AttoBASIC Version 2.3x
Device Programming Instructions

Introduction

AttoBASIC Version 2.3x, hereinafter referred to as *AttoBASIC*, supports most flavors of the ATMEL 8-bit series of AVR microcontrollers having at least 8K FLASH memory, 1K SRAM and 512 bytes of EEPROM.

AttoBASIC comes with pre-assembled HEX files for the ATmega88/P/A/PA, ATmega168/P/A/PA, ATmega328/P/A/PA, ATmega32U4 and AT90USB1286 microcontrollers at clock speeds of 4 MHz, 8 MHz, 16 MHz and 20 MHz. Other clock speeds can be built from the source code and if one has the knowledge to do so, additional commands and hardware can be added.

The only difference in the immediately supported µC's, besides the amount of memory, is that the ATmega32U4 and AT90USB1286 have a built-in USB hardware controller. Thus, the ATmega32U4 and AT90USB1286 builds can be communicated with via the USART or the USB emulating a *Virtual Com Port* (VCP). The free WINDOWS® drivers are supplied with AttoBASICv2.3x in the folder entitled "*_USB_Drivers*". Linux natively supports the VCP interface as either */dev/ttyACMx* or */dev/ttyUSBx* devices.

Most ARDUINO™ flavors and their clones use the ATmega168(P) or ATmega328(P). Newer ARDUINO™ flavors use the Atmega32U4. The PJRC Teensy 2.0++ uses the AT90USB1286. The ATmega88/168/328 µC's use the USART to communicate through. Hardware platforms using those µC's with a standard RS-232 interface are directly supported. However, the ARDUINO™-type hardware platforms communicate through a Virtual Serial Port using a separate on-board *USB to Serial converter* to make the translation to USB. This poses no problem for AttoBASIC because the hardware is the same as a non-USB platform and no modifications to software or hardware are required.

AttoBASIC also includes pre-assembled HEX files for the aforementioned microcontrollers and clock speeds containing the "*OptiBoot*" boot-loader, which is supported by the programming utility *avrdude* and the ARDUINO™ development environment.

Firmware Flavor Files

AttoBASICv2.3x comes with the following pre-assembled HEX files, which are stored in the *AVR_Specific_Builds* folder. There are versions with and without boot-loader support as well as with and without USB serial I/O support for the ATmega32U4 and AT90USB1286. The suffixes are self-identifying. The prefix "ATTOBASICV230" indicates the software version and may be different than those shown below.

ATmega88(P)
ATTOBASICV230_M88-20MHZ-uart.hex
ATTOBASICV230_M88-16MHZ-uart.hex
ATTOBASICV230_M88-8MHZ-uart.hex
ATTOBASICV230_M88-4MHZ-uart.hex

Atmega2560
ATTOBASICV230_m2560-16MHZ-uart_btldr.hex
ATTOBASICV230_m2560-16MHZ-uart_nobtldr.hex
ATTOBASICV230_m2560-20MHZ-uart_btldr.hex
ATTOBASICV230_m2560-20MHZ-uart_nobtldr.hex
ATTOBASICV230_m2560-4MHZ-uart_btldr.hex
ATTOBASICV230_m2560-4MHZ-uart_nobtldr.hex
ATTOBASICV230_m2560-8MHZ-uart_btldr.hex
ATTOBASICV230_m2560-8MHZ-uart_nobtldr.hex

ATmega168(P)
ATTOBASICV230_M168-20MHZ-uart_btldr.hex
ATTOBASICV230_M168-20MHZ-uart_nobtldr.hex
ATTOBASICV230_M168-16MHZ-uart_btldr.hex
ATTOBASICV230_M168-16MHZ-uart_nobtldr.hex
ATTOBASICV230_M168-8MHZ-uart_btldr.hex
ATTOBASICV230_M168-8MHZ-uart_nobtldr.hex
ATTOBASICV230_M168-4MHZ-uart_btldr.hex
ATTOBASICV230_M168-4MHZ-uart_nobtldr.hex

ATmega328(P)
ATTOBASICV230_M328-20MHZ-uart_btldr.hex
ATTOBASICV230_M328-20MHZ-uart_nobtldr.hex
ATTOBASICV230_M328-16MHZ-uart_btldr.hex
ATTOBASICV230_M328-16MHZ-uart_nobtldr.hex
ATTOBASICV230_M328-8MHZ-uart_btldr.hex
ATTOBASICV230_M328-8MHZ-uart_nobtldr.hex
ATTOBASICV230_M328-4MHZ-uart_btldr.hex
ATTOBASICV230_M328-4MHZ-uart_nobtldr.hex

