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, 512 bytes of SRAM and 512 bytes of EEPROM.

Current version of AttoBASIC supports and provides pre-assembled HEX files for the ATtiny84, ATtiny85, ATmega88/P/A/PA, ATmega168/P/A/PA, ATmega328/P/A/PA, ATmega16A, ATmega32A, ATmega644P, ATmega1284P, ATmega2560, 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 (refer to application note) and if one has the knowledge to do so, additional commands and hardware can be added.

AttoBASIC also supports and provides pre-assembled HEX files for the AT90S2313, ATtiny2313(A), AT908515 and ATmegaq163 microcontrollers at clock speeds of 4 MHz, 8 MHz, 16 MHz and 20 MHz (Other clock speeds can be built from the source code). However, support for these microcontrollers is limited to the circa-2002 version of AttoBASIC.

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 and Atmega2560. The PJRC Teensy 2.0++ uses the AT90USB1286.

The ATmega88/168/328, ATmega16/32, ATmega644/1284 and Atmega2560 µC's use the USART to communicate through, while the ATtiny84 and ATtiny85 uses a software-emulated UART. 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* (FT232, CH340, CP2102, etc) 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*", "*stk500v2*" or "*OptiBoot85*" 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. In the case of the ATmega32U4 and AT90USB1286, two (2) boot loaders are supported, LUFA CDC and LUFA DFU. The suffixes are self-identifying. The prefix "ATTOBASICV233" indicates the software version and may be different than those shown below.

ATtiny84
ATTOBASICV234_tn85-4MHZ-uart_btldr.hex
ATTOBASICV234_tn85-4MHZ-uart_nobtldr.hex
ATTOBASICV234_tn85-8MHZ-uart_btldr.hex
ATTOBASICV234_tn85-8MHZ-uart_nobtldr.hex
ATTOBASICV234_tn85-16MHZ-uart_btldr.hex
ATTOBASICV234_tn85-16MHZ-uart_nobtldr.hex
ATTOBASICV234_tn85-20MHZ-uart_btldr.hex
ATTOBASICV234_tn85-20MHZ-uart_nobtldr.hex

ATtiny85
ATTOBASICV234_tn85-4MHZ-uart_btldr.hex
ATTOBASICV234_tn85-4MHZ-uart_nobtldr.hex
ATTOBASICV234_tn85-8MHZ-uart_btldr.hex
ATTOBASICV234_tn85-8MHZ-uart_nobtldr.hex
ATTOBASICV234_tn85-16MHZ-uart_btldr.hex
ATTOBASICV234_tn85-16MHZ-uart_nobtldr.hex
ATTOBASICV234_tn85-20MHZ-uart_btldr.hex
ATTOBASICV234_tn85-20MHZ-uart_nobtldr.hex

ATmega644(P)
ATTOBASICV234_m644p-16MHZ-uart_btldr.hex
ATTOBASICV234_m644p-16MHZ-uart_nobtldr.hex
ATTOBASICV234_m644p-20MHZ-uart_btldr.hex
ATTOBASICV234_m644p-20MHZ-uart_nobtldr.hex
ATTOBASICV234_m644p-4MHZ-uart_btldr.hex
ATTOBASICV234_m644p-4MHZ-uart_nobtldr.hex
ATTOBASICV234_m644p-8MHZ-uart_btldr.hex
ATTOBASICV234_m644p-8MHZ-uart_nobtldr.hex

ATmega1284(P)
ATTOBASICV234_m1284p-16MHZ-uart_btldr.hex
ATTOBASICV234_m1284p-16MHZ-uart_nobtldr.hex
ATTOBASICV234_m1284p-20MHZ-uart_btldr.hex
ATTOBASICV234_m1284p-20MHZ-uart_nobtldr.hex
ATTOBASICV234_m1284p-4MHZ-uart_btldr.hex
ATTOBASICV234_m1284p-4MHZ-uart_nobtldr.hex
ATTOBASICV234_m1284p-8MHZ-uart_btldr.hex
ATTOBASICV234_m1284p-8MHZ-uart_nobtldr.hex

ATMega16A
ATTOBASICV234_m16a-16MHZ-uart_btldr.hex
ATTOBASICV234_m16a-16MHZ-uart_nobtldr.hex
ATTOBASICV234_m16a-20MHZ-uart_btldr.hex
ATTOBASICV234_m13a-20MHZ-uart_nobtldr.hex
ATTOBASICV234_m16a-4MHZ-uart_btldr.hex
ATTOBASICV234_m16a-4MHZ-uart_nobtldr.hex
ATTOBASICV234_m16a-8MHZ-uart_btldr.hex
ATTOBASICV234_m16a-8MHZ-uart_nobtldr.hex

