Debugging an Am186EM Application on the SD186EM demonstration board Using Microsoft's C/C++ Compiler and Paradigm Debug Software

My Tran
Advanced Micro Devices, Inc.

OVERVIEW

This application note provides users with step-by-step instructions for building and debugging an Am186EM application program using Microsoft's C/C++ compiler with Paradigm Debug software. To debug an embedded program, the debugger must be able to communicate with the monitor on the target board. The SD186EM board comes with the E86MON monitor. However, this monitor was not intended to function with the Paradigm Debug software; therefore, the Paradigm Remote ROM monitor (PDREM) must be built and downloaded to the board before a debugging process can take place. The user needs to install the following tools on his/her PC (PC386 or better, with at least 4 Mbyte of memory) in order to build and debug an Am186EM application using the Microsoft C/C++ compiler with the Paradigm Debug software:

- Microsoft C/C++ compiler, version 1.5 [1]
- Paradigm LOCATE, version 5.0
- Paradigm Debug/RT-186, version 4.0
- Microsoft Assembler, version 6.1
- Paradigm PDREM, version 4.02
- AMD SD186EM Demonstration board

The process of debugging an Am186EM application involves two basic steps: (1) building Paradigm Remote ROM monitor (PDREM) and downloading it to the board; (2) building and debugging an Am186EM application program. Other tools and utilities are important factors in building an embedded application; thus, this document is divided into five sections:

1. Installing Paradigm software
2. Building and Downloading PDREM
3. Building an Am186EM application
4. Optimizing the application program
5. Building an AM186EM Standalone program

Before installing the Paradigm software, please review the section Installing Paradigm Software below for specific information. This document assumes the software is installed on c:\xxx, where *xxx* is the directory where the software resides. As such, the directory names used throughout this document have the following meanings:

c:\msvc	- Microsoft C/C++ compiler directory
c:\masm_61	- Microsoft Assembler directory
c:\locate	- Paradigm LOCATE directory
c:\pdrem	- Paradigm PDREM directory
c:\pd	- Paradigm Debug/RT-186 directory

If the user installs his/her software in a different directory or disk drive, please replace the string **c:\xxx** with the correct path names.

[1] Paradigm Debug software supports different compilers from different vendors

INSTALLING PARADIGM SOFTWARE

Microsoft's C/C++ compiler and assembler should be installed before installing the Paradigm software. The Paradigm software discussed in this document includes: Paradigm LOCATE, Paradigm PDREM, and Paradigm Debug/RT-186. This software is distributed by Paradigm Systems. To obtain a copy of any of these, please contact Paradigm Systems at 1-800-582-0864. Throughout this document, the terms "Paradigm Debug/RT-186" and "Paradigm Debug/RT" are used interchangeably.

This section provides specific information for installing Paradigm software to run on the SD186EM board. Users should follow the installation instructions of the appropriate manuals distributed by Paradigm Systems when installing Paradigm software.

1. **Installing Paradigm LOCATE**: Paradigm LOCATE serves as the locator. It takes input from a relocatable .EXE file produced by the compiler/linker, and assigns memory addresses defined in the .CFG file to produce an absolute executable file (.AXE) suitable for the embedded design. Since it is the basic building block of the Paradigm tools, it should be the first Paradigm product to be installed.

   The Paradigm LOCATE installation program invokes Microsoft's C/C++ compiler to build the ROMable runtime libraries. Be sure to include the **bin** directory of the Microsoft compiler in the environment variable **PATH** (i.e., **c:\msvc\bin)** before installing Paradigm LOCATE. During the installation process, the user will be asked to choose a directory for the ROMable runtime libraries. The libraries in this directory will be searched during link time to build a ROM object file. For this reason, the extension file produced by the compiler/linker is sometimes called .ROM (versus .EXE). Paradigm LOCATE accepts object files with the extension of either .ROM or .EXE (See Figure 1). It is a good idea to include the directory of the ROMable libraries in the environment variable **LIB** after the installation. For example, if the ROMable libraries are in the directory **c:\msvc**, then the environment variable **LIB** should contain the string **c:\msvc\lib**.

