Megaload for AVR

Boot loader outline

Megaload is a Bootstrap Loader for use with AVR microcontrollers and allows application firmware to be upgraded on a unit via a serial link without the use of a programmer. It has generously been provided by Sylvain Bissonnette and is available at http://www.microsyl.com

There are two programs required to support this function;

a) A windows application "MegaBootLoad" which allows the firmware application (eg.filename.hex) to be sent to the hardware over the serial link.

b) An embedded application (eg. Mega128.hex) which resides on the hardware and handles the file upgrading in conjunction with the PC.

AVR memory considerations

The boot loader system can be used on AVR chips which have the required memory arrangements. Suitable parts (eg. ATMEGA8, ATMEGA16, ATMEGA128) have the flash memory split into a Boot section and an Application section. The boot code is programmed into the boot section using a programmer in the normal way (eg. I use an ATAVRISP programmer and AVR Studio front end) and the application program is loaded into the application area by the boot loader. The advantage of this system is that hardware can be programmed with the boot loader at manufacture and after that time, firmware can be added and changed simply using a PC running MegaBootLoad.

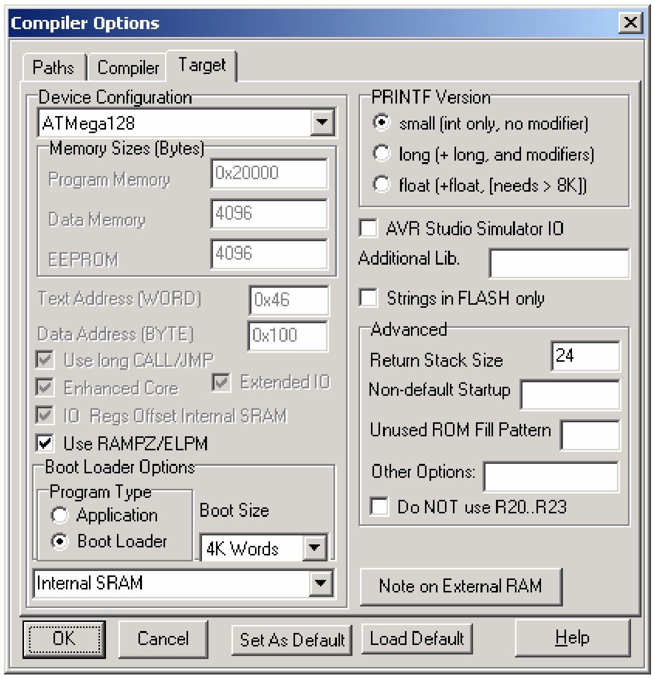
My development tools

I have been using the following tools to work with Megaload;
Imagecraft C compiler version 6.27
AVR Studio version 4.07
Atmel ATVRISP programmer
Megaload 1.1
Assorted AVR hardware all based on ATMEGA128 chip (actually use Kanda Meg Dev boards to avoid surface mounting problems).

A few notes on setting fuses and memory sizes

The setting of fuses and handling of memory allocations on AVRs is rather complex and I havn't really gotten to grips with it all yet - too much 'C' programming in recent years and loosing touch with the hardware I'm afraid!

A few things to note;
The boot code is very compact and will fit into around 500 bytes I believe. As I am not short of memory on the Mega128 I have allocated the full boot sector of 4k in my applications. This means I can extend the loader later without any real problems or memory allocation changes.
When I was working on the boot code I selected a boot size of 4k and compiled the program as a boot application;



Compiler Options

Paths | Compiler | Target

Device Configuration: ATMega128

Memory Sizes (Bytes)
- Program Memory: 0x20000
- Data Memory: 4096
- EEPROM: 4096

Text Address (WORD): 0x46
Data Address (BYTE): 0x100

- [✓] Use long CALL/JMP
- [✓] Enhanced Core
- [✓] Extended IO
- [✓] IO Regs Offset Internal SRAM
- [✓] Use RAMPZ/ELPM

Boot Loader Options
- Program Type: Application [✓ Boot Loader]
- Boot Size: 4K Words

Internal SRAM

PRINTF Version: [✓ small (int only, no modifier)] long (+ long, and modifiers) float (+float, [needs > 8K])

- [ ] AVR Studio Simulator IO

Additional Lib.:

- [ ] Strings in FLASH only

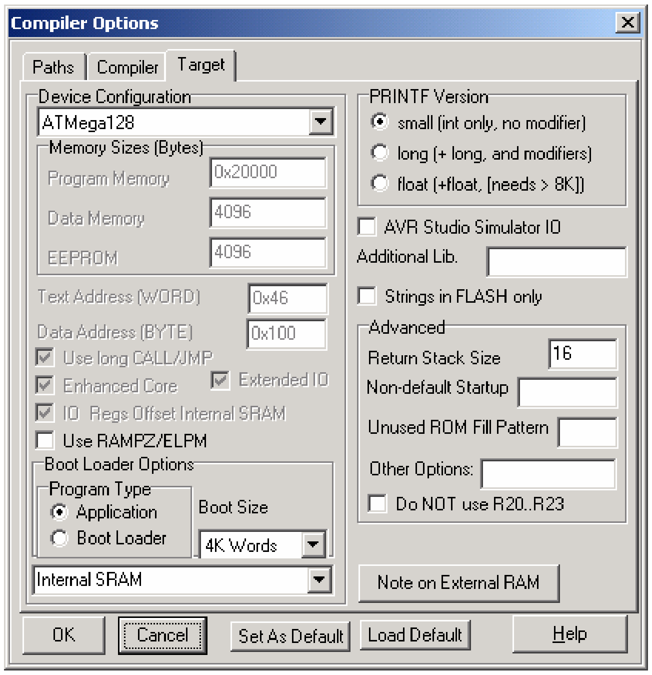
Advanced
- Return Stack Size: 24
- Non-default Startup:
- Unused ROM Fill Pattern:
- Other Options:
- [ ] Do NOT use R20..R23

Note on External RAM

OK | Cancel | Set As Default | Load Default | Help

This screen dump shows my settings in ICC for the boot compile.