ATMega32A
ATTOBASICV234_m32a-16MHZ-uart_btldr.hex
ATTOBASICV234_m32a-16MHZ-uart_nobtldr.hex
ATTOBASICV234_m32a-20MHZ-uart_btldr.hex
ATTOBASICV234_m32a-20MHZ-uart_nobtldr.hex
ATTOBASICV234_m32a-4MHZ-uart_btldr.hex
ATTOBASICV234_m32a-4MHZ-uart_nobtldr.hex
ATTOBASICV234_m32a-8MHZ-uart_btldr.hex
ATTOBASICV234_m32a-8MHZ-uart_nobtldr.hex

ATmega88(P)
ATTOBASICV234_M88-20MHZ-uart.hex
ATTOBASICV234_M88-16MHZ-uart.hex
ATTOBASICV234_M88-8MHZ-uart.hex
ATTOBASICV234_M88-4MHZ-uart.hex

ATmega168(P)
ATTOBASICV234_M168-20MHZ-uart_btldr.hex
ATTOBASICV234_M168-20MHZ-uart_nobtldr.hex
ATTOBASICV234_M168-16MHZ-uart_btldr.hex
ATTOBASICV234_M168-16MHZ-uart_nobtldr.hex
ATTOBASICV234_M168-8MHZ-uart_btldr.hex
ATTOBASICV234_M168-8MHZ-uart_nobtldr.hex
ATTOBASICV234_M168-4MHZ-uart_btldr.hex
ATTOBASICV234_M168-4MHZ-uart_nobtldr.hex

ATmega328(P)
ATTOBASICV234_M328-20MHZ-uart_btldr.hex
ATTOBASICV234_M328-20MHZ-uart_nobtldr.hex
ATTOBASICV234_M328-16MHZ-uart_btldr.hex
ATTOBASICV234_M328-16MHZ-uart_nobtldr.hex
ATTOBASICV234_M328-8MHZ-uart_btldr.hex
ATTOBASICV234_M328-8MHZ-uart_nobtldr.hex
ATTOBASICV234_M328-4MHZ-uart_btldr.hex
ATTOBASICV234_M328-4MHZ-uart_nobtldr.hex

ATmega32U4
ATTOBASICV234_m32u4-16MHZ-uart_btldr.hex
ATTOBASICV234_m32u4-16MHZ-uart_nobtldr.hex
ATTOBASICV234_m32u4-16MHZ-usb_btldcdc.hex
ATTOBASICV234_m32u4-16MHZ-usb_btlddfu.hex
ATTOBASICV234_m32u4-16MHZ-usb_nobtldr.hex
ATTOBASICV234_m32u4-20MHZ-uart_btldr.hex
ATTOBASICV234_m32u4-20MHZ-uart_nobtldr.hex
ATTOBASICV234_m32u4-4MHZ-uart_btldr.hex
ATTOBASICV234_m32u4-4MHZ-uart_nobtldr.hex
ATTOBASICV234_m32u4-8MHZ-uart_btldr.hex
ATTOBASICV234_m32u4-8MHZ-uart_nobtldr.hex
ATTOBASICV234_m32u4-8MHZ-usb_btldcdc.hex
ATTOBASICV234_m32u4-8MHZ-usb_btlddfu.hex
ATTOBASICV234_m32u4-8MHZ-usb_nobtldr.hex

AT90USB1286
ATTOBASICV234_usb1286-16MHZ-teensypp20.hex
ATTOBASICV234_usb1286-16MHZ-uart_btldr.hex
ATTOBASICV234_usb1286-16MHZ-uart_nobtldr.hex
ATTOBASICV234_usb1286-16MHZ-usb_btldcdc.hex
ATTOBASICV234_usb1286-16MHZ-usb_btlddfu.hex
ATTOBASICV234_usb1286-16MHZ-usb_nobtldr.hex
ATTOBASICV234_usb1286-20MHZ-uart_btldr.hex
ATTOBASICV234_usb1286-20MHZ-uart_nobtldr.hex
ATTOBASICV234_usb1286-4MHZ-uart_btldr.hex
ATTOBASICV234_usb1286-4MHZ-uart_nobtldr.hex
ATTOBASICV234_usb1286-8MHZ-teensypp20.hex
ATTOBASICV234_usb1286-8MHZ-uart_btldr.hex
ATTOBASICV234_usb1286-8MHZ-uart_nobtldr.hex
ATTOBASICV234_usb1286-8MHZ-usb_btldcdc.hex
ATTOBASICV234_usb1286-8MHZ-usb_btlddfu.hex
ATTOBASICV234_usb1286-8MHZ-usb_nobtldr.hex