   To run Paradigm LOCATE from a Microsoft Window's DOS box, insert the following line in the **[386Enh]** section of the file **c:\windows\system.ini**. Note: The **windpmi.386** driver is not installed until the user exits, and re-enters Windows.

   device=c:\locate\windpmi.386

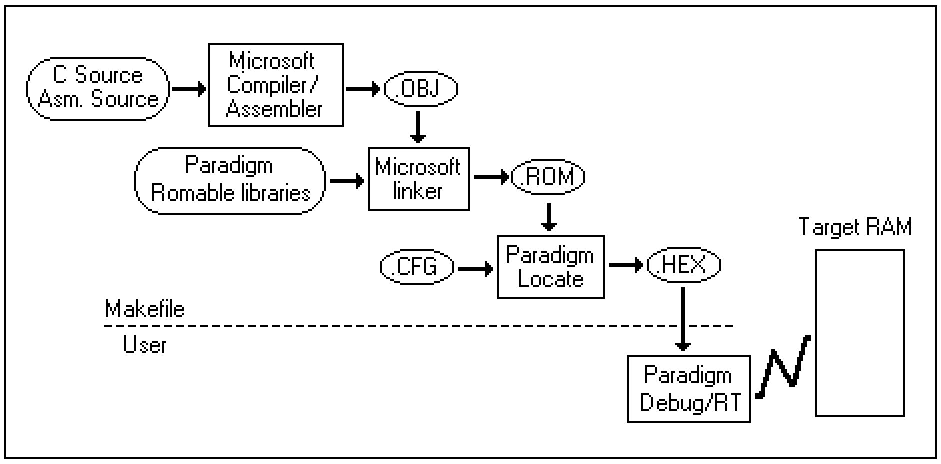


Figure 1 - Debugging an Am186EM Application program

2. **Installing Paradigm PDREM**: The installation program needs to know something about the software and hardware environment to retrieve correct source code for building PDREM. When installing PDREM, be sure to set the following parameters as indicated:

- Target processor:	AMD Am186EM
- Target Math coprocessor:	None
- Target UART:	Am186EM/internal
- System compiler:	Microsoft C/C++ 8.0 [2]

   When the installation is finished, source files needed for building PDREM files for the SD186EM board are copied to the hard drive. “Building and Downloading PDREM” shows how to build the PDREM monitor and download it to the target board.

3. **Installing Paradigm Debug/RT-186**: The Paradigm Debug/RT-186 installation program will request information such as the serial communication port and baud rate. The specified baud rate defines the communication channel between Paradigm Debug/RT-186 (**pdrt186)** and PDREM. Though the E86MON which resides on the SD186EM board is programmed to communicate with the host at the baud rate of 19200, **pdrt186** may not communicate with the target at this baud rate; rather it communicates with the target at the baud rate defined in the **dcomms.c** file in the **pdrem** directory. Make sure the baud rate being entered here matches the one defined in **dcomms.c**.

```
……
/* Select Xtal frequency and desired BAUD Rate                 */
#define   CLK      40000000UL      /* X1/X2 Crystal input frequency */
#define   BAUD     115200UL        /* Desired BAUD rate          */

#define  IOB186EM  0xff00          /* Location of PCB */

#include   <dos.h>                 /* Enable/disable functions */

#include   “typedef.h”             /* Common definition and prototypes */
#include   “target.h”              /*  Target system specific information */
#include  “helpers.h”              /*  PDREMOTE/ROM helper functions */
……
```

Figure 2 - Setting the baud rate in the **pdrem\dcomms.c** file

   In general, the PDREM file (**pdrem.hex**) is downloaded to the board at a baud rate of 19200. Afterward, the communication between Paradigm Debug/RT and the target is defined by PDREM, not by E86MON. The default baud rate defined in the **dcomms.c** file is **115200**. To change PDREM’s baud rate, simply change this number to a valid baud rate and rebuild PDREM. Figure 2 shows a portion of the **dcomms.c** file; the line defining the baud rate is shown in boldface.

4. **Setting the PATH:** After the Paradigm software is installed, the user should set the PATH environment variable to include the directories of Paradigm LOCATE, Paradigm Debug/RT, and the **bin** directory of the Microsoft C/C++ compiler and assembler. For instance, if Paradigm LOCATE is installed in the directory **c:\locate**, Paradigm Debug/RT is installed in the directory **c:\pd**, the Microsoft C/C++ compiler is installed in the directory **c:\msvc**, and the Microsoft Assembler is installed in the directory **c:\masm_61**, then the PATH environment variable should contain the string:

[2] Microsoft’s C compiler version 8.0 is the same as Microsoft’s Visual C++ version 1.5

c:\locate;c:\pd;c:\msvc\bin;c:\masm_61\bin

BUILDING and DOWNLOADING PDREM

After installing the Paradigm software, the user is ready to build and download the PDREM monitor to the SD186EM board. Figure 3 shows the process of building and downloading PDREM.

