EMP Series

PROGRAMMER

USER'S MANUAL

V4.0

NEEDHAM'S ELECTRONICS
(916) 924-8037
FAX (916) 924-8065
Visit us on the NET at http://www.needhams.com/

FCC CLASS B PRODUCT NOTICE

FCC WARNING STATEMENT:

NOTE: This equipment has been tested and found to comply with the limits for a Class B digital device, pursuant to Part 15 of the FCC rules. These limits are designed to provide reasonable protection against harmful interference in a residential installation. This equipment generates, uses and can radiate radio frequency and if not installed and used in accordance with the instructions may cause harmful interference to radio communications. However, there is no guarantee that interference will not occur in a particular installation. If this equipment does cause harmful interference to radio or television reception, which can be determined by turning the equipment on and off, the user is encouraged to try to correct the interference by one or more of the following measures:

- Reorient or relocate the receiving antenna.
- Increase the separation between the equipment and the receiver.
- Connect the equipment into an outlet on a circuit different from that to which the receiver is connected.
- Consult the dealer or an experienced Radio/TV technician for help.

CAUTION: Changes or modifications not expressly approved by the party responsible for compliance could void the users authority to operate the equipment

Table Of Contents

CHAPTER 1 - What can be programmed with the EMP Series Programmers?

EPROMS

EPROM's are cell based where each bit has a cell that is either charged or not charged to determine a logic one, true, or a logic zero, false. They can be bulk erased with ultraviolet light and reused many times. Many other devices use EPROM cells for the internal programmable memory sections.

ERASING AN EPROM

An erasable EPROM has a quartz window located on the chip just above the die. Exposing the EPROM to high-frequency ultra-violet light waves will erase an EPROM. This usually takes from 15-20 minutes, but may be shorter or longer, depending on the device.

Many manufacturers make EPROM erasers. If you would like to purchase an eraser, call Needham's Electronics, (916) 924-8037. When an EPROM is not being erased, the window may be covered with an opaque label. Sometimes (over a period of years) an EPROM will start to erase due to ambient fluorescent light. Direct exposure to sunlight also has this effect, but happens more rapidly.

EEPROMS

EEPROMS are Electrically-Erasable Programmable Read-Only Memory devices. EEPROMS program byte-by-byte (as opposed to FLASH memory, which must be erased before programming). No erasure is needed before programming any location, the EEPROM automatically erases each location before it is programmed.

FLASH MEMORY

Flash memories are devices that are also electrically erasable but may be erased of programmed in blocks. They also tend to require special algorithms to access internal registers in order to do any operations on the device.

MICRO-CONTROLLERS

These are CPU type deceives callable of executing strings of instructions. They have some type of internal programmable memory made up from the types above.

PLD's

PLD's are made up of logic arrays or groups of logic arrays that are configured by programming interconnect information to the internal memory. This memory may be made up from any of the above memories

NOVRAMS and SRAMS Modules

Generally these are made up of battery backed up SRAM and require only 5 volts to program. Erasing is not usually required but can be done by an interruption of power to the SRAM.

CHAPTER 2 - WARNINGS

Never place the target device in the socket backwards. If this happens and programming is attempted, it could permanently damage the device and is not good for the EMP. To avoid accidentally doing this, always execute a verify erase before programming the part. When installing the Family Module, be careful not to misalign the pins. Do not remove or insert a device in the ZIF socket while programming, verifying, or reading. This could damage the device and/or the programmer. When not programming, it is not necessary to power down the EMP when removing or inserting devices or family modules.

CHAPTER 3 – Getting Started

INSTALLING NEEDHAM'S ELECTRONICS DEVICE PROGRAMMER SOFTWARE FOR WINDOWS*

MINIMUM SYSTEM REQUIREMENTS

- An IBM-Compatible PC, 386 or above, running one of the following:
  Microsoft Windows 95, 98, ME, NT, 2000
- Approximately 55-75Mb of free space, for a full install, (Once the software is installed, less is needed).
- A parallel port (can be bi-directional, SPP, EPP, or ECP)
- At least 16MB of RAM to start Windows Software

*For NT\2000 systems, you must be in **administrator mode** to install the software (once the software is installed, you may run it without administrator privileges).

INSTALLING EMP DEVICE PROGRAMMING SOFTWARE FOR WINDOWS

When downloading the software from the Needham's Electronics Inc. web page

Hold down the shift key, and click on the download link. When asked where you wish the file to be saved, select a convenient location.

To execute the installer use any one of the following:

- In Windows/NT/2000 explorer, double-click on the installer.
- Use the Start > Run dialog to find and execute EMPWIN.EXE.

Installing from CD-ROM

1. Place the CD into your CD-ROM drive.
2. The master installer should start automatically. If the master installer does not start automatically:
   a. Double-click on the "My Computer" Icon on the desktop.
   b. Double-click on the Drive letter of the CD-ROM.
   c. Double-click the "Setup.exe" file, and follow the prompts.
3. When the master installer starts, you will see a form with multiple programmers pictured. Double-click on the programmer for which you want to install software.
4. The master installer will then launch the software installer.

The installer will prompt for the location you want the software installed. By default this creates the directory:
"C:\Program Files\Needham's Electronics\EMP Device Programming software", where the software will be installed. Note that you can change this directory to any convenient directory on your hard drive.

For Windows NT/2000 systems, the installer also installs a special kernel driver (you must be in **administrator mode**).

CONNECTING THE PROGRAMMER

1. Plug appropriate wall transformer into AC source, then insert its output plug into the programmer.
2. Connect the standard printer cable to the programmer, then to your PC printer port.

RUNNING EMP DEVICE PROGRAMMING SOFTWARE

Click on the task-bar START button, then select "Programs". By default, the installer adds a submenu to the "Programs" menu called "Needham's Electronics". Move the pointer to this selection and two choices are presented – one is a link to Needham's Electronics web page, and the other is called "EMP Device Programming software" – click on "EMP Device Programming software". No power switch exists on the programmer- the programmer is software activated – in a few seconds, you should see the power LED on the programmer light up. You are now ready to use your programmer. NOTE: EMPWIN attempts to find programmer by searching the I/O ports normally used for printers. This can cause interference with running printers or other parallel port d evices during the scanning process. However, the software only scans for the programmer the first time it runs, and should not need to scan the parallel ports again unless the programmer is moved to another parallel port.

WORKING WITH THE EMP DEVICE PROGRAMMING SOFTWARE

Working with the EMP Device Programming Software should be one of the easiest parts of the whole developing process. Using the mouse you should be able to easily navigate to the EMP's menus. Below you will find a guide to help you start programming.

