

Readme File: Version 1.1 –September 21, 1999

1 Overview

This CodeKit software contains the USB port expander firmware and host driver software. The combined package can be run on any Windows® 98 computer to create additional serial ports. The Am186™CC communications controller is connected to the Windows 98 computer by a USB cable. Data is transferred between either of the Am186CC communications controller's two RS-232 serial ports and the host.

IMPORTANT: By loading this software or any portion thereof, you agree to all of the terms of the License Agreement in the file LICENSE.PDF. Do not use this software until you have carefully read and agreed to terms and conditions in the Agreement. If you do not agree to the terms of the Agreement, do not use this software or any portion thereof.

For support: e-mail epd.support@amd.com
or call 1-800-222-9323

1.1 Tool and System Requirements

The following target hardware and tools are required to build and execute this CodeKit software.

ITEM	DESCRIPTION
Target Hardware	Am186CC/CH/CU microcontroller customer development platform (CDP) Revision 1.0, 1.1, or 2.0 (the CDP's ISDN TA development module is not required)
Monitor	AMD E86MON™ software, version 3.3.2 (or newer)
Workstation Operating System	Windows 98 with USB hub
Tool Suite	Microsoft® Visual C++ V1.52 Microsoft® Visual C++ V5.0 with Service Pack 3 Microsoft® Windows 98 DDK
Compiler*	Microsoft C/C++ optimizing compiler, version 8.00c
Linker*	Microsoft segmented executable linker, version 5.60.339 Dec 5 1994

* Note the compiler and linker are part of the Visual C++ V1.52 tool suite.

1.2 Installing the CodeKit Software

This CodeKit software is delivered as a self-extracting executable file. Double-click on the EXE file and a directory called C:\AMD\CK001801 is created. The CK001801 directory contains the entire contents of the USB port expander CodeKit software, including the subdirectories named FIRM_APP, FIRM_BLD, HOST_BLD, HOST_INC, HOST_SHIM, HOST_VCOMM, and TTY.

1.3 Files and Directories

The EXE file contains the following files and subdirectories.

FILE	DESCRIPTION
README.DOC	This document in Microsoft Word 97 format
README.PDF	This document in Adobe Acrobat PDF format
FIRM_APP	Subdirectory that contains Am186CC communications controller firmware source files
AM186CC.H	Standard register defines for the Am186CC communications controller
AM186SER.C	Asynchronous serial port utilities
AM186SER.H	Header file for AM186SER.C
BQUE.C	Queue utilities for AM186SER.C
BQUE.H	Declarations for BQUE.C
BUFFER.C	Buffer management utility code
BUFFER.H	Declarations for BUFFER.C
COMPILER.H	Standard compiler-specific defines
LLINT.ASM	Low level interrupt routines for AM186SER.C
LLINT.H	Header file for NEWHEAP.ASM
NEWHEAP.C	Heap management utilities
NEWHEAP.H	Header file for NEWHEAP.C
STDINC.H	Standard typedefs and structures
USB.C	The Am186CC communications controller USB driver
USB.H	Definitions for USB.C
USBFIRM.C	Main application source code
USBFIRM.H	Definitions for USBFIRM.C
VCOMM.H	Windows header file with needed definitions
USBSER.H	Shared definitions for USBFIRM.C and host driver
FIRM_BLD	Subdirectory that contains build files for the firmware
BUILDIT.BAT	A batch file that builds the CodeKit software
CLEANIT.BAT	A batch file that cleans up intermediate files after a build
ENVIRON.BAT	A batch file that sets up the build environment
MAKEHEX.EXE	An executable file that converts an EXE program into a HEX file for downloading into the target board using the E86MON software
USBPORT.HEX	A HEX file that can be downloaded to the target and executed. This file results from executing the BUILDIT.BAT file
HOST_APP	Subdirectory that contains the host communication application
MAKEFILE.DSW	Microsoft Project Workspace for Visual C++
MAKEFILE.DSP	Microsoft Project for Visual C++

