**EZProm DataStream**

**File Format Specification**

**Usage: *convert(ihexname, lines, base)***

**Kyle Jiang**

**March 2018**

Background and Introduction

EZProm is a collection of hardware hacks and software libraries that allow quick, inexpensive programming and examining of 8-bit EEPROMs. Reading from the EEPROM is enabled with the main EZProm library’s *dispBytes* function, which takes in an address size, a start address, and a counter of bytes to read and displays the result on a serial terminal.

To write to the EEPROM is a slightly more difficult task. It is not efficient to manually drive the clock and data according to the bytes of a machine code. Most assemblers and low-level compilers, which I assume is what you are using to produce programs for the EEPROM, generally produce an object file known as Intel HEX (referred to as iHEX in this text). It is a rather compact form which includes an address, the number of bytes to be written starting at that address, the raw bytes, and a checksum. For instance, the following is from my BB85 I2C communications library for the Intel 8085:

:1000000031FF8301F07F3E02C5F5CDBE70CD337266

:10001000CD13717600000000000000000000000019

:10700000E5F5210700397E3DC20770F1E1C9E5F5DC

:107010002100807EF60477D301F1E1C9E5F5210076

:10702000807EE6FB77D301F1E1C9E5F52100807EA2

:10703000F60877D301F1E1C9E5F52100807EE6F796

:1070400077D301F1E1C9F5DB001F1F1FDA5370F19F

:10705000373FC9F137C9F5DB001F1F1F1FDA647006

:10706000F1373FC9F137C9E5F52100807EE61F778A

:10707000F1E1C9E5F5CD1C70CD38702100807EE6C8

The formatting and coloring were done by Notepad++. Each line begins with a colon, followed by the number of data bytes (in this case, each line contains 16 bytes, so a 0x10 follows the colon). Then, in blue, is the two-byte address, followed by a field type (red, 00 = data), then the data itself. In green at the very end is a checksum, which is simply computed as the complement of the sum of all previous bytes, mod 0x100 (256 decimal). For instance, examine line 2. The sum of the (nonzero) bytes is

0x10 + 0x10 + 0xCD + 0x13 + 0x71 + 0x76 = 0x1E7

which is just 0xE7 mod 0x100. The complement is found simply by inverting all the bits, producing 0x18, then adding 1, giving the checksum 0x19, which is correct.

iHEX is a very versatile format and is commonly used in dedicated chip programmers. Unfortunately, it is not easily processed by EZProm. The reasons for this are obvious – if not all address lines are connected to the EEPROM, writing the entire iHEX file will result in collisions in the data. Consider the above example. It appears that we write 32 bytes of data starting at 0x0000, and many more starting at 0x7000. But if our EZProm is configured to only have 10 address lines (A0 – A9), we will overwrite the data that was supposed to be at 0x0000 with the beginning of the data at 0x7000 (assuming A10 – A15 are tied low). To address this issue, a new filetype specific to EZProm is here introduced.

EZProm DataStream (EZDS) File Description

The DataStream file has extension *.ezds*. The EZProm main library is currently only able to write to the EEPROM from this form (excluding user-created functions using lower-level library subroutines, which is highly not recommended). EZDS can be generated by conversion from iHEX and direct assembly / compilation. For general use, the first method is the suggested one, mainly because of its library support; the second one may be better for those making their own assemblers / compilers or those who are hand-assembling their programs (and the program is relatively short).

The iHEX 🡪 EZDS conversion algorithm is implemented as a standalone Python file. The algorithm essentially decomposes iHEX to a *stream* of bytes, with *jumps* interspersed throughout. The algorithm ignores all data which lies outside the *programmable segment* determined by the EZProm setup; if no data in a line are in the programmable range, the checksum at the end of the line is *not* checked, otherwise it is.

Notice that the *programmable segment* consists of two parameters: (1) the *range* of addresses, which is 2^N, where N is the number of active lines connected to the memory, and (2) the *base address*, which is some nonnegative integer multiple of 2^N. The program checks that N is less than or equal to 16 (the entire address space), that the base is indeed a multiple of 2^N, and that 0 <= base <= 2^16 – 2^N.

Each line begins with a colon, and the first two bytes (four characters) are address. Each newline signifies an address jump (a skip of at least one byte from the proceeding byte).

Example

Below is an example assembly listing that multiplies two bytes in memory. We wish to write it to our memory using an EZProm setup, but we only have access to five address lines (32-byte segments). We approach this problem by producing multiple EZDS files, each with a different base address. The first one will have a base address of 0x00, the second 0x20 (32), the third 0x40 (64), and so on until the whole program memory is spanned.