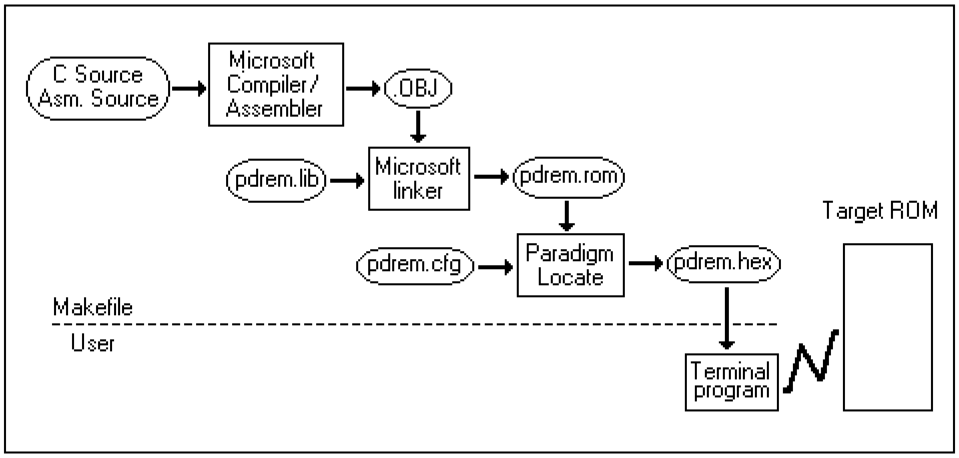


Figure 3 - Building and Downloading PDREM

1. BUILDING PDREM:

Building PDREM involves the following steps:

- "cd" to pdrem directory (i.e., **cd c:\pdrem**).
- Edit the **target.h** file as shown in Figure 4.
- Edit the **pdrem.cfg** file as shown in Figure 5.
- Run **nmake** or **nmaker** (real-mode nmake).

```
.......

#if  0                          /* Disabled: include if PDREMOTE/ROM UI is used */
#include    "kernel.h"          /* Application interface extensions */
#endif

#if  1                          /*0 = no, 1=yes Define if interrupt-driven serial I/O*/
#define COMMINT     0x14        /* Change to desired interrupt vector */
#endif

........
```

Change these 2 lines to enable I/O

Figure 4 - Defining I/O Interrupt Vector

Building PDREM is mostly taken care of by the **makefile** created by the PDREM installation program. The **makefile** performs the necessary steps to produce the ROM PDREM file called

pdrem.hex. However, before invoking **nmake** (or **nmaker**), the user needs to be aware of the following items:

- Though PDREM supports a virtual UART interface that the application can use to communicate with the Remote Console window of the Paradigm Debug/RT, the **target.h** file in the **pdrem** directory does not have the serial I/O interrupt vector set. To be able to send the string from a printf() statement of a C program to the Remote Console window, the user needs to change two lines of code in the **target.h** file as shown in Figure 4. The information that needs to be changed is shown in boldface.

```
//
//          Paradigm LOCATE configuration file for building the PDREMOTE/ROM kernel.
//          Copyright (C) 1993 Paradigm Systems.  All rights reserved.
//

hexfile intel86                                // Intel hex output
listfile     segments                          // Create a segment map

map    0x00000 to 0x003ff as reserved          // Interrupt vector table
map    0x00400 to 0x00fff as rdwr              // PDREMOTE/ROM data area
map    0x01000 to 0xdffff as reserved          // Reserved for applications
map    0xe0000 to 0xfffff as rdonly            // PDREMOTE/ROM kernel EPROM

cputype AM186EM                               // Select the processor type

initcode inbyte 0x1000                        // inbyte is here to create ??LOCATE class

// Chip select initialization not necessary (flash utility sets up
// chip selects).  However, there is a memory limitation on how many
// initcode parameters can be used at address 0xf7ff0 (will run into
// flash downloader).  Since the flash downloader will jump to address
// 0xf7ff0, the initcode is used here to serve as a jump to pdremote's
// code.  The class, ??LOCATE, will contain the jump automatically.
//
//                      umcs = 0xfe38       \
//                      lmcs = 0x0ff8       \
//                      mpcs = 0x80bf       \
//                      pacs = 0x003f       \

dup          DATA ROMDATA                      // Make a copy of the initialized data

class        CODE = 0xe000                     // Modify to set the EPROM address
class        DATA = 0x0040                     // And the RAM address
class        ??LOCATE = 0xf7ff                 // Initialization code

order        DATA                              \    // Fix the class ordering
                        CONST                  \
                        BSS BSSEND             \
                        STACK

order        CODE                              \    // Code, initialized data in EPROM
                        ROMDATA ENDROMDATA

output       CODE                              \    // Write to the output file
                        ROMDATA ENDROMDATA   \
                        ??LOCATE
```

These are commented out, E86MON has already set these up before jumping to

Figure 5 **- pdrem.cfg** File for SD186EM Board

- The **pdrem.cfg** file created by the installation is not customized for the SD186EM board. The correct configuration for the SD186EM board is shown in Figure 5. The information that needs to be changed is shown in boldface.