When compiling my applications I use the following settings;



Compiler Options

Paths | Compiler | Target

Device Configuration: ATMega128

Memory Sizes (Bytes)
- Program Memory: 0x20000
- Data Memory: 4096
- EEPROM: 4096

Text Address (WORD): 0x46
Data Address (BYTE): 0x100

- [✓] Use long CALL/JMP
- [✓] Enhanced Core
- [✓] Extended IO
- [✓] IO Regs Offset Internal SRAM
- [ ] Use RAMPZ/ELPM

Boot Loader Options
- Program Type: [✓ Application] Boot Loader
- Boot Size: 4K Words

Internal SRAM

PRINTF Version: [✓ small (int only, no modifier)] long (+ long, and modifiers) float (+float, [needs > 8K])

- [ ] AVR Studio Simulator IO

Additional Lib.:

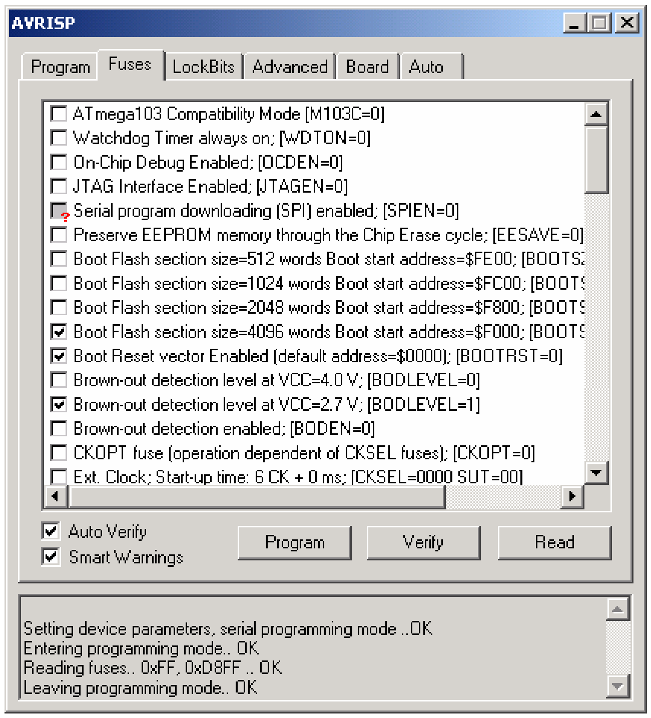
- [ ] Strings in FLASH only

Advanced
- Return Stack Size: 16
- Non-default Startup:
- Unused ROM Fill Pattern:
- Other Options:
- [ ] Do NOT use R20..R23

Note on External RAM

OK | Cancel | Set As Default | Load Default | Help

When programming the boot code into my hardware with the ATAVRISP I use the following fuse settings;



- ☐ ATmega103 Compatibility Mode [M103C=0]
- ☐ Watchdog Timer always on; [WDTON=0]
- ☐ On-Chip Debug Enabled; [OCDEN=0]
- ☐ JTAG Interface Enabled; [JTAGEN=0]
- ☑ Serial program downloading (SPI) enabled; [SPIEN=0]
- ☐ Preserve EEPROM memory through the Chip Erase cycle; [EESAVE=0]
- ☐ Boot Flash section size=512 words Boot start address=$FE00; [BOOTS
- ☐ Boot Flash section size=1024 words Boot start address=$FC00; [BOOTS
- ☐ Boot Flash section size=2048 words Boot start address=$F800; [BOOTS
- ☑ Boot Flash section size=4096 words Boot start address=$F000; [BOOTS
- ☑ Boot Reset vector Enabled (default address=$0000); [BOOTRST=0]
- ☐ Brown-out detection level at VCC=4.0 V; [BODLEVEL=0]
- ☑ Brown-out detection level at VCC=2.7 V; [BODLEVEL=1]
- ☐ Brown-out detection enabled; [BODEN=0]
- ☐ CKOPT fuse (operation dependent of CKSEL fuses); [CKOPT=0]
- ☐ Ext. Clock; Start-up time: 6 CK + 0 ms; [CKSEL=0000 SUT=00]

☑ Auto Verify
☑ Smart Warnings

```
Setting device parameters, serial programming mode ..OK
Entering programming mode.. OK
Reading fuses.. 0xFF, 0xD8FF .. OK
Leaving programming mode.. OK
```

Note that the Boot Reset vector is enabled which forces the AVR to run the boot program on reset. If the MegaBootLoad program is connected at reset then a code upgrade can be done otherwise the unit will run the current application program.

One point to bear in mind;
The latest boot code initially runs a loop trying different UART speed settings in order to find the PC application running at 38k Baud. This means that after a normal reset the boot program takes some time to run before 'dropping-through' to the main application. During that time lots of data will come out of the serial port at all sorts of speeds and any connected devices could be upset by data, parity and framing errors.

Disclaimer
This note was produced by Tony Pattison (tony@appliedlaser.co.uk) as a simple introduction to boot loaders on AVRs. It is not an exhaustive treatise and only outlines things I have found over the last few days whilst experimenting with Megaload.
I cannot accept any responsibility for errors or inaccuracies contained in the above - things may change and I may be doing it all wrong!

Tony Pattison 11/4/03
Version 1.00 11/4/03