MAKEFILE	Makefile for TTY.EXE
README.TXT	Additional documentation on the communication application
RESOURCE.H	Resource header for TTY2.C
TTY.DEF	Definitions file for TTY2.C
TTY.EXE	Host communication application. This a simple two way communication application which connects to COM port 12
TTY.H	Header file for TTY2.C
TTY2.C	Application code
TTY.RC	Icon file for TTY application
VERSION.H	Version header file for TTY2.C
WIN32.MAK	Windows standard makefile
HOST_INC	Subdirectory that contains shared include files for the host driver
NTDDSER.H	IOCTL definitions used by the WDM and VXD drivers
HOST_SHIM	Subdirectory that contains the WDM driver source code for the host
MAKEFILE	Windows DDK Makefile for this directory
SOURCES	Windows DDK Sources file for this directory
SHIMINIT.CPP	Routines for opening and closing WDM files for the host
SHIMIO.CPP	The loopback test application
SHIMUTIL.CPP	Routines for creating the USB device and the serial port devices
PRECOMP.H	Header file for SHIMUTIL.CPP
SHIMUSB.H	Header file for SHIMUTIL.CPP
USBSER.H	Shared definitions for USBFIRM.C and host driver
HOST_VCOMM	Subdirectory that contains the VCOMM driver source code for the host
MAKEFILE	Windows DDK Makefile for this directory
SOURCES	Windows DDK Sources file for this directory
SHIMSER.DEF	Definitions for SHIMSER.C
SHIMSER.C	Initialization and shutdown routines for the port driver
SHIMPORT.C	Host routines for opening and closing a port
SHIMPORT.H	Definitions for SHIMPORT.C
SYSCTRL.ASM	Device control procedures for SHIMSER.VXD
MYLOCAL.INC	Include file for the VXD C wrapper
INSTALL	Subdirectory that contains the host system files
SHIMSER.VXD	The installable WCOMM driver built from the HOST_VCOMM directory
SHIMSER.INF	The Windows 98 install file for SHIMSER.VXD
SHIM.SYS	The installable WDM driver built from the HOST_SHIM directory
SHIM.INF	The Windows 98 install file for SHIM.SYS

1.4 Setting Up the Build Environment

Before the CodeKit software is built, you must set up the build environment. To do this, run the ENVIRON.BAT command files in the FIRM_BLD directory. These set up the environment variables listed in the following table.

This batch file assumes that the tools have been installed in their default locations on the 'C:' drive.

ENV Variable	Setting
ASSMROOTDIR	C:\MASM611
TOOLROOTDIR	C:\MSVC
INCLUDE	C:\MSVC\INCLUDE
LIB	C:\MSVC\LIB

1.5 Setting Up the Target Hardware

The USBPORT program expects a 48-MHz main clock. The USB clock can either be derived from the main clock, or driven from a separate crystal.

- The main clock switch is SW13 on revision 1.0 and 1.1 CDPs, and SW16 on revision 2.0 CDPs. Use a 24-MHz crystal in socket X1 and set pins 1 and 2 to OFF for the main clock switch.
- The USB clock switch is SW13 on the revision 1.0 and 1.1 CDPs, and SW16 on the Revision 2.0 CDP. The USB transceiver switch is SW12 on the revision 1.0 and 1.1 CDPs, and SW14 on the Revision 2.0 CDP.
- To use the 48-MHz internal clock, set pins 3 and 4 to OFF for the USB clock switch.
- For a 12-MHz external clock, set pins 3 and 4 to ON for the USB clock switch.
- For a 24-MHz external clock, set pin 3 to ON and pin 4 to OFF for the USB clock switch.
- On revisions 1.0 and 1.1 of the CDP, there is an external USB transceiver that must be enabled by setting pin 3 of the USB transceiver switch to ON. On the revision 2.0 CPD, there is no transceiver and pin 3 of the USB transceiver switch must be set to OFF.

1.6 Building the CodeKit Software

Building this CodeKit is very simple:

1. Execute the FIRM_BLD/ENVIRON.BAT file to set up your environment.
2. Execute the FIRM_BLD/BUILDIT.BAT file.

This builds the USBPORT.HEX file. You can download this to the target hardware.

3. Go to the Windows Start Menu and select Programs.
4. From the Programs Menu, select Windows 98 DDK.
5. From the Windows 98 DDK, select Free Build Environment.
6. A Free Build Command window will pop up. Use the CD command to switch to the CodeKit directory.
7. Use the CD command to switch to either the HOST_SHIM or HOST_VCOM directory, depending on which files you want built.
8. Type Build and press return. Windows builds the file.

This process builds the SHIMSER.VXD and SHIM.SYS files. These are installed on your Windows 98 computer when the target hardware is attached with a USB cable.

NOTE: The MAKEHEX.EXE utility converts an EXE file into a HEX file. The version of MAKEHEX.EXE you have is matched to a version of the E86MON software. If you have problems loading your HEX file using the E86MON

software on the target board, please contact our support team for a correct version of E86MON software. Note that the support team will need to know what version of E86MON software is loaded on your board.