- Use **nmaker** (real-mode nmake) instead **nmake** to run the makefile in a Microsoft Windows DOS box to avoid possible conflict between **nmake** and **locate**. The README.TXT file in the **c:\locate** directory explains this conflict in more detail.

After **nmake** (or **nmaker**) finishes, the ROM file **pdrem.hex** in the **pdrem** directory is created and is ready to be downloaded to the SD186EM board.

2. DOWNLOADING PDREM TO THE SD186EM BOARD:

The purpose of downloading PDREM to the SD186EM board is to create a high level source and data debugging environment for the Paradigm Debug/RT. The PDREM file (**pdrem.hex**) can be downloaded to the target by any communication program, provided that the protocols are set up correctly. This section shows how to download the PDREM monitor to the target board using the Windows **Terminal** program available in Microsoft Windows. The process involves the following steps:

a) Invoke the **Terminal** program by selecting the **Accessories** icon from the **Program Manager** Window; then double click on the **Terminal** icon.

b) When the **Terminal** program starts up, click **Settings|Communications** to set the communication protocol as shown in Figure 6A; where **COM2** displayed in **connector** should be replaced by the serial port which connects the SD186EM board to the PC, then hit **OK**. As mentioned in the previous section, the baud rate defined on the target system at this point is controlled by E86MON and is programmed at 19200 (PDREM has not been downloaded yet).

c) Push the reset button on the board, the LEDs should flash off and on. Press <ctrl-break>[3] on the keyboard to signal E86MON to begin operation. Once communication is established, the **e86**: prompt will appear on the screen.

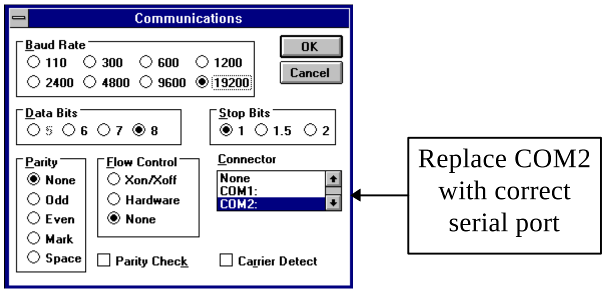
d) Click **Settings|Text Transfer** from the **Terminal** program, and set the handshaking protocol as shown in Figure 6B, then click **OK**.

e) From the **Terminal** program, type a **p**<CR> at the **e86**: prompt to tell the monitor to receive a HEX file and program it into the FLASH memory. The message “Begin file transfer...” appears on the screen.

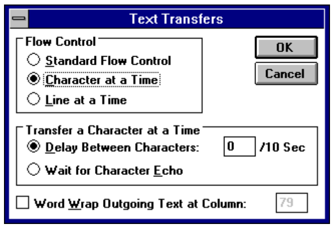
f) Click **Transfer|Send TextFile...** from the **Terminal** program. As soon as the selection is made, a window with a list of file names pops up, select the **pdrem.hex** file in the **pdrem** directory. When the user clicks the OK button, E86MON starts programming the FLASH memory using data in the **pdrem.hex** file. During this time, the LEDs on the board flash randomly, and the **Terminal** program reports the status of the process on the screen.

[3] Press Ctrl key and Break[Pause] key at the same time.

g) Wait until the **e86:** prompt shows up on the screen again, then reset the target, and wait for several more seconds before invoking Paradigm Debug/RT.



Baud Rate: 110 300 600 1200 2400 4800 9600 [✓ 19200]
Data Bits: 5 6 7 [✓ 8]
Stop Bits: [✓ 1] 1.5 2
Parity: [✓ None] Odd Even Mark Space
Flow Control: Xon/Xoff Hardware [✓ None]
Connector: None COM1: [✓ COM2:]
Parity Check
Carrier Detect



Flow Control: Standard Flow Control [✓ Character at a Time] Line at a Time
Transfer a Character at a Time: [✓ Delay Between Characters: 0 /10 Sec] Wait for Character Echo
Word Wrap Outgoing Text at Column: 79

Figure 6A - Setting Communication Protocol Protocol

Figure 6B - Setting Handshaking

The board is now ready to communicate with Paradigm Debug/RT **pdrt186**. Once PDREM is downloaded and programmed, it stays in the FLASH memory. Unless there are changes in PDREM's source file, or somehow the PDREM on the target board is contaminated, there is no need to download PDREM again regardless of how often the application program is compiled, loaded, and executed. Building and debugging an Am186EM program is discussed in the next section.