***Programming a new device when the file is on disk*.**

1) Choose the device that you wish to program by selecting DEVICE
    a. Choose the Manufacture of the device i.e. Intel
    b. Choose the Device using the chip number i.e. 27c010
2) A graphic will appear showing the proper Family Module and Adapter
    a. Place these on the programmer as shown
3) Load the file that is to be programmed into the device by pressing EDIT BUFFER
    a. In the new window select File::Open
    b. Change Look In: to the folder that contains the file to be loaded
    c. Select the file that you want to load
    d. If you know the format of the file, change File of Type: to match, if the file type is not known choose AutoDetect
    e. Select Open
    f. If AutoDetect was chosen for the File of Type: a confirmation window will appear confirming the type of file, select Yes if you agree with the File Type shown.
    g. The next window allows you set three options
        i. Preload buffer with erase value with load the buffer (where all information that is programmed into the device is held) with the erased value of the device in case the file that is loaded does not fill the entire buffer
        ii. File 00000000 to FFFFFFFF will set from and to what address in the file we start loading from
        iii. Buffer 00000000 to ???????? (the last address in the device) will set from and to what address in the buffer we started loading to
    h. You will get a confirmation confirming that you want to load the file that you chose
4) Place the device properly into the socket
5) Select PROGRAM to program the device
6) Although the device is verified after programming, if you would like to verify the device select VERIFY

Programming a used device when the file is on disk

1) Choose the device that you wish to program by selecting DEVICE
    a. Choose the Manufacture of the device i.e. Intel
    b. Choose the Device using the chip number i.e. 27c010
2) A graphic will appear showing the proper Family Module and Adapter
    a. Place these on the programmer as shown
3) Load the file that is to be programmed into the device by pressing EDIT BUFFER
    a. In the new window select File::Open
    b. Change Look In: to the folder that contains the file to be loaded in
    c. Select the file that you want to load in

d. If you know the type of file the file is change File of Type: to match, if the file type is not known choose AutoDetect

e. Select Open

f. If AutoDetect was chosen for the File of Type: a confirmation window will appear confirming the type of file, select Yes if you agree you the File Type.

g. The next window allows you set three options

  i. Preload buffer with erase value with load the buffer (where all information that is programmed into the device is held) with the erased value of the device in case the file that is loaded does not fill the entire buffer

  ii. File 00000000 to FFFFFFFF will set from and to what address in the file we start loading from

  iii. Buffer 00000000 to ???????? (the last address in the device) will set from and to what address in the buffer we started loading to

h. You will get a confirmation confirming that you want to load the file that you chose

4) Place the device properly into the socket

5) Erase the device

  a. Any device that is OTP (one time programmable) cannot be programmed again

  b. If the device has window on the top. This device is erased using an UV eraser

  c. If the device is an EEPROM. These device do not need to be erased before being reprogrammed, but to blank the device program with FF

  d. If the device is Flash. Choose ERASE at the main menu.

6) Select PROGRAM to program the device

7) Although the device is verified after programming, if you would like to verify the device select VERIFY

Programming a device from a master device

1) Choose the device that you wish to program by selecting DEVICE

  a. Choose the Manufacture of the device i.e. Intel

  b. Choose the Device using the chip number i.e. 27c010

2) A graphic will appear showing the proper Family Module and Adapter

  a. Place these on the programmer as shown

3) Place the master device properly into the socket

4) Read the master device by selecting READ

5) Place the device to be programmed properly into the socket

6) Erase the device if needed

  a. Any device that OTP (one time programmable) cannot be programmed again

  b. If the device has window on the top. This device is erased using an UV eraser

  c. If the device is an EEPROM. These device do not need to be erased before being reprogrammed, but to blank the device program with FF

  d. If the device is Flash. Choose ERASE at the main menu.

7) Select PROGRAM to program the device

8) Although the device is verified after programming, if you would like to verify the device select VERIFY

CHAPTER 4 -- EMP MAIN-MENU COMMANDS

4.1 PROGRAMMING THE DEVICE

To program the device, make sure the device is correctly inserted into the ZIF socket and the lever is down. Then Check the Buffer Range, Device Range, Split and Set values before you start.(These values should default to their correct values)when a device is selected, the default algorithm is selected also.
When ready, press ‘Program’. The device can be automatically verified after programming.

4.2 VERIFYING ERASURE

To verify the device is erased, make sure the device is correctly inserted into the ZIF socket and the lever is down. Otherwise, the device may appear to be erased, even though it isn’t. Also check the device range before starting When ready, press ‘Blank Check’ to verify erasure.

4.3 VERIFYING THE DEVICE

When verifying a device, the content of the buffer and content of the device are compared through the specified ranges. If all bytes compare, the message ‘Verify Successful’ will appear on the screen. After this, press Enter to return to the main menu. Now the device contains the same information as the buffer. If there is a bad comparison, the log will contain the error information. At this point, press any key to return to the main menu.

4.4 READING THE DEVICE INTO THE BUFFER

To read the device into the buffer, make sure the device is correctly inserted into the ZIF socket and the lever is down. Then Check the Buffer Range, Device Range, Split and Set values before you start. Caution: Reading the device into the buffer destroys the buffer contents through the range of the read. Make sure anything in the buffer that is needed has been saved. When ready, press ‘Read’ to read the device into the buffer.

4.5 SELECTING THE DEVICE

The Part Selection Menu is where the device that will be used for all main-menu operations (i.e. programming, reading) is selected.

Selecting a Manufacturer

All of the supported manufacturers are displayed in the left box. To select a manufacturer use the arrow keys to scroll the cursor bar to the manufacturer. As the cursor bar scrolls down each manufacturer, a new list appears in the right-hand box. These are the devices that are supported for this manufacturer. Once the manufacturer has been selected, press Enter. If you need to re-select the manufacturer, press <TAB> to bring the cursor back to the Manufacturer menu.

Selecting a Device

Once the manufacturer has been selected with an Enter, the cursor bar moves to the right-hand box, and all devices supported for the selected manufacturer are displayed. To select a device, the same procedure is used as above. Some lists exceed the capacity of the screen. In this case, continue to use the arrow keys to scroll past the screen. The screen will automatically adjust so that the unseen devices will display.

Once the device has been selected, press Enter. This will select the device and return EMP to the main menu. Make sure that the device is displayed in the title bar. If the device has not been selected properly, press ‘Device’ again to reselect the device.

Using the Mouse

When selecting a manufacturer and/or device, the mouse may be used. With the mouse, simply point the mouse-cursor to the manufacturer and click once. Then point to the proper device and click again. This will select the device, just as above.

4.6 THE BUFFER EDITOR

The buffer editor allows you to view and/or edit the data in the data buffer. Data that has been read in from a file or a device is placed in the data buffer.

When loading a file, note the file range to make sure the buffer will be loaded through the range of the device. Also note the buffer range to see where the device data will be loaded.

To enter the buffer editor, press ‘Edit Buffer’. Once in the editor, there is no need to quit to return to the main menu.

LOADING IN A FILE

Loading a file is done from within the Buffer Editor. Before loading in a file, check the buffer range and the file range. Make sure the files of type is compatible with the file being loaded in. There are currently 6 file formats: BINARY, HEX (i.e. ASCII), INTEL HEX, IH AUTO, MOTOROLA S, and JEDEC. If the file name cannot be seen in the window make sure that the extension of the file is one of the filters. Make sure the filename has been entered correctly. The contents of the buffer through the specified range will be written over by the new file. Make sure everything in the buffer has been saved or is not needed. Press ‘Open’ to transfer a file to the buffer.

SAVING THE BUFFER TO A FILE

Before Saving a file, check the buffer range and the file Range. Make sure the file of type is compatible with the file loaded . There are currently 6 file formats, BINARY, HEX (i.e. ASCII), INTEL HEX,IH AUTO, MOTOROLA S and JEDEC.