Atmega2560
ATTOBASICV234_m2560-16MHZ-uart_btldr.hex
ATTOBASICV234_m2560-16MHZ-uart_nobtldr.hex
ATTOBASICV234_m2560-20MHZ-uart_btldr.hex
ATTOBASICV234_m2560-20MHZ-uart_nobtldr.hex
ATTOBASICV234_m2560-4MHZ-uart_btldr.hex
ATTOBASICV234_m2560-4MHZ-uart_nobtldr.hex
ATTOBASICV234_m2560-8MHZ-uart_btldr.hex
ATTOBASICV234_m2560-8MHZ-uart_nobtldr.hex

AT90S8515
ATTOBASICV234_8515-16MHZ-uart_nobtldr.hex
ATTOBASICV234_8515-20MHZ-uart_nobtldr.hex
ATTOBASICV234_8515-4MHZ-uart_nobtldr.hex
ATTOBASICV234_8515-8MHZ-uart_nobtldr.hex

AT90S2313
ATTOBASICV234_2313-16MHZ-uart_nobtldr.hex
ATTOBASICV234_2313-20MHZ-uart_nobtldr.hex
ATTOBASICV234_2313-4MHZ-uart_nobtldr.hex
ATTOBASICV234_2313-8MHZ-uart_nobtldr.hex

ATMega163
ATTOBASICV234_m163-16MHZ-uart_nobtldr.hex
ATTOBASICV234_m163-20MHZ-uart_nobtldr.hex
ATTOBASICV234_m163-4MHZ-uart_nobtldr.hex
ATTOBASICV234_m163-8MHZ-uart_nobtldr.hex

ATtiny2313
ATTOBASICV234_tn2313-16MHZ-uart_nobtldr.hex
ATTOBASICV234_tn2313-20MHZ-uart_nobtldr.hex
ATTOBASICV234_tn2313-4MHZ-uart_nobtldr.hex
ATTOBASICV234_tn2313-8MHZ-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 or (less) 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.

Author's Note: The author has written a BASH shell script for use under Linux that simplifies uploading HEX file to most of the AttoBASIC supported platforms. The shell script, named `mk_prog-avr.sh`, is located in the root project directory and uses the *avrdude* utility. It allows one to select the target processor's type, the HEX file to be uploaded to the target platform and the programming protocol (if multiple can be used). The programming utility is invoked with *./mk_prog-avr.sh*. It has not been tested with "*Cygwin*" under WINDOWS®.

ARDUINO™ and compatibles: For those having an ARDUINO™ compatible platform available, the firmware files with the "nobtldr" suffixes, for either ATmega88/168/328, ATmega644/1284, ATmega16/32 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.

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 if the "boot loader protection" fuses are not set. 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 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 (byte address 0x3E00)* and the *BOOTRST* fuse must be programmed.
    b. for the ATmega328(P), the boot-loader resides at *0x3F00 (byte address 0x7E00)* and the *BOOTRST* fuse must be programmed.
5. AttoBASIC is using "stk500v2" for its boot-loader support on Mega2560 devices. *Stk500v2* uses *avrdude's* "stk500v2" or "avrispmkii" protocol, which is specified as either "-c stk500v2 -D" or "-c avrispmkii -D" on the command line. Note that the current versions of the stk500v2, including those included with the ARDUINO IDE's do not support the "chip erase" command and will fail programming if the *avrdude* "-D" option is not specified. The boot-loader resides at *0x1F000 (0x3E000 byte address)* and the *BOOTRST* fuse must be programmed.

ATtiny84 and ATtiny85: AttoBASIC supports ATtiny85 (and ATtiny84) based platforms that do not use the Virtual USB (VUSB) stack as the ADAFRUIT *Trinket* and OLIMEX *OLEMIXINP85S/BC/ASM* products do. This is because the VUSB code consumes roughly 25% of the ATtiny85's 8KB program space and uses much of the MCU's processing power to service USB requests from the host.

Instead, the ATtiny84/ATtiny85 implementation of AttoBASIC uses a software-emulated UART (reference the application note) and a highly optimized version of the *OptiBoot* boot-loader called *OptiBoot85*. One must supply a Serial-to-USB converter (refer to application note) to achieve a connection using this method. If *OptiBoot85* is pre-loaded into the ATtiny84 and ATtiny85 then the firmware files that do not exceed 7712 bytes (ending before byte address of 0x1E20 or word address of 0x0F10) may 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®. Be sure to connect the DTR signal from the Serial-to-USB converter to the Resistor/Capacitor network on the RESET pin of the ATtiny84 or ATtiny85 as *avrdude* uses the DTR signal to invoke the *OptiBoot85* boot-loader with a hardware reset pulse.