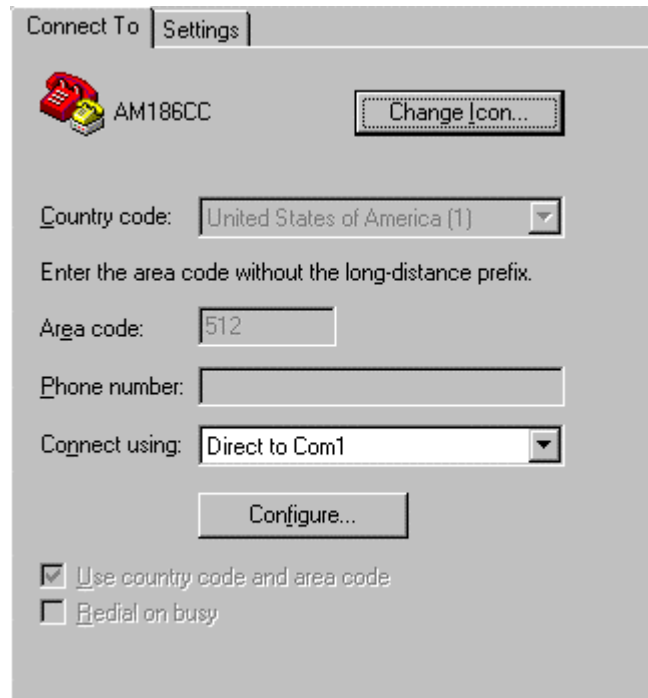
For help with MAKEHEX.EXE, run MAKEHEX with no arguments.

1.7 Loading the Code

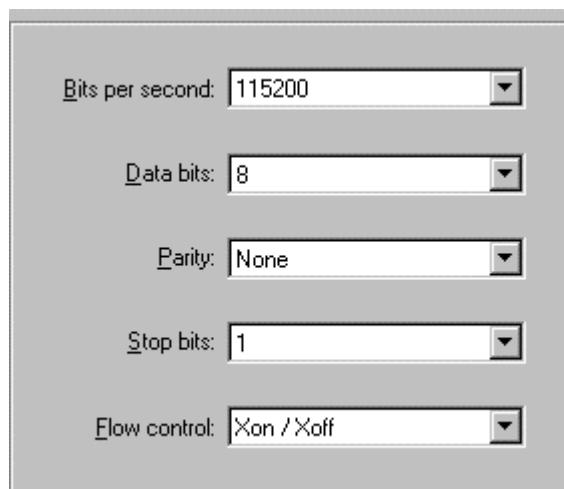
You need some kind of terminal program like HyperTerminal (on Windows 98). The following steps show you how to connect to an Am186CC/CH/CU microcontroller CDP using COM1 on your PC and HyperTerminal:

Step 1: Connect a *null modem cable* from COM1 of your PC to either of the serial ports on your Am186CC communications controller CDP.

Step 2: Run HyperTerminal and create a new connection directly to COM1.



Step 3: Click the “Configure...” button to set the baud rate and other communications parameters to 115200 Baud, 8 Data Bits, No Parity, 1 Stop Bit and Xon/Xoff Flow Control.



Step 4: Reset the target board, wait a second or two, then press the “A” (either upper or lower case). The E86MON software (running on the target board) detects the baud rate and issues a prompt, like this:

```
Welcome to AMD's EMon 186!      <? <Enter> for help>
cc86mon:
```

Step 5: Type LL to load the DOS extensions needed by the USB driver.

Step 6: This step loads the program into RAM. If you reset the board, you have to load the program again¹. Select “Send Text File” from the “Transfer” menu. (Note this is a basic ASCII file transfer. In other words, the file is sent as is with no protocol, just as if you could type it in really fast.) In the resulting dialog box, select the USBDEMO.HEX file. The file transfer begins. After it is finished, your terminal window looks something like this:

```
cc86mon: :04000003004F001694
Transferring hex file <Press Esc to abort>.....
File download successful
cc86mon:
```

Step 7: To execute the program, use the G (for GO) command. The program will now run.

Step 8: You will need to attach the board to host PC with a USB cable. The host should recognize the USB device but should not be able to find a driver for it. When it requests a driver, point it to the HOST_BLD directory. Windows uses the .INF files to install the driver.

Step 9: Go to the HOST_APP directory and run the TTY application. A window pops up. Go to the Action menu and select Connect. Now you can send messages between the TTY window and the low speed UART on the Am186CC communications controller board.

2 The Device Code

The device code implements a limited set of Windows 98 VCOMM commands that are passed to it by the host. Each command is sent on the C endpoint and a status reply is returned on the D endpoint. When a serial port is opened, data is passed from the port to the host on the Interrupt endpoint.