Make sure the filename has been entered correctly.

The contents of the buffer through the specified range will be written into the new file, completely erasing any existing file with the same name. Before saving to disk, make sure that no file with the same name exists, or is needed.

Press ‘Save’ to save the buffer contents to disk.

4.7 EDITING THE ENCRYPTION TABLE

Some devices (i.e. micro-controllers) have 16 or 32 byte encryption arrays that allow restricted access to the device. Each byte of data from the device is XNOR’ed with each successive byte of the encryption table. Therefore, the only way to remove data from an encrypted device is to know the contents of the encryption array.

The Ecryption Table Editor command is only visible when a device with an Encryption Table is selected.

To change the encryption table, first select the device, then place the device in the ZIF socket and make sure the lever is down, then select option ‘A’. EMP will attempt to read the device identifier to establish the size of the encryption array(Some devices have different size arrays that cannot be determined from the part number on the package). If the device is not properly placed in the ZIF socket, EMP will use a default array size.

Once this is done, two menus will appear. The top one will display the encryption array. To enter the array, simply enter each byte, in hex, until finished. To move about in the array, use the arrow keys. Press Enter to enter the array into memory, or Esc to abort entering the array and return the array to its original state.

Once the encryption array has been entered, it is saved to disk (in the file EMP20.IOP). The next time EMP is loaded, the same encryption array will be in memory.

4.8 ENCRYPTION MODE

The second menu, below the encryption array, displays the current encryption mode. When entering EMP, this defaults to OFF. To toggle the encryption mode to ON/OFF, press TAB.

When the encryption mode is on, a message displays at the bottom of the screen, 'Encryption mode is on'. This means that when reading (option 4) the device into memory or verifying (option 3) the device, the encryption array will be used on the data read from the device. To test a device after it has been encrypted, load the data buffer before encrypting. After it has been encrypted, turn the encryption mode ON, and then verify the device. if the device verifies (encryption mode ON), then the encryption was successful.

Note: At least one security bit must be set in order for the encryption array to be useful. Without this, the encryption array may be read and/or patched over. Select option 'A' to program security bits.

4.9 PROGRAMMING SECURITY BITS

Security bits on Intel and Signetics microcontrollers and PLD's are used to prevent unauthorized access to the contents of the EPROM array on the device. Once the security bit(s) have been set, there is no way to program/read the device.

Devices that support encryption tables and/or security bits fall into 4 categories:

1. Single bit security systems—
   supported devices: Intel- 8751H, All GAL devices(16V8, 20V8, 22V10)
2. Two bit security systems with 16-byte encryption array supported devices: Signetics- 87C751, 87C752.
3. Two bit security systems with 32-byte encryption array supported devices: Signetics- 87C51,87C52,87C451,87C550,87C552,87C652 87C654,87C528 Intel -8751BH,8752BH
4. Three bit security systems with 64-byte encryption array supported devices: Intel -87C51, 87C51FA/FB/FC In a single bit security system, there is only one bit, which is either inactive (unprogrammed), or active (programmed). Once set, this bit makes it impossible to read the device contents.
   In a two-bit security system, the bit combinations are as follows:

   U=unlocked P=programmed

1. UU (both bits unlocked)--When both bits are unlocked, there is no lock on the device. However, any encryption array programmed into the device will be valid and if used, will encrypt data on the device.
2. PU (security bit 1=ON, bit 2=OFF)--This combination disables the device from being programmed further, and MOVC instructions executed from external program memory are disabled from fetching code bytes from internal memory.
3. PP (security bit 1= ON, bit 2=ON)--When both bits are set, all of the above-mentioned features are active, also the device may no longer be verified.

In a three-bit security system, the bit combinations are:

1. UUU (all bits unlocked)--When all bits are unlocked, there is no lock on the device. The encryption array is still valid, and if used, will encrypt any data on the device.
2. PUU (sec. bit 1=ON, bit 2=OFF, bit 3=OFF)--This combination disables the device from being programmed further, and MOVC instructions executed from external memory are disabled from fetching code bytes from internal memory.
3. PPU (sec. bit 1=ON, bit 2=ON, bit 3=OFF)--Same as above. In addition, the device may no longer be verified.
4. PPP (all security bits on)---All the above features will be active, in addition external execution will be disabled.
   2.16PROGRAMMING ENCRYPTION TABLES

Once the Encryption Array has been defined, you'll need to program it into the device. Make sure the device is properly seated in the ZIF socket, and the lever is down. Then, press ('C') to program the Encryption Array into the device. Once the device has been secured(Choice 'B'), you'll need to use the same encryption array to read that device.

4.10 BUFFER START – BUFFER END

This sets up where the programmer will start reading and writing from and to within the buffer.

4.11 DEVICE START – DEVICE END

This sets up where the programmer will start reading and programming from and to within the device.

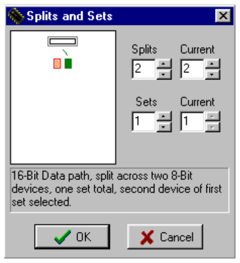
4.12 BUFFER CHECKSUM

To do a checksum of the buffer contents, press 'BUFEER CHECKSUM'. Before doing so, however, check the buffer range to make sure it is correct. When a buffer checksum is calculated, FF's are included. Some checksum methods disregard FF's in their calculations. Also, the same method of calculating a checksum is not always used. Therefore, the checksum generated by EMP may not coincide with a checksum done on the same data by another means.

4.13 DEVICE CHECKSUM

To do a checksum of the device, press 'DEVICE CHECKSUM'. Before doing so, however, check the device range, as the checksum is done only for this range. Also make sure the correct device has been selected.

When a device checksum is calculated, FF's are included. Some checksum methods disregard FF's in their calculations. Also, the same method of calculating a checksum is not always used. Therefore, the checksum generated by EMP may not coincide with a checksum done on the same data by another means



Splits: 2 Current: 2
Sets: 1 Current: 1

16-Bit Data path, split across two 8-Bit devices, one set total, second device of first set selected.

4.14 Splits

Splitting a device is used to program devices for systems more than 8 bits wide. For example, to program an 8-bit device for a 16-bit system, a split of 2 would be used. For a 32-bit wide system, a split of 4 would be used. To change the SPLIT value, press 'Splits and Sets'.

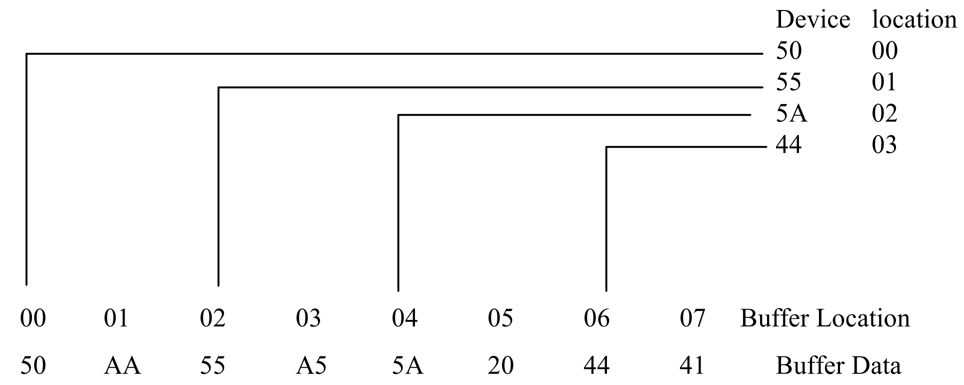
Once you have pressed 'Splits and Sets', a graphical menu will appear. The first box prompts you to enter the SPLIT factor (i.e. how many 8-bit devices you will be using). The second box prompts you to enter the current SPLIT device. For instance, if you have selected a SPLIT factor of 4, there are 4 devices in that set.