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 attiny85 -c arduino -P COMx -b 38400 -U flash:w:myprogram.hex
```

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

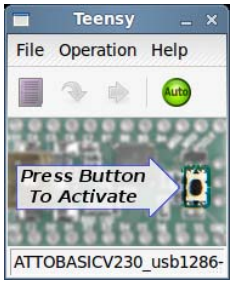
If using the Linux OS (or Mac OSX?), open a terminal window, traverse to the appropriate path containing the desired HEX file and issue one of the two following commands (boot-loader dependant):

```
avrdude -V -F -p attiny85 -c arduino -P /dev/ttyACMx -b 38400 -U flash:w:myprogram.hex
```

Replace "`myprogram.hex`" with the desired firmware filename and "`/dev/ttyACMx`" with the actual serial port name your Attiny85 platform responds on (usually */dev/ttyACMx* or */dev/ttyUSBx* where *x* is a number between 0 and 9).

The other option in programming AttoBASIC into an ATtiny85 platform is by using an ISP or High-voltage Serial (HVSP) programmer, such as the ATMEL AVR DRAGON. In the *avrdude* invocation examples above, replace the "`-c arduino`" with "`-c dragon_isp`" or "`-c dragon_hvsp`" and leave off the "`-b 38400`" parameter.

Once again, do not attempt to use the *OptiBoot85* boot-loader to upload a firmware file that contains a boot-loader image or exceeds 0x1E20 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. The *OptiBoot85* code starts at 0x1E20. If using the author's *mk_prog-avr.sh* BASH script file then any HEX files that are detected as harmful to the boot-loader are truncated to prevent this fatal mistake.



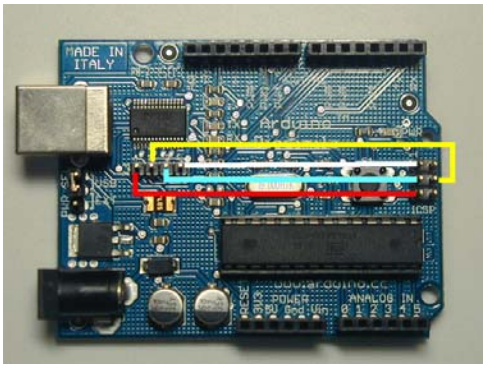
TEENSY++ 2.0™ using Teensy Loader application: For those having a TEENSY++ 2.0™ platform available, the PJRC "Teensy Loader" application can be used. The application supports both WINDOWS® and most Linux platforms. Be sure to use the specific AttoBASIC build compiled especially for the TEENSY++ 2.0.

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 or CDC bootloader integrated can be programmed into the target platform using an ISP or JTAG programmer.

Generic FT232R USB-to-Serial adapter

Note: The author has not yet tested this method of ISP programming as he does not own an FT232R-based adapter.

There appears to be a method of creating an ISP programmer using a generic FT232R USB-to-Serial adapter. However, the FT232 adapter MUST have the "RI" signal available to connect to. The pin connections for the 6-pin ISP connector are:



ISP name	ISP Pin#	FT232 Name
MISO	1	CTS
VTG	2	3V3 or 5V
SCK	3	DSR
MOSI	4	DCD
RST	5	RI
GND	6	GND

The photo shows connections on an ARDUINO Diecimila to it's own ISP connector. Using version 6.3 of *avrdude*, the desired HEX file can be programmed into the target via the ISP connector as follows:

WINDOWS: `avrdude.exe -V -F -p atmega328p -c arduino-ft232r -P ft[0..7] -U flash:w:myprogram.hex`

Linux: `avrdude -V -F -p atmega328p -c arduino-ft232r -P ft[0..7] -U flash:w:myprogram.hex`

Replace "`atmega328p`" with the target µC, "`myprogram.hex`" with the desired firmware filename.

Be aware that using the ISP programming method WILL erase the boot-loader unless the HEX file specified indicates that it has a boot-loader incorporated.

Integrated DFU or CDC Bootloader Support

The boot-loaders incorporated into AttoBASICv2.3x for the ATmega32U4/AT90USB1286 are Dean Camera's LUFA in either DFU or CDC mode. Both use the USB port. Therefore *avrdude* or ATMEL's *FLIP* programming tool can be used, both tools are available for the WINDOWS® and Linux computing platforms. For the remainder of this subsection, the term *bootloader* refers to either the DFU or CDC bootloader.

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 either the DFU or CDC Bootloader to ease in future programming.

Once the 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 bootloader to be invoked after a hardware reset. Once a hardware reset has occurred, the 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 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 and CDC Bootloaders support USB only.
3. For the ATmega32U4 with DFU or CDC 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 or CDC 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 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 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.