BUILDING AN AM186EM APPLICATION PROGRAM

The process of building and debugging an Am186EM application program is demonstrated in Figure 1 on Page 2; it is similar to the process of building the PDREM file. The source files are compiled and linked by Microsoft tools to produce a ROM relocatable object file (.ROM). Paradigm LOCATE takes information defined in the .CFG file, and assigns code and data addresses to the .ROM file to produce an Absolute Executable file (.AXE). The .AXE file is then loaded and debugged using Paradigm Debug/RT **pdrt186**.

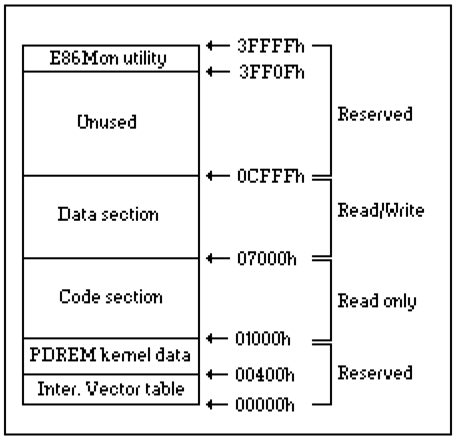
Besides building the ROMable runtime libraries as mentioned in the previous section, the Paradigm LOCATE installation program also installs other utilities needed for building an embedded program, such as startup code, library helpers, as well as examples showing how to build various application programs. Each example has a **readme.txt** file containing useful information. The example directories are placed in the directory **c:\locate\msc80\examples**. Unfortunately, these examples were configured to run on an in-circuit emulator. This section shows how to make the **demo** program in the **examples** directory work with the SD186EM board. It is divided to four parts:

1. Building the demo program
2. Invoking Paradigm Debug/RT
3. A look at the makefile
4. Running the standard I/O program - stdio.c

1. Building the demo program:

The demo program is located in the **c:\locate\msc80\examples\demo** directory, the name of the demo program is **sieve.c**. Building the demo program to run on the SD186EM board involves the following steps:

- "cd" to the demo directory (i.e. **cd \locate\msc80\examples\demo**).
- Change the **sieve.cfg** configuration file as shown in Figure 8.
- Run **nmake** or **nmaker** (real-mode nmake).



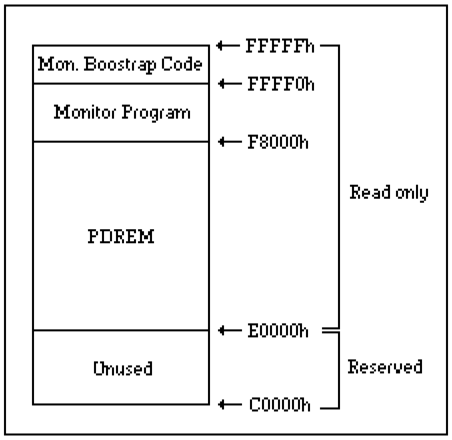


Figure 7A - RAM

Figure 7B - ROM (FLASH)

♦ **Changing the configuration file**: The configuration file is the place where target-specific information such as memory configuration is defined. Figures 7A and 7B demonstrate the memory configuration for executing the sieve.c program. Figure 8 shows the memory configuration file **sieve.cfg** based on the memory models shown in Figures 7A and 7B. This file has been modified from the original version. The lines shown in boldface in Figure 8 indicate information that has been changed. Among the changes are:

- Line 15: This line maps memory addresses 0x0 to 0xfff as reserved. This memory location contains interrupt vector table and PDREM kernel data.

- Line 16: This line maps 24 Kbyte of RAM memory ranging from 0x1fff to 0x6fff to be rdonly (read only). It is used for storing the code section of the application. If the code section of the application program is bigger than 24 Kbyte, the user should adjust the size of the region. The length of each segment of class CODE is listed in the .MAP file (generated by the compiler), or .LOC file (generated by Paradigm LOCATE). The exact size of the CODE section can be obtained by tallying the lengths of the segments belong to class CODE.

- Line 17: This line maps 24 Kbyte of RAM memory ranging from 0x7000 to 0xcfff to be readable and writeable (rdwr) for storing the data section of the application. If the data section is bigger than 24 Kbyte, the user should adjust the size of this region. The length of each segment of class DATA is listed in the .MAP file (generated by the compiler), or .LOC file (generated by Paradigm LOCATE). The exact size of the DATA section can be obtained by tallying the lengths of the segments belonging to class DATA.

