User instructions for the "Atmega_in_circuit_programmer: Flash and EEPROM"

Preliminary steps:
A programming pcb is required, details of which are given in the BMP and Eagle attachments. (See previous posting on On Feb 22 2019.) An Atmega 328 must be programmed independently. The following configuration bytes are recommended:

The extended fuse byte	0xFF	Fuse byte high	0xD7
Fuse byte	0xE2	Lock bits	0xFF

Critical parameters are:

8MHz internal RC clock,	65ms SUT, BOD disabled
Watch dog under program control	EEPROM preserved during chip erase
No extra memory protection	

Obviously other sets of configuration bytes will also meet these requirements. However these bytes have been used in trials.

Once the pcb has been assembled and the Atmega devices loaded, the programmer will output the one or other of the following user prompts to a PC running a terminal program. Y Y Y Y Y.....
or Z Z Z Z Z Z.....
Note: The serial port settings are 56700 baud, 8 data bits, no parity, 1 stop bit and no handshaking.

Programming the flash memory
The user responds by echoing the user prompt and then following the on screen instructions. Below is a typical copy of the dialogue:

```
Y Y
ATMEGA328 detected
Press P to program target or R to run target application or -X- to escape
8 MHz internal clock.
Press -P- to send a program file or -E- to send a text file.
Send Program file.
Integer(0-FF)?  0000    **********************************************
```

It is normal to enter 0 in response to the user prompt after which program memory is read and the number of commands present compared with the number that were downloaded in the hex file.

```
Atmega_programmer_V2.0
Config bytes: Fuses extended, high, low and lock
00FF 00D7 00E2 003F     on-chip cal bit:  005B 005B
Hex_file_size:3840   d'loaded:  3840  in:  3840  out
Y Y Y Y Y Y Y Y.........
```

Notes:
1. The asterisks are printed out as the flash memory is read back for verification.
2. The term "on-chip" refers to the device hosting the programmer.
3. The "on-chip cal byte" is repeated. This means that it is user selected (see project "Using the Atmega 328 internal RC clock" posted here by osbornema on Jan 10, 2019 for details).
4. It is normally OK to used the default calibration byte in which case the "on-chip" cal byte is only given once.
5. The programmer code can now be copied to any number of additional chips. Note however that if identical programmers are allowed to run on the same pcb, UART contention issues will arise. To avoid this, slightly different user prompts must be implemented.

Programming the EEPROM

The user PC dialogue develops as follows:

Y Y The user enters Y at this prompt.
ATMEGA328 detected
Press P to program target, R to run target application or -X- to escape
8 MHz internal clock.
Press -P- to send a program file or -E- to send a text file.

Press E then W and the dialogue continues:
Press W, R or D to write to, read from or delete the EEPROM.? ? ?
Integer(0-FF)?

Enter 0.

A file containing strings and data to be programmed to the target device EEPROM must now be downloaded several times. See "Full_Atmega_Programmer\Sample EEPROM file_for_328" for a typical example. Press any key at the AK? prompt and the file with be echoed back to the PC with all the appropriate addresses in EEPROM.

The dialogue continues:

Send text file. Again! Again! Again! AK?
0005 First string to be saved to EEPROM follows first quotation symbol. Press AK to continue
005D All strings terminated in -cr-. AK?
0081 The program prints out the strings together with the address of the first char of each string.
00E0 Numbers plus an optional reference name can be entered below the second quotation symbol.
013A Numbers preceded with a -$- sign are interpreted in integer hex format
0181 The advantage of using hex numbers is that they can be combined into long numbers or split into char numbers.
01EF For this file the start address of the numbers is 0x229.
0228
0229 0096 150 00FA 250 D9AC -9812 FCD8 -808
0231 1234 4660 CDEF -12817 CDEF -12817 1234 4660
0239 7FED 32749 5678 22136 CBA9 -13399 8FED -28691
0241 5678 22136 9ABC -25924 0010 16 0020 32
….....
0339 8000 -32768 0002 2 0226 550 0294 660
0341 0302 770 0370 880 03DE 990 FDDA -550
0349 FD6C -660 FCFE -770 FC90 -880 FC22 -990 Y Y

Note:
The Integer(0-FF)? prompt invites the user to specify a section of EEPROM to be reserved for use by the application code.
If -D- is pressed in place of W or R the EEPROM is deleted leaving only the calibration bytes. Note this takes several seconds (a long wait!).
There is also a -C- option that reads the calibration bytes back. These will be 0xFF if the device has not been user calibrated.

The client program

Each test string is given a name (i.e. message_1, message_2 etc.). The client program includes a header file that contains a series of statements of the following form:

```
//Addresses in EEPROM of the various messages

#define  message_1        0005
#define  message_2        005D
….....................
#define  message_7        01EF

//Addresses in EEPROM of the numerical arrays

#define I_array   0x229
#define L_array   0x231
#define Ch_array 0x241
#define I_data 0x245
#define L_data 0x2B5
#define T_data 0x2F5
#define large_nos 0x325
```

The client program includes the following subroutines:

```
void Text_from_EEPROM(int EEPROM_address)
```

which is called using `Text_from_EEPROM( message_1);`
and prints out message_1.

```
char Char_from_EEPROM(int location)
int I_num_from_EEPROM(int location)
long L_num_from_EEPROM(int location)
```

Which return char, int or long numbers from the EEPROM.