Programming a device

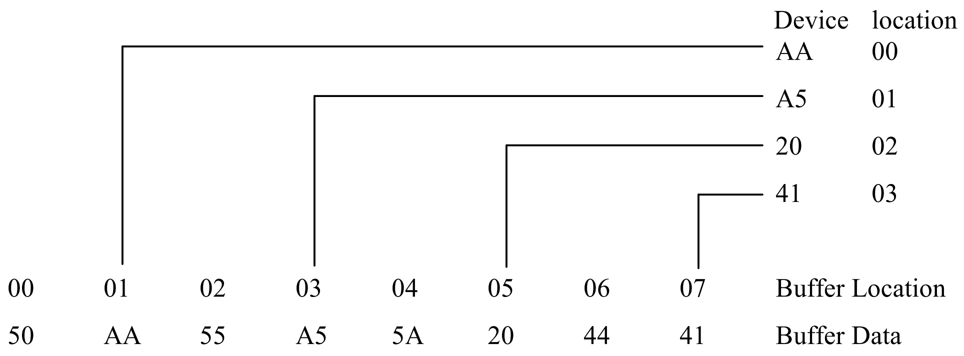
When programming a device with a SPLIT factor of 1, EMP programs the device sequentially. That is, one byte after another.   With a different SPLIT factor, however, EMP programs the device according to the number of SPLITS and the specified device number.

For instance, if a split of N1 has been selected, EMP takes every other (N1)th byte from the data buffer and puts it into the device. EMP also adjusts for the device number. If the device number is K1, Emp adjusts the buffer by K1 bytes. For example, if a SPLIT of 2 has been selected (N1=2), and the DEVICE number is 2 (K1=2) (Note that the buffer and device range are assumed to be 0), every 2nd byte will be programmed starting at address 1 (Bytes 1,3,5,7, etc.). Graphically, a SPLIT of 2 might be represented in this way:

SPLIT of 2 (device 1):


Device location
50 00
55 01
5A 02
44 03
00 01 02 03 04 05 06 07 Buffer Location
50 AA 55 A5 5A 20 44 41 Buffer Data

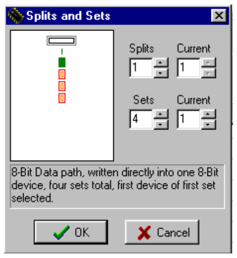
SPLIT of 2 (device 2)


Device location
AA 00
A5 01
20 02
41 03
00 01 02 03 04 05 06 07 Buffer Location
50 AA 55 A5 5A 20 44 41 Buffer Data

After both devices are programmed:

Device 1	Device 2	Memory Map
8 bits	8 bits	16 bits
50	AA	50AA
55	A5	55A5
5A	20	5A20
44	41	4441

Note: When selecting a split factor, EMP automatically re-adjusts the current buffer size. For instance, moving from a SPLIT factor of 1 to 2 doubles the size of the buffer, and moving from 1 to 4 quadruples it.



Splits 1 Current 1

Sets 4 Current 1

8-Bit Data path, written directly into one 8-Bit device, four sets total, first device of first set selected.

4.15 SETS

Sets allow you to place information from the data buffer into multiple devices. For instance, if the data buffer is 64k, and 8-kbyte devices are being used, it requires 8 devices.

To set the SETS values, press 'Splits and Sets'.

Once you have pressed 'Splits and Sets', two menus appear, in order. The first menu prompts you for number of SETS. The second menu prompts you for current SET number.

When programming, EMP adjusts for the set number. When a SET is selected, EMP simply takes the device length (i.e. 8-kbytes) and adds it to the buffer start for each device. For instance, if there are 4 SETS selected, and set number 3 has been selected, EMP points to the area for which SET 3 would be in memory. If 8-kbyte devices were being used, this would be whatever the buffer starting address is plus 16-kbytes (i.e. buffer start + 8k (device 2) + 8k (device 3)).

Graphically, data is drawn from the buffer like this (assume device length is 8k-bytes):

Buffer start (+ 0 bytes)	- Device 1
Buffer-start + 8 Kbytes	- Device 2
Buffer-start + 16 Kbytes	- Device 3
Buffer-start + 24 Kbytes	- Device 4
Buffer-start + 32 Kbytes	- Device 5, etc.

Note: When setting the SET factor, EMP automatically readjusts the current buffer size. For instance, moving from a SET factor of 1 to 2 doubles the size of the buffer, and moving from 1 to 4 quadruples it.

4.16 In Circuit Programming

Quite a few of the serial devices on the EMP-11 and EMP-30 Windows software can be programmed in circuit using the EICPA14 for the EMP-11 and the EICP14 on the EMP-30. This adapter allows you to program your device in circuit using the 14 pin header on the EICP(A)14. Following are the some of the adapter pinouts to device pinouts.

EIPA14 Pin	JTAG Device Pin	Xilinx configurators	Atmel DataFlash	Atmel Series AVR	Microchip PIC
D0	TERR	SDATA			Data
D1	TDO		SO	CLK	CLK
D2	TCLK	CLK	SCLK		
D3	TMS	CE	#CS		
D4	TSTAT	Busy	Rdy/#bsy	PB0	
D5	TDI	TDI	SI	PB1	
D6	TRST	RESET/#OE	RESET	PB2	
D7	JEN	#CEO	#WP		
Vpp	Vpp	Vpp			Vpp
Vcc	Vcc	Vcc	Vcc	Vcc	Vcc
GND	GND	GND	GND	GND	GND

Chapter 5 -- *Supported File Types*

There are currently 6 file formats: BINARY, ASCII HEX, INTEL HEX, MOTOROLA S, JEDEC, POF and BIT.

Note: When loading and saving make sure the file type is correctly set. A file may still load even though it is of the wrong file type, resulting in the transfer of incorrect data to the buffer.

5.1 BINARY FORMAT

This format is a binary string of data, which does not have any other header or footer data, that when read into the buffer is displayed as Hex values. Any file and file type can be read in as binary, although the chip will not work in circuit.