2.1 The USBFirm Code

The main() function begins by initializing its heap, buffer management system, and the USB interrupt handlers. The function then goes into a loop, waiting to see if the target has been attached to a host. When the connection is made, the interrupt handlers pass set-up and configuration information to the host and configure the USB endpoints.

The application then goes into a loop, polling the C data endpoint's buffer queue for data. If there is data on the C endpoint, the demonstration application retrieves the data and parses it for a known command code. If the packet contains a proper code, the application calls a function to handle it. The function will return a status code on the D endpoint. An error code is returned for any commands that attempt to manipulate a closed port except the Open command.

¹See the *E86MON™ Software User's Manual*, order #21891, for instructions on how to load programs into Flash memory.

The normal sequence of commands to open and configure a port is `OpenCommUSBF()`, `GetCommStateUSBF()`, `SetCommStateUSBF()`, `PurgeCommStateUSBF()`, followed by any number of `WriteCommUSBF()` commands. A close sequence begins with a `EscapeCommFunctionUSBF()` command and is followed by the `CloseCommUSBF()` command.

While a port is opened, the main loop checks the port's buffers for received data. Any received data is placed on the interrupt endpoint for transfer to the host.

The interrupt handlers place and remove the data in the endpoint buffer queues transparently to the `main()` code.

2.2 The USB Code

USB.C contains all the functions required to set up a USB connection and transmit and receive data. USB.C normally runs in interrupt mode, retrieving data from the endpoints and storing them in buffer queues, or taking data stored in a buffer queue and writing it to an endpoint. The demonstration application then polls the buffer queues to retrieve or send data as described in the `main.c` section above.

`USBInit()` initializes the Am186CC/CU microcontroller (the Am186CH HDLC microcontroller does not support USB) hardware for operation by setting up `USBInt()` in the interrupt vector table, enabling the channel 2 interrupt channel in the internal mode, setting the USB interrupt masks, and allocating the endpoint buffer queues. It also builds the descriptor table that `USBInt()` uses to respond to Get Configuration tokens from the host.

The `USBInt()` function handles all the interrupts generated by the USB source. It checks the USB status registers to determine what event caused the interrupt, and then calls a subfunction to handle the event. The two most important subfunctions are `USBToHost()` and `USBFromHost()`, which are the functions that move data between the endpoint buffer queues and the endpoints.

`USBFromHost()` is called separately for each endpoint that has data on it. The function allocates a buffer and fills the buffer with data from the endpoint. When a command is received on the Control endpoint, it then calls `USBProcessEPOCommand()`, which examines the buffer, decodes the command, and handles it by calling the appropriate handler function. When data is received on the C endpoint, `USBFromHost()` holds onto the buffer until it receives a short packet signaling the end of the transmission, and then it places the buffer at the end of the buffer queue for the C endpoint.

`USBToHost()` is also called separately for each endpoint that the host wants data from. The function pulls a buffer from the endpoint's queue and writes a packet's worth of data to the endpoint's transmit register. When it has consumed all the data in the buffer, the function frees the buffer. If the host wants data from an endpoint that has no buffers in its queue, a 0-length packet is sent instead.

A silicon bug in the A revision of the Am186CC communications controller caused the hardware to lock up when a 0-length packet was sent on a Data/In endpoint. `USBToHost()` was modified to send a 1-byte packet containing 0x0 when the host sent an IN token, if there was no data available. This problem does not occur if general-purpose DMA or the SmartDMA controller is used with the IN endpoint, or if a revision B or later chip is used. The symbol `REVISION_A` should be defined in USB.C when a revision A chip is used to enable this additional code.

2.3 Other Code

AM186SER.C contains all the functions required to set up asynchronous serial port connections. AM186SER.C defaults to a 48-MHz main clock. The default clock speed can be changed by editing line 495 of AM186SER.C and recompiling the code.

BUFFER.C contains all the functions required to set up allocable buffers and queues of buffers.

NEWHEAP.C contains all the functions required to set up a heap to be used by BUFFER.C.

LLINT.ASM contains interrupt-handling routines used by AM186SER.C.

BQUE.C contains additional queue functions used by AM186SER.C.

2.4 Open Issues and Design Notes

The code as it is currently written is meant to run on a revision 2.0 Am186CC communications controller CDP. The revision 2.0 CDP does not include certain modem status signals, such as Ring Indicate or Carrier Detect. The

revision 2.1 CDP includes those signals for the high-speed serial port. If you are compiling this code for use with a revision 2.1 CDP, the symbol `FULL_MODEM_HS` should be defined for the `USBfirm.c`. If you are using another board that has full modem signals for both serial ports, then `FULL_MODEM_HS` should be defined and the low-speed PIOs should be defined.

