AttoBASIC Version 2.20
Device Programming Instructions

Introduction

AttoBASIC Version 2.20, 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 and ATmega32U4 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 has a built-in USB hardware controller. Thus, the ATmega32U4 builds can be communicated with via the USART or the USB emulating a *Virtual Com Port* (VCP). The free WINDOWS® drivers are supplied with AttoBASICv2.20 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). 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

AttoBASICv2.20 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. The suffixes are self-identifying.

ATmega88(P)

```
ATTOBASICV220_M88-20MHZ-uart.hex
ATTOBASICV220_M88-16MHZ-uart.hex
ATTOBASICV220_M88-8MHZ-uart.hex
ATTOBASICV220_M88-4MHZ-uart.hex
```

ATmega168(P)

```
ATTOBASICV220_M168-20MHZ-uart_btldr.hex
ATTOBASICV220_M168-20MHZ-uart_nobtldr.hex
ATTOBASICV220_M168-16MHZ-uart_btldr.hex
ATTOBASICV220_M168-16MHZ-uart_nobtldr.hex
ATTOBASICV220_M168-8MHZ-uart_btldr.hex
ATTOBASICV220_M168-8MHZ-uart_nobtldr.hex
ATTOBASICV220_M168-4MHZ-uart_btldr.hex
ATTOBASICV220_M168-4MHZ-uart_nobtldr.hex
```

ATmega328(P)

```
ATTOBASICV220_M328-20MHZ-uart_btldr.hex
ATTOBASICV220_M328-20MHZ-uart_nobtldr.hex
ATTOBASICV220_M328-16MHZ-uart_btldr.hex
ATTOBASICV220_M328-16MHZ-uart_nobtldr.hex
ATTOBASICV220_M328-8MHZ-uart_btldr.hex
ATTOBASICV220_M328-8MHZ-uart_nobtldr.hex
ATTOBASICV220_M328-4MHZ-uart_btldr.hex
ATTOBASICV220_M328-4MHZ-uart_nobtldr.hex
```

ATmega32U4

```
ATTOBASICV220_M32u4-20MHZ-uart_btldr.hex
ATTOBASICV220_M32u4-20MHZ-uart_nobtldr.hex
ATTOBASICV220_M32u4-16MHZ-uart_btldr.hex
ATTOBASICV220_M32u4-16MHZ-uart_nobtldr.hex
ATTOBASICV220_M32u4-16MHZ-usb_btldr.hex
ATTOBASICV220_M32u4-16MHZ-usb_nobtldr.hex
ATTOBASICV220_M32u4-8MHZ-uart_btldr.hex
ATTOBASICV220_M32u4-8MHZ-uart_nobtldr.hex
ATTOBASICV220_M32u4-8MHZ-usb_btldr.hex
ATTOBASICV220_M32u4-8MHZ-usb_nobtldr.hex
ATTOBASICV220_M32u4-4MHZ-uart_btldr.hex
ATTOBASICV220_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 ATmega88/168/328 firmware files with the "nobtldr" suffixes 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
```

Replacing "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).

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. 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.
3. 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 …

4. AttoBASIC is using "*OptiBoot*" for its boot-loader support. The *avrdude* command line that supports *Optiboot* contains "`-c arduino`".
    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.
5. 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.

ATmega32U4: For those having a ATmega32U4 compatible platform available (like the ADAFRUIT *Mega32U4 Breakout Board*), the ATmega32U4 firmware files with the "nobtldr" and/or "nousb" suffixes can be programmed into the target μC using that manufacturer's programming tool. The boot-loader incorporated into AttoBASICv2.20 for the ATmega32U4 is Dean Camera's LUFA in DFU mode. Therefore ATMEL's FLIP programming tool can be used, which is available for the WINDOWS® and Linux computing platforms.

Notes:

1. On the USB enabled version of the ATmega32U4 firmware, one must type a key on the terminal emulator once a connection to the ATmega32U4 is achieved. AttoBASIC will then respond with its sign-on message.
2. For the ATmega32U4 with boot-loader support, the boot-loader resides at *0x3800* and the *BOOTRST* fuse must be programmed.
3. The "Self-Start" feature uses PIND7 on the ATmega32U4 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.