5.2 ASCII HEX FORMAT

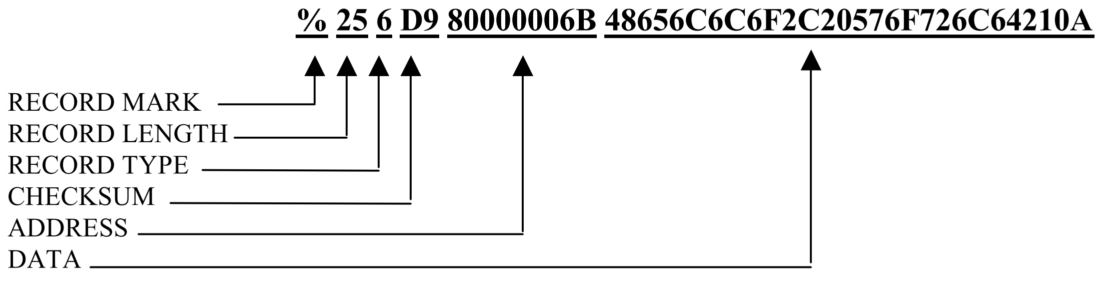
With this format, ASCII hex values are typed into an editor with a space separating each byte and transferred into to the buffer as they appear in the file. If the first line of the file is $A00000000, then the file will be offset by that value. $A00000010 will start loading at address 10, $A10000000 will start loading at address 10000000. The address must be 8 digits in length, starting with '$A' and ending with a ','.

5.3 TEKTRONIX EXTENDED HEX

Tektronix Extended Hex files have records always starting a percent (%) character, with each line consisting of 5 fields. These are the length field, the type field, the checksum, the address field (including address length), and the data field.

Record Structure

When a record is specified, it must follow a certain order so that the loader may read it properly:



When entering a record, all data (except the record mark, '%') is entered in ASCII hex. That is, when the word 'byte' is used, it refers to the two-digit hex number. i.e. '01020304' is 8 digits wide, but is only 4 bytes of data when converted.

Record Length:

The record length is always 1 byte and specifies the number of digits (not bytes) in the record, excluding the percent, the length field, the type field and the checksum. For instance, if there are 10 bytes of data in the current record (i.e. on the current line), the record length will be '0A'. Remember that all bytes are entered as their 2-digit ASCII-HEX values. 'A' is invalid and must always be '0A'.

Record type:

The record type is 1 character length (1 digit). Record Types are as follows:

6 -- Data Record
8 -- Termination Record

Checksum:

The checksum is a 1 byte (2 digits) field that represents the sum of all the nibbles on the record (excluding the record mark ':' and the checksum itself).

The checksum simply adds all of these numbers and the sum is the checksum. For instance, the line

%256D980000006B48656C6C6F2C20576F726C64210A

will have the checksum of 'D9'. Note that 'D9' is the underlined portion of the record. The sum of all nibbles excluding the checksum should equal the checksum. If this is not the case, the line is reloaded and tried again. If a record has an incorrect checksum, the appropriate error-message will be displayed, and EMP will discontinue loading of the file.

Address field:

The address field is nine characters (i.e. 9 digits) in length, and tells the loader where to put the current line of data. The first character is the address size and is always 8. The remaining 8 characters are the 4-byte address that specifies where the data is to be loaded into memory. In the example above the data would have been loaded to address 6B. Address fields may optionally contain the address of the instruction to which control is passed in termination records.

Data field:

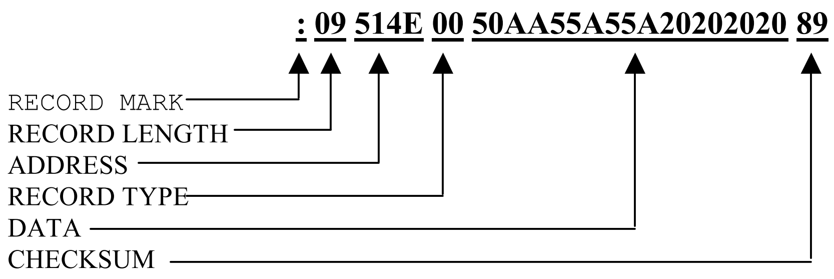
The DATA FIELD contains the actual data for the loader. If the record type is a Data record, the data field will contain actual file data. The termination record does not contain a data field. In this case, the data field is non-existent. The Data field is directly related to the record length. The record length minus the 9 characters of the address determines how large the data field will be. For instance, if the record length is 25, the data field will contain 15 bytes (30 digits) of information.

5.4 INTEL HEX FORMAT

Each line in an INTEL hex file corresponds to a 'record'. Each record tells the loader (in this case, EMP) what to do with that line. Each record starts with a Colon (:), called a record mark. If the record mark does not exist, the loader should ignore the line.

Record Structure

When a record is specified, it must follow a certain order so that the loader may read it properly:



When entering a record, all data (except the record mark, ':') is entered in ASCII hex. That is, when the word 'byte' is used, it refers to the two-digit hex number. i.e. '01020304' is 8 digits wide, but is only 4 bytes of data when converted.

Record Length:

The record length is always 1 byte and determines how wide the data field is. For instance, if there are 10 bytes of data in the current record (i.e. on the current line), the record length will be '0A'. Remember that all bytes are entered as their 2-digit ASCII-HEX values. 'A' is invalid and must always be '0A'.

Address field:

The address field is two bytes (i.e. 4 digits) in length, and tells the loader where to put the current line of data. Address fields are only valid in data records. For record types that do not use the address field, the address field is always '0000'.

Record type:

The record type is 1 byte in length (2 digits). Record Types are as follows:

00 -- Data Record
01 -- End-of-File Record
02 -- Extended Address Record
03 -- Start-Address record.

How these record types are used is discussed later. The record type determines what the current record (i.e. the current line) will instruct the loader (i.e. EMP) to do. For instance, a record of type '00' (data record) specifies that this record contains data; record type '01' (End of file record) terminates loading of the file, and so on.

Data field:

The DATA FIELD contains the actual data for the loader. If the record type is a Data record, the data field will contain actual file data. If the record type is a Start-address or Extended address, the data field will contain these addresses. The End-of-file record does not contain a data field. In this case, the data field is non-existent. The Data field is directly related to the record length. The record length determines how large the data field will be. For instance, if the record length is 5, the data field will contain 5 bytes of information.

Checksum:

The last byte (i.e. last 2 digits) of the record is an 8-bit checksum of the entire record (excluding the record mark ':' and the checksum itself).

The checksum simply adds all of these numbers and takes the twos compliment as a checksum. For instance, the line

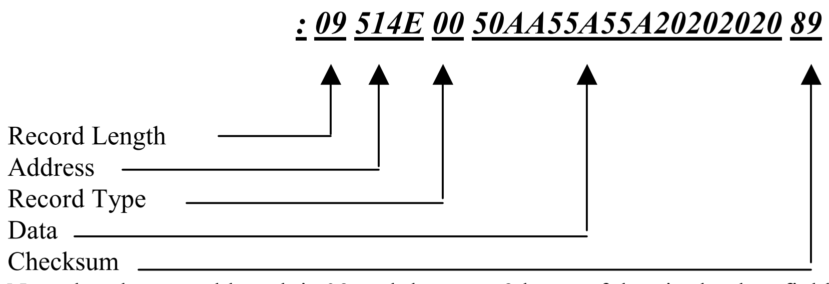
:0900000050AA55A55A2020202029

will have the checksum of '29'. Note that '29' is the last byte of the record. When a line is loaded, the negative of the sum of all bytes prior to the checksum (i.e. 0-SUM (bytes) should be equal to the checksum. Or, in another way, the sum of all bytes including the checksum should be 0. If this is not the case, the line is reloaded and tried again. If a record has an incorrect checksum, the appropriate error-message will be displayed, and EMP will discontinue loading of the file.