- Line 18: This line maps memory addresses from 0x0d000 to 0xdffff as reserved. Most memory space in this region is unused, protected (E86MON utility), or non-existant on the SD186EM board. Paradigm Debug/RT will catch the error if an application program attempts to access the addresses defined as reserved.

- Line 19: This line defines the PDREM and E86MON memory regions as read-only. As defined in the **pdrem.cfg** file, PDREM starts at address 0xe0000.

```
 1    //
 2    //       Configuration file for the Microsoft C/C++ example from
 3    //       the Paradigm LOCATE manual.  This is about as basic
 4    //       as an embedded application can get.  Check out the
 5    //       other examples for more sophisticated applications.
 6    //

 7    hexfile intel86               // Intel extended hex for EPROMs
 8    absfile axe86                 // Optional Paradigm DEBUG support
 9    listfile    segments          // Segment map for documentation

10    //
11    //       Define how the target system address space is
12    //       partitioned.  Paradigm LOCATE will check that this
13    //       mapping is followed.
14    //

15    map 0x00000 to 0x00fff as reserved   // PDREMOTE and interrupt vector table
16    map 0x01000 to 0x06fff as rdonly     // 24K bytes of code
17    map 0x07000 to 0x0cfff as rdwr       // 24K bytes of data
18    map 0x0d000 to 0xdffff  as reserved  //  inaccessible memory
19    map 0xe0000 to 0xfffff as rdonly     //  ROM - EPREM starts at 0xe000

20    cputype i80C186EA                    // Target CPU is defined here

21    //initcode    reset                \    // Reset vector
22    //            umcs = 0xf838        \    // 32KB EPROM
23    //            lmcs = 0x0ff8             // 64KB RAM

24    dup     DATA ROMDATA            // Dup initialized data
25    dup     CONST ROMDATA
26    dup     MSG ROMDATA

27    class       CODE = 0x0100         // Code at 1000h
28    class       DATA = 0x0700         // Data at 7000h
29    //class       ??LOCATE = 0xfff0   // 80C186EA chip select code

30    order      DATA CONST MSG     \     // RAM class organization
31                     BSS            \
32                     STACK

33    order      CODE               \     // EPROM class organization
34                     ROMDATA ENDROMDATA     \

35     output  CODE                 \     // Classes containing code/data
36                     ROMDATA ENDROMDATA     \
37    //                          ??LOCATE
```

Figure 8 - **sieve.cfg** File for SD186EM Board

- Lines 21 to 23 are commented out because the chip select registers have been set by the E86MON utility.

- Line 27 indicates that the code section starts at segment 0x100 (or at address 0x1000[4]) as indicated in the map statement on line 16. Line 28 indicates that the data section starts at segment 0x700 (or at address 0x7000[4]) as indicated in the map statement on line 17.

- Lines 29 and 37 are commented out because there is no code for chip select.

♦ **Running nmake** or **nmaker**: To build the absolute executable file, simply invoke **nmake** from the **demo** directory (make sure the environment variable **LIB** contains the directory of the run-time ROMable libraries as described in the section "Installing Paradigm LOCATE"). When **nmake** finishes, the absolute executable file **sieve.axe** is created and is ready for debugging. NOTE: Use **nmaker** instead of **nmake** in a Microsoft Windows DOS box to avoid possible conflict between **nmake** and **locate**.

Though Microsoft Visual Workbench (VWB) included in the Microsoft C/C++ compiler is a powerful tool, it is incompatible for use in embedded applications. VWB does not support Microsoft assembler **MASM** directly, and it always assumes that the user wants to create a PC-type application. The user must take care of the extra steps such as including ROM startup code and run-time libraries in a makefile. Once the makefile is generated, it seems easier to invoke **nmake** from a DOS shell rather than trying to include the external makefile in the VWB environment. Page 118 of the Paradigm LOCATE Reference Manual explains in detail how to include an external makefile from the VWB environment.

2. **Invoking Paradigm Debug/RT - PDRT186**: **pdrt186**

To invoke Paradigm Debug/RT, and load the demo program from the command line, simply invoke the "**pdrt186 sieve**" command from a DOS shell. The serial communication information such as the communication port and baud rate is defined in the **pdrt186.ini** file, which was created when Paradigm Debug/RT was installed. The baud rate defined in this file must match the baud rate defined in the **dcomms.c** file in the **pdrem** directory. Information in the **pdrt186.ini** file can be changed at anytime; it resides in the Paradigm Debug/RT **c:\pd** directory.

