The Hex Parser

The Hex Parser is a C++ class developed for parsing hex files for use by the microcontroller boot loader. Hex files are the completely compiled binary data that must be sent to the microcontroller for programming. This page will work to describe the basic principles of a hex file and how the hex parser operates to work with the microcontroller bootloader to create a bootloadable application.

What is a hex file?

A hex file is a file composed of ASCII hexadecimal values. Detailed information about hex file format can be found on Intel HEX (Note: This wikipedia article makes Hex files very easy to understand, and it is highly recommended that users review this material to clarify any points missed here).

The basic form of a hex file is to understand that this is a generated binary file used for directly programming a microcontrollers memory. There are two main portions for this: Address information and Data information. Address information specifies WHERE the data should be programmed. Within our microcontrollers, 32 bits are used for the address. Hex files are used to SET the address values to be programmed and will then provide the DATA to program into those locations sequentially.

A hex file can be broken down into a number of lines. Each line specifies a single command to be executed, and there are a total of 6 different commands available for a hex file to implement. A 00 command code indicates that this is data to be programmed. A 01 command indicates that this is the last line of the HEX file. A 02 command is used to set the MIDDLE TWO bytes of the 4 byte address (AKA: 0x0ZZ0, where Z are set by the command).

A 03 command specifies the initial values of the CS and IP registers within the microcontroller. A 04 command specifies the two HIGHEST BYTES of the 4 byte address (aka 0xZZ00). Both command 02 and 04 will remain intact until a new setting is provided by either a 02 or a 04 command. These addresses are persistently kept for programming sequential data. Code 05 specifies more configuration initials for CPU registers. The bootloader program ignores any codes using 05, 01, and 03. Thus, there are only 3 actual commands used. The command always occupies the THIRD HEX byte of data on a line. That is, it will be a combination of the 7th and 8th characters in a row (in hex form).

Hex line form is also very intuitive. Break each two ascii values into a single hexidecimal byte. The first hexadeciaml byte (1st and 2nd digits) represent the size of the data section. This section will be discussed later. The following two hex bytes (characters 3 and 4, and 5 and 6) represent the LOW 2 bytes that will be used as an address for programming. These bytes are ONLY used when the command is 00 and specify the last two bytes of the address (aka 0x00ZZ where they replace ZZ). They are commonly used after a command to specify the two HIGHEST bytes to create a unique 32-bit address to program into. The next hex byte (7 and 8) represent our command code. All but the last two characters of the line represent the DATA to be programmed. The size of this section is specified by the first hex byte in the line, and can go up to 255 bytes at most. Typically, 16 bytes of data are programmed for each line. This data is programmed from the start address specified in the second and third hex bytes and goes sequentially. The final byte of the hex line is used as a checksum for the line. This value is the negative twos complement of the summation of all other hex bytes within the line. That means, if all hex bytes are added together, the sum will be zero. This is used by the microcontroller to validate the integrity of the data sent over the bootloader.

How to use the Hex Parser

The hex parser will take in a file name of a respective hex file. It will then look through the hex file and find the total number of hex lines by calling the `loadFile` function. The hex parser will then read through the hex file and provide a line by line binary interpretation of the hex line to a users program through calling the `getNextRecord()` function. Please ensure that you provide a sufficiently large buffer to store the hex information into! The hex parser will automatically move to the next line for reading of records sequentially for transmission across some boot loading protocol (currently in use: UART).

The hex file pointer can be manually reset to start at the beginning of the file if there is an error in the boot loader. This will effectively allow for a hard reset for boot loading.