Record Types

Data Records:

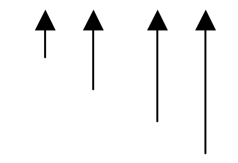
The data record specifies that the data field in the current record will be file data. This is the actual data that is being loaded

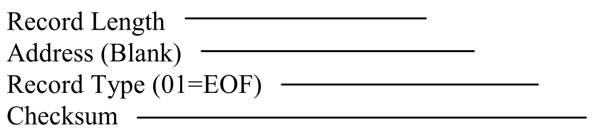


Note that the record length is 09 and there are 9 bytes of data in the data field. Since the address is 514E, data for this record will be loaded sequentially starting at address 514E (relative to the start of the buffer).

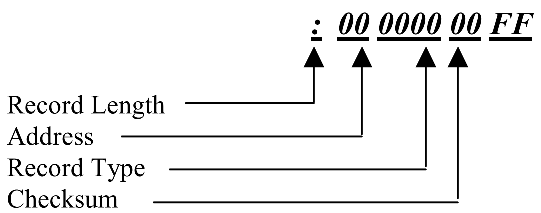
End-of-File Record:

The EOF record specifies the end of the file. At this point, the loader discontinues loading the file.





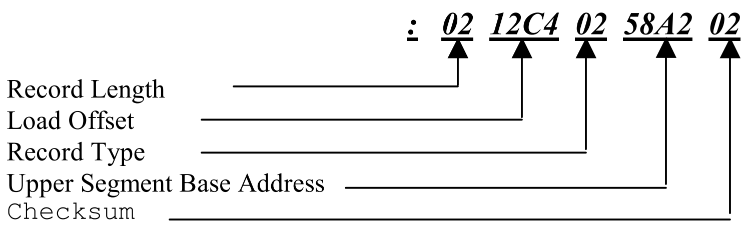
Note that the data field is 0 bytes in length and that the address contains “0000”.
There is another form of EOF record, referred to as the Alternate EOF Record



Although the record type is a data record, there is no data so the record is taken as the end of file. EOF records are not variable records. That is, they always appear as they do here.

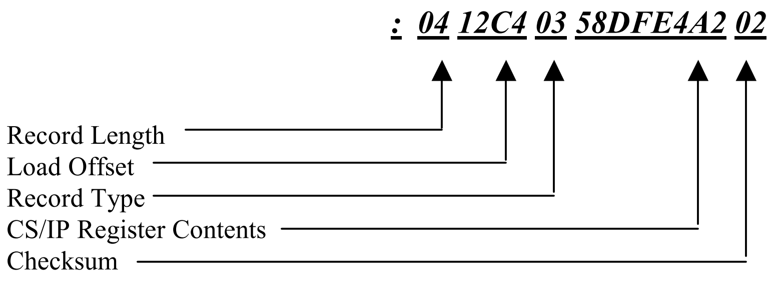
Extended Segment Address Record:

The extended segment address record specifies the segment that data will be loaded into. For example, if the segment is specified as “58A2” the loader will change the segment to 58A2H, which will cause data to be loaded at address 58A20H. In the following data records, the address specified in the address field will be added to the new extended address. For example, if the address in the data record is “12C4” and the extended address is “58A2”, the actual address will be 58A2:12C4 or 58A20+12C4 or 59CE4. The extended address will continue to offset data record addresses until a new extended address record is specified. The record length is only “02” since the segment address is the only data and is only 2 bytes long.



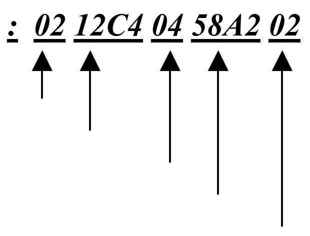
Start Segment Address Record:

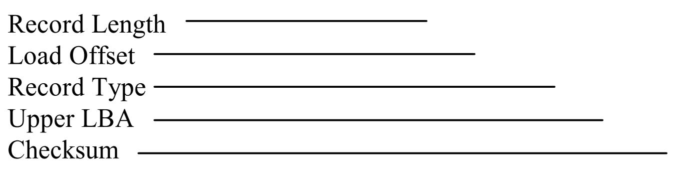
The start segment address record specifies the execution start address for the object file. The value given is the 20-bit segment address for CS and IP registers of 8- and 16-bit Intel processors.



Extended Linear Address Record:

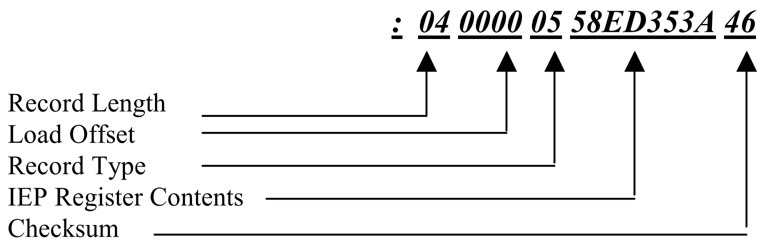
The extended linear address record is for 32-bit processors only. It is used to specify bits 16-32 of the linear base address (LBA). It may appear anywhere within a 32-bit hexadecimal object file. Its value remains in effect until another extended linear address record is encountered.





Start Linear Address Record:

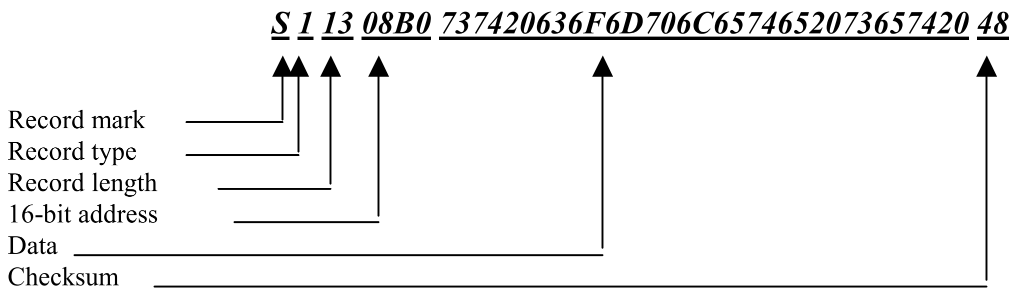
The start linear address record is used to specify the execution address of a 32-bit Intel object file only.



5.5 MOTOROLA S HEX FORMAT

Motorola S format is very similar to INTEL hex format. The source file is an ASCII-HEX file, with each line in the file representing a record.

Record Structure



Record Marker:

The record marker in the Motorola file is in ‘S’. That is, each line is preceded with a ‘S’.

Record Type:

Unlike INTEL files, this record type is only 1 digit in length. For instance, record type 1 is represented by ‘S1<...>’, not ‘S01<...>’.

Record Length:

Following the record type is the record length. It’s a Two-digit hex value specifying the number of data bytes in the record, including the address and checksum. Each byte is represented as two hex characters like Intel Hex format.

Load address:

The Load Address determines where in memory the current record (i.e. current line) will load. Instead of using Extended Address record types to load data above 64k, the Motorola file uses the actual address. For this reason, the address field may vary in size. The size of the address field may be 2,3, or 4 bytes (16, 24, or 32 bits), and is determined by the record type (discussed below).