AT90USB1286

ATTOBASICV230_usb1286-16MHZ-teensypp20.hex
ATTOBASICV230_usb1286-16MHZ-uart_btldr.hex
ATTOBASICV230_usb1286-16MHZ-uart_nobtldr.hex
ATTOBASICV230_usb1286-16MHZ-usb_btldr.hex
ATTOBASICV230_usb1286-16MHZ-usb_nobtldr.hex
ATTOBASICV230_usb1286-20MHZ-uart_btldr.hex
ATTOBASICV230_usb1286-20MHZ-uart_nobtldr.hex
ATTOBASICV230_usb1286-4MHZ-uart_btldr.hex
ATTOBASICV230_usb1286-4MHZ-uart_nobtldr.hex
ATTOBASICV230_usb1286-8MHZ-uart_btldr.hex
ATTOBASICV230_usb1286-8MHZ-uart_nobtldr.hex
ATTOBASICV230_usb1286-8MHZ-usb_btldr.hex
ATTOBASICV230_usb1286-8MHZ-usb_nobtldr.hex

ATmega32U4

ATTOBASICV230_M32u4-20MHZ-uart_btldr.hex
ATTOBASICV230_M32u4-20MHZ-uart_nobtldr.hex
ATTOBASICV230_M32u4-16MHZ-uart_btldr.hex
ATTOBASICV230_M32u4-16MHZ-uart_nobtldr.hex
ATTOBASICV230_M32u4-16MHZ-usb_btldr.hex
ATTOBASICV230_M32u4-16MHZ-usb_nobtldr.hex
ATTOBASICV230_M32u4-8MHZ-uart_btldr.hex
ATTOBASICV230_M32u4-8MHZ-uart_nobtldr.hex
ATTOBASICV230_M32u4-8MHZ-usb_btldr.hex
ATTOBASICV230_M32u4-8MHZ-usb_nobtldr.hex
ATTOBASICV230_M32u4-4MHZ-uart_btldr.hex
ATTOBASICV230_M32u4-4MHZ-uart_nobtldr.hex

Loading the firmware into a specific hardware platform

For most hardware platforms, using the *In System Programming* (ISP) feature of the AVR µC's is the preferred method. Choose a file and clock speed that is compatible with the µC on the target platform. Keep in mind that all "factory fresh" AVR's come with the fuse setting that enables the on-chip oscillator and *divide by 8 prescaler* so the µC runs at 1 MHz. One will need to insure that the programmer's ISP clock speed is ¼ of the µC's clock and likely wish to set the fuses to enable an external crystal and disable the *divide by 8 prescaler.* Setting the AVR's fuses is beyond the scope of this writing. Refer to target µC's datasheet and programmer's documentation for further information.

ARDUINO™: For those having an ARDUINO™ compatible platform available, the firmware files with the "nobtldr" suffixes, for either ATmega88/168/328 or Atmega2560, can be directly uploaded using the *avrdude* utility, which is available as part of the *avr-gcc* software package under Linux or the *WinAVR* software package under WINDOWS®.

If using the WINDOWS® OS, open a *CMD* window, traverse to the appropriate folder containing the desired HEX file and issue one of the two following commands (boot-loader dependant):

```
avrdude.exe -V -F -p atmega328p -c arduino -P COMx -b 115200 -U flash:w:myprogram.hex
avrdude.exe -V -F -p atmega328p -c stk500v1 -P COMx -b 57600 -U flash:w:myprogram.hex
```

Replace "`atmega328p`" with the target µC, "`myprogram.hex`" with the desired firmware filename and "`COMx`" with the actual serial port name your ARDUINO™ responds on.

If using the Linux OS (or Mac OSX?), open a terminal window, traverse to the appropriate path containing the desired HEX file and issue one of the two following commands (boot-loader dependant):

```
avrdude -V -F -p atmega328p -c arduino -P /dev/ttyACMx -b 115200 -U flash:w:myprogram.hex
avrdude -V -F -p atmega328p -c stk500v1 -P /dev/ttyACMx -b 57600 -U flash:w:myprogram.hex
```

Replace "`atmega328p`" with the target µC, "`myprogram.hex`" with the desired firmware filename and "`/dev/ttyACMx`" with the actual serial port name your ARDUINO™ responds on (usually */dev/ttyACMx* or */dev/ttyUSBx* where *x* is a number between 0 and 9). For an ARDUINO™ *Mega 2560*, replace "`atmega328p`" with "`atmega2560`" and "`arduino`" with "`stk500v2`" and leave the "`-b 115200`" off.

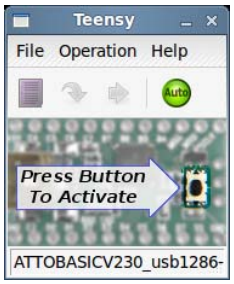
The author has written a BASH shell script for use under Linux that simplifies uploading HEX file to ARDUINO™ platforms. The shell script, named `mk_prog-duino.sh`, is located in the root project directory and uses the *avrdude* utility. It allows one to select the target processor's type then the HEX file to be uploaded to the target platform and is invoked with *./mk_prog-duino.sh*.

Notes:

1. If *avrdude* responds with a "**stk500_getsync(): not in sync**" message, it is because the ARDUINO™ boot-loader is not responding. Check the command line for correctness. If *avrdude*

continues to error with the same message, substitute either "115200", "57600", "19200" or "9600" as the baud rate in the "-b NNNNN" option and/or power-cycle the ARDUINO™. If the problem persists, consult other resources for remedy.