0000 21 00 FF LXI H, 0FF00H

0003 31 FA FE LXI SP, 0FEFAH

0006 3E 00 MVI A, 00H

0008 21 FE FE LXI H, 0FEFEH

000B 77 MOV M, A

000C 23 INX H

000D 77 MOV M, A

000E 3E 00 MVI A, 00H

0010 21 FC FE LXI H, 0FEFCH

0013 77 MOV M, A

0014 23 INX H

0015 77 MOV M, A

0016 3E 00 MVI A, 00H

0018 21 FA FE LXI H, 0FEFAH

001B 77 MOV M, A

001C 23 INX H

001D 77 MOV M, A

001E 21 FE FE LXI H, 0FEFEH

0021 E5 PUSH H

0022 21 7B 00 LXI H, 007BH

0025 D1 POP D

0026 7D MOV A, L

0027 12 STAX D

0028 13 INX D

0029 7C MOV A, H

002A 12 STAX D

002B 21 FC FE LXI H, 0FEFCH

002E E5 PUSH H

002F 21 EA 00 LXI H, 00EAH

0032 D1 POP D

0033 7D MOV A, L

0034 12 STAX D

0035 13 INX D

0036 7C MOV A, H

0037 12 STAX D

0038 21 FA FE LXI H, 0FEFAH

003B E5 PUSH H

003C 21 FE FE LXI H, 0FEFEH

003F 7E MOV A, M

0040 23 INX H

0041 66 MOV H, M

0042 6F MOV L, A

0043 E5 PUSH H

0044 21 FC FE LXI H, 0FEFCH

0047 7E MOV A, M

0048 23 INX H

0049 66 MOV H, M

004A 6F MOV L, A

004B D1 POP D

004C CD 56 00 CALL 0056H

004F D1 POP D

0050 7D MOV A, L

0051 12 STAX D

0052 13 INX D

0053 7C MOV A, H

0054 12 STAX D

0055 76 HLT

0056 44 MOV B, H

0057 4D MOV C, L

0058 21 00 00 LXI H, 0000H

005B 79 MOV A, C

005C 0F RRC

005D D2 61 00 JNC 0061H

0060 19 DAD D

0061 AF XRA A

0062 78 MOV A, B

0063 1F RAR

0064 47 MOV B, A

0065 79 MOV A, C

0066 1F RAR

0067 4F MOV C, A

0068 B0 ORA B

0069 C8 RZ

006A AF XRA A

006B 7B MOV A, E

006C 17 RAL

006D 5F MOV E, A

006E 7A MOV A, D

006F 17 RAL

0070 57 MOV D, A

0071 B3 ORA E

0072 C8 RZ

0073 C3 5B 00 JMP 005BH

*Assembly Listing, including assembled bytes and source*

:100000002100FF31FAFE3E0021FEFE7723773E00FD

:1000100021FCFE7723773E0021FAFE77237721FE2D

:10002000FEE5217B00D17D12137C1221FCFEE5212F

:10003000EA00D17D12137C1221FAFEE521FEFE7E3C

:1000400023666FE521FCFE7E23666FD1CD5600D17D

:100050007D12137C1276444D210000790FD261008D

:1000600019AF781F47791F4FB0C8AF7B175F7A175A

:0600700057B3C8C35B009A

:00000001FF

*Intel HEX File generated by assembler*

Note the last line. It contains no data bytes (size field = 00) and has a field code of 01. This is the end-of-file demarcation for Intel HEX. On the first iteration (base = 0), we collect all bytes in the range 0x0000 – 0x001F, so after running the iHEX 🡪 EZDS converter, we get out a file containing a single line:

:00002100FF31FAFE3E0021FEFE7723773E0021FCFE7723773E0021FAFE77237721FE

*EZProm DataStream File generated by iHEX 🡪 EZDS conversion program*

*[Lines = 5, Base = 0x0000]*

Then, on the next iteration, we set our base to 32, and convert 32 more bytes:

:0020FEE5217B00D17D12137C1221FCFEE521EA00D17D12137C1221FAFEE521FEFE7E

*EZProm DataStream File generated by iHEX 🡪 EZDS conversion program*

*[Lines = 5, Base = 0x0020]*

Errors and Exceptions

The following exceptions are provided directly in the program:

Wrong number of lines. Must be between 1 and 16.

The number of address lines should be an integer between 1 and 16 (inclusive).

Invalid base address.

The base address should be a nonnegative multiple of the range of the address. Furthermore, since it is impractical to write beyond the 64K memory region, the base address should not exceed one range of the address less than 64K.

Wrong Intel HEX file extension. Expected: .hex, got:

The input file should be in the Intel HEX format, with the *.hex* extension.

Checksum invalid. Aborting rest of write...

The checksum in the provided Intel HEX file was incorrect; addresses and data may be wrong, and writing will cease.

Final Note: The accuracy of the *.ezds* file depends largely on the validity of the supplied Intel HEX file. If, for instance, the HEX file contains multiple instances of data at the same address, the resulting EZProm DataStream file will also contain conflicting information.