File Data:

Following the address is the actual file data. The record length indicator determines the number of bytes of data. The EOF record does not contain file data.

Checksum:

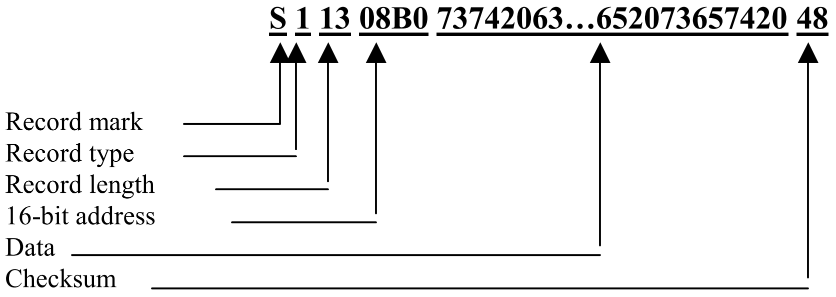
The checksum is the last byte (2 digits) in the record, and follows the file data. The checksum is calculated differently than the INTEL hex format. All bytes with the exception of the 'S', record type digit, and the checksum itself, are added and then negated (i.e. NOT checksum). For instance, if the sum of the record is 4A, the INTEL checksum would be B6, where the MOTOROLA checksum would be B5. Thus, if all numbers plus the checksum are added, the result should be 0FFH (not 0 like the INTEL format).

Record Types

Motorola HEX files have the following record types:
1 -- 16-bit addressing
2 -- 24-bit addressing
3 -- 32-bit addressing
7 -- End of file record
8 -- End of file record
9 -- End of file record

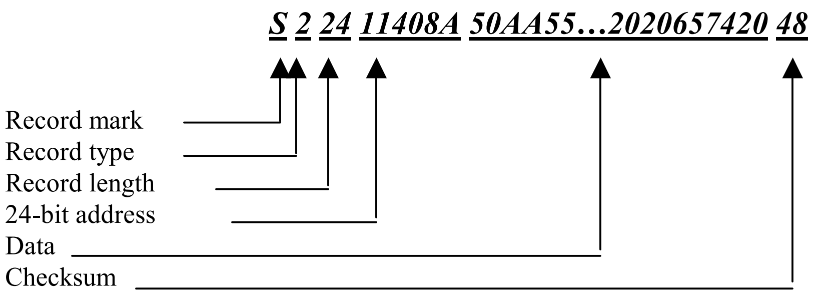
16-bit addressing:

Files less than 64k only need 16 bits to address every location in the file.
When files are this small, the data record used is '1'.



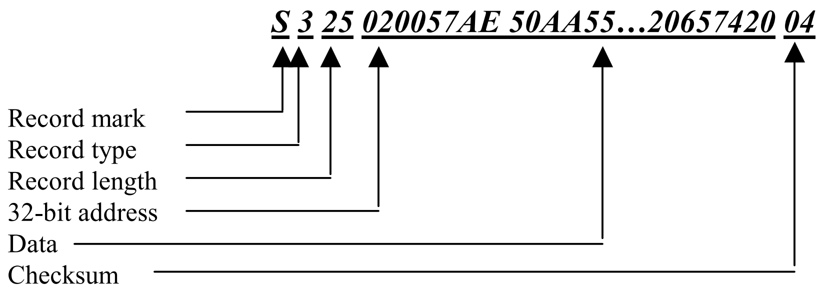
24-bit addressing:

With files greater than 64k, more bits are needed to address every location within the file. Depending on the number of bits are needed to address the file, 24 or 32 bits may be used for addressing. If only 24 bits are needed, file type '2' is used to specify 24-bit addressing.



<u>32-bit</u> <u>addressing:</u>

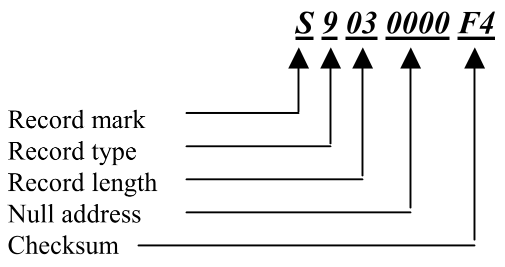
In the case where more than 24 bits are needed, record type ‘3’ may be used to specify 32-bit addressing:



Note that in each case the record length increased. Specifically, the record length for 16-bit addressing was 23. When the record length increased to 24-bits, the record length moved to 24. The record length increased to 25 when 32-bit addressing was used. This is due to the fact that one byte in each case was added due to the extra byte in the address. Since the record length includes the address, it must be increased also.

<u>End-of-file</u> <u>record:</u>

Record types 7, 8, and 9 represent the end of the file. The different End-of-file records are used with different record types—record types 3, 2, and 1 respectively. That is, record type ‘3’ (32-bit addressing) uses End-of-file marker type ‘7’, record type ‘2’ (24-bit addressing) uses End-of-file marker type ‘8’, and record type ‘1’ (16-bit addressing) uses End-of-file marker type ‘9’. An example of record type ‘9’:



NOTE ON FILE POINTERS

When using EMP with INTEL hex files, the message ‘0 bytes loaded’ may appear. In this case, the offset in record type ‘02’, the extended address record, may be out of range for EMP. For instance, if the extended address is at ‘F8000’, and the file buffer ranges from 0-FFFF, no data after the ‘F8000’ specification will be loaded. In this case, change the file offset in to coincide with the extended address range in the INTEL hex file. For instance, in the case of ‘F8000’, set the initial address in the file buffer range to ‘F8000’ (the ending address will adjust automatically). This will allow the file to load properly.

In the case of MOTOROLA S files there is no extended address. The address is encoded directly in the address field (the size of which is determined by the record type). In this case, adjust the file pointer value to the initial address value.

5.6 JEDEC FORMAT

The JEDEC file format defines the standard for transferring data for PLD’s from a development system to a device programmer. The standard outlines the transfer of fuse, test, identification and comment information in an ASCII representation. It does not define device specific information such as algorithms and methods for actually programming the device. The JEDEC standard utilizes a transmission protocol, which has error checking in the form of a checksum, and a start/end transmission character, which is used when a device programmer shares a serial port with the programmer.