3. **A look at the makefile**

The **makefile** included in the **demo** directory is ready to build the **sieve.c** program. A closer look at the makefile reveals that there are two other files that are compiled and linked with the demo program to produce a ROMable object file; the files are: **msc80.asm**, and **cinit.asm**. These are two of the five startup files being distributed with the Paradigm LOCATE software. Startup code takes care of necessary steps from the time the target is reset to the time the application program is executed. The source code of the startup files resides in the **c:\locate\msc80** directory. Startup code, if used, should always be linked first.

The **sieve.c** demo program is a simple program, more sophisticated programs are demonstrated in other sub-directories of the **examples** directory. These programs are not only linked with the startup code, they are also linked with ROMable run-time library helpers. The sources of the library helpers reside in the **c:\locate\msc80\helpers** directory. Library helpers define variables/symbols and interrupt handlers needed by the ROMable run-time libraries. When compiling the application program, if the linker generates some "Undefined symbols" messages, chances are those unfamiliar symbols are

[4] Physical address is calculated by shifting the segment value left 4 bits and adding the 16-bit offset value to yield a 20-bit physical address. The value of the 16-bit offset is 0 in this case.

defined in one of the library helper files. The user may want to search for the symbols in the library helper files to find the correct helpers to be included in the makefile. The Paradigm LOCATE Reference Manual explains the functions of these files in detail.

The user should make use of the makefiles and configuration files in the **examples** sub-directories. The **readme.txt** file in each directory explains the purpose and features of each example.

4. **Running the standard I/O program - stdio.c**

An example program demonstrating the Paradigm Debug/RT's ability to handle standard I/O is distributed in the Paradigm LOCATE software. This example is located in the **c:\locate\msc80\examples\stdio** directory, and the name of the program is **stdio.c**. Building the stdio.c program to run on the SD186EM board involves the following steps:

- "cd" to the stdio directory (i.e., **cd \locate\msc80\examples\stdio**).
- Change the configuration file.
- Change the value of the DEBUG variable in the makefile to 2.
- Run **nmake** or **nmaker**.

There are two configuration files in the **stdio** directory, they are: **stdio.rm**, and **stdio.rt**. Neither of these files was customized for the SD186EM board; therefore, the memory mapping addresses and segment addresses for the class CODE and DATA must be changed. The CODE and DATA sections shown in Figure 7A should be big enough to handle stdio.c. Users should change the addresses of the **map** statements, and the segment addresses of the **class** statements of the selected configuration file to be the same as the corresponding statements shown in Figure 8.

The makefiles in this and in most other example directories are designed to build different object files, depending on the value of the DEBUG variable in the makefile. If the value of DEBUG=0, the configuration file **stdio.rm** will be chosen to build the standalone version; if the value of DEBUG=1, the configuration file **stdio.rm** will be chosen to build the emulator version; if the value of DEBUG=2, the configuration file **stdio.rt** will be chosen to build the PDREM version. To build a PDREM version of **stdio.c**, simply change the value of the DEBUG variable in the makefile to 2 (See Figure 9), then run **nmake** (or **nmaker)**. When **nmake** finishes, it creates the absolute executable file **stdio.axe**. From a DOS shell, **stdio.axe** can be downloaded to the RAM area of the SD186EM board using the **pdrt186 stdio** command.

```
......

FARDATA    =     0        # 0 - none, 1 - normal, 2 - compressed

DEBUG      =     2        # 0 - none, 1 - debug EPROM, 2 - debug PDREMOTE
WARNINGS  =      2        # 0 - none, 1 - min, 2 - medium, 3 - max
OPTIMIZE   =     0        # 0 - none, 1 - size, 2 - speed
 ......
```

Figure 9 **-** Makefile to build a debuggable program

The input and output of the **stdio.c** program delivered with Paradigm LOCATE reads input from and writes output to memory buffers defined in the **console.c** file. The interactive version of the stdio program and its related files is distributed with the PDREM software in the **c:\pdrem\demo\console** directory. Instructions in the **readme.txt** file in this directory explain in detail how to make **stdio.c** work interactively.

OPTIMIZING THE APPLICATION PROGRAM

Do not try to optimize an application program when it is in the debugging process, because optimization may move instructions around, and thus make it harder to debug. When the program is fully functional, optimization may be applied to improve the speed of the program and reduce the size of the code; these are two important factors in designing an embedded application. Certain optimization options are turned on by default (/Oc, /Of, /Oo, /Ot, /Ov). This section attempts to explain briefly other optimizations users might want to try to produce faster and smaller code. Please review the Microsoft C/C++ Environment and Tools manual for a complete listing.