Certain signal PIOs, such as `USB_VCC_DETECT`, are defined in `USB.C`. These definitions will work for all Am186CC communications controller reference designs released by EPD. If you are using this code on a non-AMD board, you should either design the board to route the same signals to the same pins, or alter the signals to reflect the board design.

This code is not recommended for a production design. The interrupt endpoint should not be used for transmitting high volume data. Configuration commands should not be sent on a bulk endpoint. Instead, the configuration commands and replies should be handled as vendor specific commands on the control endpoint, and each asynchronous serial port should have two bulk endpoints, each with a dedicated SmartDMA channel, to provide bidirectional communication. Each serial port should use two general-purpose DMA channels to transfer data. The interrupt endpoint would be used to signal flow control.

A revised CodeKit containing a near production design of this software may be available in the future.

More information about the code is supplied in the inline documentation.

3 The Host Code

The host driver consists of a VCOMM virtual device driver (`SHIMSER.VXD`) and a WDM device driver (`SHIM.SYS`). `SHIMSER.VXD` provides a standard interface for the Windows-standard VCOMM communications API, and `SHIM.SYS` interfaces with the Windows low-level USB drivers.

3.1 SHIMSER Code

Almost all the functions in this virtual driver are callback functions. The Windows 98 VCOMM API calls these functions when dealing with devices attached to either of the virtual serial ports. The functions either create VCOMM structures for handling additional calls, or pass information requests or data to `SHIM.SYS` for transmission to the Am186CC communications controller.

`SHIMSER.C` contains the initialization and shutdown routines for the VCOMM driver. These functions are called when the device is first plugged in.

`SHIMPORT.C` contains high-level implementations of the VCOMM commands. Each function sets any appropriate variables and then calls `ioctl()` to pass a request to `SHIM.SYS`. Several commands are not passed on to `SHIM.SYS` because they deal with higher-level structures that the Am186CC communications controller is not aware of.

3.2 SHIM Code

Almost all the functions in this driver are called by `SHIMSER.VXD` calling the `ioctl()` function to pass a message to `SerialIoControl()`. `SerialIoControl()` passes the request and calls the appropriate function. A few of the functions are callbacks that Windows uses to open or close the driver.

`SHIMINIT.CPP` contains the driver entry function `DriverEntry()` and the seven Windows control functions. Windows will call these functions in response to user application requests. It also contains `SerialIoControl()`, the function that parses requests from `SHIMSER.VXD` and calls the appropriate `SHIM.SYS` function to pass the data to the firmware.

`SHIMIO.CPP` contains the functions that create data structures that are passed to the firmware in `SHIM_RunCommand()`. `SHIMIO.CPP` also contains the functions that check the receiver data buffers for stream data received from the firmware on the interrupt endpoint. If `SHIMIO.CPP` finds data, it uses a callback function to pass the data back up to `SHIMSER.VXD`.

`SHIMUTIL.CPP` contains the utility functions that read and write data to and from the Windows USB D class driver. These functions also perform device enumeration. These functions are called by functions in `SHIMIO.CPP` and in turn call USB D functions to pass their messages to the firmware. The code is configured to assign new USB devices to COM ports 12 and above. The default value is determined by the `BASE_ADDRESS` symbol at the top

of SHIMUTIL.CPP. Setting BASE_ADDRESS to 3, rebuilding the code, and re-installing the USB driver causes the USB virtual ports to appear as COM 3 and COM 4, for use with standard terminal programs such as Hyperterm or ProComm.

If you intend to modify these drivers for use with your own USB devices, the most important functions are the seven Windows control functions assigned in the DriverEntry() function of SHIMINIT.CPP. These are the functions that determine how the host will interact with the driver. SHIM_CreateDeviceObject() is the function that registers the device with Windows. SHIM_StartDevice() gets the USB descriptors from the device and configures the device.

3.3 Other Issues and Design Notes

Many of the functions in these files do not examine thoroughly the responses returned from the device. Additional error handling should be added before this device is used in a production design.

More information about the code is supplied in the inline documentation.

Trademarks

© 2000 Advanced Micro Devices, Inc. All rights reserved.

AMD, the AMD logo, and combinations thereof, Am186, E86MON, and SmartDMA are trademarks of Advanced Micro Devices, Inc.

Microsoft and Windows are registered trademarks of Microsoft.

All other product names are for identification purposes only and may be trademarks of their respective companies.