2. For convenience, residing in the root project folder, there is a Linux shell script named `mk_prog-duino.sh`, which will allow one to select the target MCU, a HEX file for that MCU and a programming port (i.e. /dev/ttyUSB0) to program an ARDUINO™. The shell script makes use of *avrdude*.
3. Do not attempt to use the ARDUINO™ boot-loader to upload a firmware file that contains a boot-loader image as this may corrupt the existing boot-loader and render it inoperable. Those particular firmware files are meant to be programmed with an ISP programmer and is the only way to recover from a corrupted boot-loader.
4. AttoBASIC contains the *BLDR* command to invoke the resident boot-loader if one exists. However, if the *BOOTRST* fuse is programmed, the resident boot-loader will automatically be invoked. On ARDUINO™ platforms, the resident boot-loader will start the application program after a short delay. Using AttoBASIC to invoke a boot-loader that uses the serial port may emit non-printable characters that confuse the terminal emulator program. This will likely require a restart of the terminal emulator program to recover. One's particular boot-loader's behavior may vary …
5. AttoBASIC is using "*OptiBoot*" for its boot-loader support on Mega168/328 devices. *Optiboot* uses *avrdude's* "arduino" protocol, which is specified as "-c arduino" on the command line.
    a. for the ATmega168(P), the boot-loader resides at *0x1F00* and the *BOOTRST* fuse must be programmed.
    b. for the ATmega328(P), the boot-loader resides at *0x3F00* and the *BOOTRST* fuse must be programmed.
6. AttoBASIC is using "stk500v2" for its boot-loader support on Mega2560 devices. *Stk500v2* uses *avrdude's* "avrispmkii" protocol, which is specified as "-c avrispmkii" on the command line. The boot-loader resides at *0x1F000* and the *BOOTRST* fuse must be programmed.
7. The "Self-Start" feature uses PINC3 on the ATmega88/168/328 firmware images. If that feature is not used, the I/O pin can be used as an I/O port pin without interference from AttoBASIC and does not require an external pull-up resistor.



TEENSY++ 2.0™ using Teensy Loader application: For those having a TEENSY++ 2.0™ platform available, the PJRC "Teensy Loader" application can be used. The application supports both WINDOWS® and most Linux platforms. Be sure to use the specific AttoBASIC build compiled especially for the TEENSY++ 2.0.

Generic ATmega32U4 and AT90USB1286: For those having a ATmega32U4 or AT90USB1286 compatible platform available (like the ADAFRUIT *Mega32U4 Breakout Board* or the PJRC *Teensy 2.0++* product without the HALKAY bootloader), the ATmega32U4/AT90USB1286 firmware files with the "nobtldr" suffixes can be programmed into the target µC using that manufacturer's programming tool or the firmware files having the DFU bootloader integrated can be programmed into the target platform using an ISP or JTAG programmer.

Integrated DFU Bootloader Support

The boot-loader incorporated into AttoBASICv2.3x for the ATmega32U4/AT90USB1286 is Dean Camera's LUFA in DFU mode, which uses the USB port. Therefore ATMEL's FLIP programming tool can be used, which is available for the WINDOWS® and Linux computing platforms.

Of course, using ISP via the JTAG or ISP interfaces works well if one has the programmer to support this manner of programming. In which case, it is recommended to use a firmware file containing the DFU Bootloader to ease in future programming.

Once the DFU Bootloader is programmed into the target device, the BOOTRST fuse must be programmed to "0" and the BOOTSZ[1:0] fuses must be programmed appropriately (see notes below). Doing so enables the DFU bootloader to be invoked after a hardware reset. Once a hardware reset has occurred, the DFU Bootloader will be invoked, at which time it will check the state of PORTF4 (JTAG TCK) looking for a logic low on that pin. If PORTF4 is low then the DFU Bootloader will initialize itself as a USB device and await a connection from the host's programming tool, otherwise, it will continue to invoke AttoBASIC.

Notes:

1. On the USB enabled version of the ATmega32U4/AT90USB1286 firmware, once a connection from the host's terminal emulator is achieved, AttoBASIC will respond with its sign-on message.
2. The DFU Bootloader supports USB only.
3. For the ATmega32U4 with DFU boot-loader support, the BOOTSZ[1:0] fuses must be programmed to "00" for the boot-loader to reside at *0x3800* and the *BOOTRST* fuse must be programmed and the BOOTRST fuse must be programmed to "0".
4. For the AT90USB1286 with DFU boot-loader support, the BOOTSZ[1:0] fuses must be programmed to "00" for the boot-loader to reside at *0xF000* and the *BOOTRST* fuse must be programmed and the BOOTRST fuse must be programmed to "0".
5. Another method of invoking the DFU Bootloader is by using AttoBASIC's "BLD" command. Once the command is issued, the target MCU will receive a hardware reset after 8 seconds from the on-board watchdog timer in *system reset* mode. As described above, PORTF4 must be held low to start the DFU Bootloader.
6. The "Self-Start" feature uses PIND7 on the ATmega32U4/AT90USB1286 firmware images. If that feature is not used, the I/O pin can be used as an I/O port pin without interference from AttoBASIC and does not require an external pull-up resistor. Be sure the pin is not held low at startup.