JEDEC FILE

<STX>The area from the start-of-text character to the first asterisk
is a note field, and can be used to hold any text the user wishes.*

```
N Fields beginning with N are note fields as well.  They can be used
to hold comments about the individual lines in a JEDEC file, as they
are below.  All fields are ended with an asterisk, and all numeric
values are in decimal.*
The QP field = # of pins in the device*
QP20*
The QF field = # of fuses in the device*
QF448*
The QV field = # of test vectors in this file*
QV8*
F = Default fuse state(any blank areas are filled with this)*
F0*
X = Default vector state. Any pins with X vectors will be
driven high if X1, and driven low if X0.*
X0*
These are the fuses. The number next to the L is the address
(See your PLD development system documentation for details),
and the fuse states follow the space.*

L0000 1111101111111111111111111111*
L0028 1011111111111111111111111111*
L0056 1110111111111111111111111111*
L0112 0101011101111011111111111111*
L0224 0101011101110111111111111111*
L0336 0101011101110111111111111111*
These are the test vectors.
    V0001 000000XXXNXXXHHHLXXN*
    V0002 010000XXXNXXXHHHLXXN*
    V0003 100000XXXNXXXHHHLXXN*
    V0004 110000XXXNXXXHHHLXXN*
    V0005 111000XXXNXXXHLHHXXN*
    V0006 111010XXXNXXXHHHHXXN*
    V0007 111100XXXNXXXHHLHXXN*
    V0008 111110XXXNXXXLHHHXXN*
    Each character in the vector corresponds to a pin on the device, starting with pin one.

    0 = Drive pin low
    1 = Drive pin high
    C = Drive pin low, then high, then low(Clock)
    K = Drive pin high, then low, then high(Inverted clock)
    L = Test output low, fail test if high
    H = Test output high, fail test if low
    Z = Test output for high impedance
    X = Drive pin to default X state(See X above)
    N = Power pin or untested output*

Fuse checksum field - 16 bit mod 8 checksum of
the FUSES only.*
C124E*
End-of-Text and transmission checksum.
16 bit mod 8 Checksum of all bytes in the file.*
<ETX>8646
```

5.7 POF FORMAT

POF stands for Programmer Object Files. The POF format is based on the "Tag Image File Format" used by Microsoft and Aldus for graphical data. All information in the POF's are contained in packets. Each packet contains a "tag" to identify the type of data the packet contains and the data itself. The entire POF file is composed of a header followed by a sequence of packets. The header has a constant length, always 12 bytes. It uses 4 bytes to describe itself as a POF header, 2 bytes to describe the byte order, 2 bytes for the version number and 4 bytes to describe the length or packet count. The packets have variable lengths and structures, but the first 6 bytes always follow the same format. The first 2 bytes, (always LSB..MSB), describe the tag number and packet type, the next 4 bytes are the number of bytes in the rest of the packet.

Note: In the following example the data types are:

"char" occupies 1 byte

"short" occupies 1 byte

"long" occupies 1 byte

The following examples show the format for 2 tag numbers:

(1)Creator_ID   tag=1

This packet contains a version ID string from the program that created a POF file. As a "C" construct it would look like:

```
Struct POF_Creator_ID
{
short     tag;              /* tag number      */
long      length;           /* number of bytes in the rest of packet */
char      creator[];        /* null-terminated string    */
}
```

In this case 4 bytes for length contain the characters used to describe the Creator ID

(5)Security_Bit   tag=5

This packet declares whether security mode should be enabled on the target device.

```
{
short     tag;              /* tag number      */
long      length;           /* number of bytes in the rest of packet */
unsigned short    security; /* zero if non security mode          */
                            /* non zero if security mode          */
```

To read a POF file a program examines each packet and if the tag value is recognized, the data in that packet is used. Otherwise the packet is ignored and the next packet is examined, etc. Tag numbers can be in any order except the terminator (8) must be in the Nth packet, where N is the packet count declared in the POF header. This packet also contains the checksum. For the present only the tag packet values of 1,2,3,5,8,and 17 are used by the EMP series.

5.8 BIT FORMAT

The Xilinx .bit format is pretty simple. It uses keys and lengths to divide the file.

Here is an example. Below is a hex dump from the beginning of a .bit file:

```
00000000  00 09 0F F0 0F F0 0F F0 0F F0 00 00 01 61 00 09                a
00000010  6A 75 6E 6B 2E 6E 63 64 00 62 00 09 73 30 35 76  junk.ncd b  s05v
00000020  71 31 30 30 00 63 00 0B 32 30 30 31 2F 31 32 2F  q100 c  2001/12/
00000030  32 38 00 64 00 09 31 39 3A 30 38 3A 31 32 00 65  28 d  19:08:12 e
00000040  00 00 1A 5C FF 20 0D 2D 9F 5F FE FE DE F3 BD 6F     \   - _     o
```

Field 1

```
2 bytes          length 0x0009                  (big endian)
9 bytes          some sort of header
```

Field 2

```
2 bytes          length 0x0001
1 byte           key 0x61                       (The letter "a")
```

Field 3

```
2 bytes          length 0x000a                  (value depends on file name length)
10 bytes         string design name "junk.ncd"  (including a trailing 0x00)
```

Field 4

```
1 byte           key 0x62                       (The letter "b")
2 bytes          length 0x000c                  (value depends on part name length)
12 bytes         string part name "s05vq100"    (including a trailing 0x00)
```

Field 4

```
1 byte           key 0x63                       (The letter "c")
2 bytes          length 0x000b
11 bytes         string date "2001/12/28"       (including a trailing 0x00)
```

Field 5

```
1 byte           key 0x64                       (The letter "d")
2 bytes          length 0x0009
9 bytes          string time "19:08:12"         (including a trailing 0x00)
```

Field 6

```
1 byte           key 0x65                       (The letter "e")
4 bytes          length 0x00001A5C (6748 bytes) (value depends on device type,
                                                (and maybe design details)
```

The FF 20 is the start of the bit stream at address 44 in the example. Everything from this point on is data, including the FF 20.

Remember that the data is shifted from the LSB of each byte in sequence, however it is stored in the file as MSB, so the starting sequence in the buffer will be 1 1 1 1 1 1 1 1 0 0 0 0 0 1 0 0. The software handles this conversion.

CHAPTER 6. DEFINITIONS AND TERMS

Some terms and definitions used throughout EMP and the help system are specific to EMP-Series. Some of them are as follows:

Device—

Device always refers to the device in the ZIF socket that is being programmed, read, verified, or otherwise manipulated by the EMP.
Buffer—
The buffer is the area of memory into which the contents of a device is read.
Main-menu—
The main-menu is the initial menu brought up by EMP. This is where reading, programming, and most other file and device functions are performed. There are two sub-menus, the part-selection menu, and the Buffer editor.

Part-selection menu—

To program, read, or verify a device, it must first be selected. To do this, chose DEVICE at the main menu to enter the part-selection menu. Once there, you may select a manufacturer, appropriate part, and package style.

Buffer Editor—
To view or edit the data in the buffer, use the EDIT BUFFER at the main-menu.

-- Warranty—

EMP-Series is sold on a 30-day satisfaction guarantee. It is further warranted against failure for one (1) year. Covered is replacement or repair, at our option, at no cost for parts, labor and second day return shipping within the United States. Needham's Electronics Inc. is not responsible for incidental damages that may arise out of use of the EMP-Series. If a failure is experienced, call Needham's Customer Service Monday through Friday between the hours of 8:00AM and 5:00PM at (916) 924-8037 for a Return Material Authorization.

After the warranty period expires, the EMP-Series can be repaired for a flat fee. This includes all parts, labor, upgrading to the latest revision, recalibration and second day return shipping within the continental US. Use the same RMA procedure as above. No other warranties apply or are implied.

Needham's Electronics Inc. reserves the right to modify the terms and conditions of the EMP warranty at any time. Failures due to abuse or misuse of the EMP-Series will void the warranty and will only be repaired on a time and material basis, an estimate will be provided and agreed to before any work is started.