding byte
                  send(paddingByte)

      // Return false if writing page failed
```

```
            if(receive() != "\r")
                  return false

      // Set address
      send('A')
      send('\x00')
      send('\x00')

      // Return false if setting address failed
      if(receive() != "\r")
            return false

      // Update firmware version in EEPROM
      eepromWriteInt(VERSION_OFFSET, VERSION_LENGTH, firmwareVersion)

      // Update firmware CRC in EEPROM
      eepromWriteInt(CRC_OFFSET, CRC_LENGTH, firmwareCrc32)

      // Request flash CRC32
      send("CA")

      // Get response
      uint32_t flashCrc32 = toBigEndian(receive())

      // Return false if firmware wasn't installed correctly
      if(flashCrc32 != firmwareCrc32)
            return false

      // Return true
      return true
}

// Update the printer’s firmware
updateFirmware(PAGE_SIZE, FIRMWARE_DATA, FIRMWARE_LENGTH, FIRMWARE_VERSION,
FIRMWARE_CRC32, PADDING_BYTE)
```

For this example to work the ATxmega32C4 microcontroller’s flash page size, which is 0x80 bytes long, and the padding byte for the Micro 3D printer, which is 0x23, would need to be known, and the format of the of the firmware, `FIRMWARE_DATA`, also needs to be in M3D LLC’s encrypted firmware format.

When M3D LLC distributes a firmware update for the Micro 3D printer, they swap every even byte in the firmware with the following odd byte while providing 0xFF bytes to the last encrypt the firmware’s contents, and include a CRC32 of the decrypted firmware with it. The firmware update mechanism that the bootloader includes will only work when

FIRMWARE

This section covers

- Blah
- Blah
- Blah

In the beginning...

B lah

APPENDIX

Appendix A: EEPROM map

B lah