User guide for an Atmega 328 bootloader

Introduction

This C bootloader program has been developed from an earlier in-Circuit programmer published by osbornema in "https://community.atmel.com/projects" on 22 February 2019.

The Hard ware routines required to clock data and commands between programmer and target have been replaced by assembly routines designed to implement the LPM (load program memory) and SPM (store program memory) commands. Prototypes for these subroutines are given in the Atmega 328 data sheet.

Two .zip files are posted alongside this user guide:

The bootloader.zip file contains all the source files including the make file required to build the complete code.

The application.zip file contains the source files required to build several applications based on maths and I/O library routines.

The only peripherals required are:

A USB bridge between the UART port of the Atmega 328 and a PC

Connection of an external reset circuit to PC6 (the reset pin) such as a 10K resistor to Vcc and a push button switch to 0V.

The bootloader does in fact assume the presence of a LED between PB0 and 0V but this can be omitted.

The applications are offered as test files for the bootloader and require no additional peripherals.
The remaining contents of this document fall under the following headings:

Configuration bytes
Makefile
Compilation messages
The .hex file
Bootloader operation
Applications
Downloading applications
EEPROM reservations
The test applications

Configuration bytes

The Atmega 328 bootloader runs under configuration bytes:

Extended	0xFD	This sets the BOD which is non critical to this application
High	0xD0	This configures the Reset pin, WDT, Boot size and reset vector all of which are critical
Fuse	C2	This configures the internal RC clock and SUT. An 8MHz clock is assumed but its source is not critical
Lock	EF	To protect the boot space
	EB	To protect both boot and application regions

The make file

Additional lines are required to:

Link in an assembly file

Ensure that the code will be loaded into memory at address 7000 (the start of the boot section)

These lines are:

ASRC = Resources\Atmega328bootloader.S

and

BOOTSTART = 0x7000

LDFLAGS += -Wl,--section-start=.text=$(BOOTSTART)

Compilation messages

Only compile the bootloader using optimisation level 's'.

The compiler produces the following warning message twice:

"assignment makes integer from pointer without a caste"

In each case this is because of the way in which pointers are passed to the assembly file, which would not be appropriate in an application that consisted entirely of C modules.

The .hex file

The resulting hex file appears to start at address 0x7000 however the last but one line in the hex file will be:

:040000030000700089

This is equivalent to the assembly statement ("jmp 0x7000"); and is loaded at location zero. It is not necessary because the configuration bytes have been chosen to ensure that the processor automatically resets to location 0x7000. It will normally be overwritten by the application but if for any reason it is not then it should be deleted from the hex file.

The bootloader operation

All resets take program control to line 0x7000, the start of the bootloader program. Having configured the hardware (i.e. UART, WDT etc.) the bootloader checks the source of the reset.

For power on, watch dog and brown-out resets:

The interrupt vector table at the start of the application section is selected

A "jmp 0x0000" is executed.

Application (code if present) runs

In the absence of an application the user escapes by operating the PB switch to generate another external reset.

For an external reset

The interrupt vector table at the start of the boot section is selected

The "External Reset Flag" is cleared.

The program generates a "p/r p/r p/r....." user prompt.

For keypress 'p' the bootloader checks the lock byte setting

If it is set the message "Device locked." is printed

If it is not set a new hex file is requested.

At the end of the download the file sizes are verified and the user is offered the option to set the lock byte.

The user prompt is repeated

For keypress 'r' a WD timeout is triggered and control handed to the application section.

Note: The lock byte can be cleared if necessary by reprogramming the bootloader.

The application

Two points should be noted:
The reset pin will always trigger the bootloader.
The configuration bytes can be changed but the following parameters should be maintained:

Ext reset pin enabled
WDT under program control
Resets to 0x7000 (start of bootloader)
4kByte boot partition (max allowed)

Otherwise the application SW should be unaffected by the presence of the bootloader.

Downloading applications

The bootloader downloads applications via a terminal program such as Tera Term or Br@y++. The baud rate should be set to 57600. Other settings are 8 data bits, no parity,1 stop bit and no handshaking.

If the size of the hex file is more than 50% that of the application section it may be necessary to split the hex file into two files of approximately equal size. Having initiated the download of the first half wait for the download to stop (i.e. the led if implemented stops flashing).
Then send the second file.

EEPROM reservations

Generally speaking the accuracy of the internal RC clock is good enough to enable communication with a PC at 57.6k. Occasionally however this is not so and users can set their own calibration value in place of OSCCAL the default value.
In projects posted by osbornema the following EEPROM reservations are made:

0x3FF user cal if set
0x3FE user cal if set
0x3FD Default cal supplied by Atmel

During HW configuration the bootloader checks locations 0x3FE and 0x3FF. If they both hold the same value and it is between 0x0F and 0xF0 the bootloader assumes that it is the user calibration byte and copies it to the OSCCAL register.

For most devices user calibration of the RC clock can be ignored. For those interested in its operation the following posting was made by osbornema on Jan 10, 2019.

Using the Atmega 328 internal RC clock.

The test applications

These involve keying in and printing out scientific numbers and doing arithmetic on them.
Note that the makefile has been modified for these applications. The following lines have been added/modified

```
PRINTF_LIB = $(PRINTF_LIB_FLOAT)
SCANF_LIB = $(SCANF_LIB_FLOAT)
```

The following line has also been added above the main file: #include <math.h>