/G1	Processor specific instructions	Use this option to produce 80186 instructions
/Gr	Register calling convention	This option instructs the compiler to pass arguments between functions using registers as much as possible. Thus it increases the speed. WARNING: Using this option with inline assembly language may cause conflicts in the use of registers.
/Gs	Turn stack checking OFF	Use this option to eliminate the stack-probe routine being called at every function entry point, thus it enables the compiler to produce faster code.
/Gy	eliminate dead-code at link time	This option eliminates unintended functions at link time, thus it enables the linker to produce smaller code.
/Ob[0,1,2]	Control inline expansion	Inline expansion produces faster code because the overhead associated with function calls is eliminated, but it will increase the size of the code. /Ob0: Disable inline expansion (default with /Od) /Ob1: Only expand functions marked as inline or __inline /Ob2: Expand functions marked as inline or __inline and any other functions the compiler chooses
/Oe	global register allocation	This option allows the compiler to store frequently used variables and expressions in registers, thus increases the speed and decreases the size of the code. This option should always be turned ON.
/Og	Global-level common subexpression optimization	When the same expression is repeated within a function, and the value has not changed, the compiler calculates the expression and stores it in a temporarily variable. Thus this option decreases the size, and increases the speed. This option should always be turned ON.
/Oi	Generate intrinsic functions	This option instructs the compiler to replace the common string or memory functions such as memcpy, strcpy, etc. with their inline code. As with inline option /Obn, intrinsic functions are faster, but they may be larger due to additional code generated.
/Ol	Enable loop optimization	This option removes invariant code within a loop. Any expression whose value is constant inside the loop will be moved outside when this option is turned ON. Thus it increases the speed of the program execution.
/Os	Minimize executable file size	This option minimizes the size of the object file. Though it produces smaller code, it may also slow the code.
/Ox	Maximize optimization	This option is a short way to combine optimizing options to produce the fastest possible code. It is the combination of the following options: /Oblcegilnot /Gs.
/Oz	Turn on potential unsafe loop optimization	This option works like /Ol, only it is much more aggressive. WARNING: Use of this option may generate unwanted “features”.

BUILDING AN AM186EM STANDALONE PROGRAM

The main function of PDREM is to create an environment for high level debugging. Once the debugging process is done, PDREM is no longer needed in memory. The debug information included in the object files such as symbol names only takes up space, and has very little use. Therefore, to build a stand-alone program, users need to recompile the source code to discard the debug information, and add other optimizations to create more productive code. Users should also re-LOCATE the .ROM file using a configuration file similar to the **pdrem.cfg** file to make use of the memory region previously occupied by PDREM.

This section shows how to build the **sieve.c** demo program as a stand-alone program; it is divided into three parts: Changing the makefile, changing the configuration file, and downloading a stand-alone program.

1. **Changing the makefile**: The makefile of the **sieve.c** demo program is very simple; the user only needs to add the desired optimization options to the command **cl** in the makefile. As mentioned earlier, the standalone versions of **stdio.c** program and other examples are controlled by the value of the variable DEBUG in the makefile. In these cases, simply add the desired optimization options and change the value of DEBUG from 2 to 0 (Figure 10).

```
......

FLOAT      =    0       # 0 - none, 1 - alternate, 2 - emulator, 3 - coprocessor
FARDATA    =    0       # 0 - none, 1 - normal, 2 - compressed

DEBUG      =    0       # 0 - none, 1 - debug EPROM, 2 - debug PDREMOTE
WARNINGS   =    2       # 0 - none, 1 - min, 2 - medium, 3 - max
OPTIMIZE   =    0       # 0 - none, 1 - size, 2 - speed
 ......
```

Figure 10 - Makefile to Build a Standalone Program

2. **Changing the configuration file**: The configuration file for building a standalone program should be similar to the **pdrem.cfg** configuration file shown in Figure 5, except for the memory mapping statements. Figure 11 shows the configuration file for building the **sieve.c** program as a standalone program.

```
......
hexfile intel86                                // Intel extended hex output
listfile    segments                           // Create a segment map

map   0x00000 to 0x003ff as reserved           // Interrupt vector table
map   0x00400 to 0x01fff as rdwr               // data area
map   0x02000 to 0xdffff as reserved
map   0xe0000 to 0xfffff as rdonly             // EPROM

cputype Am186EM                                // Select the processor type

initcode    inbyte 0x1000                      // Reset vector
.......
```

Figure 11 - Memory Mapping for a Standalone Program

The user can use information in the .LOC or the .MAP file to calculate the exact address range of the read-write data in the RAM, and the read-only data in the ROM. Memory needed for RAM is calculated by subtracting the **Stop** address of class **STACK** from the **Start** address of the first **DATA** class. Memory needed for ROM is calculated by subtracting the **Stop** address of the class **ENDROMDATA** from the **Start** address of the class **CODE**. A portion of the **sieve.loc** file is shown in Figure 12. According to this data, the minimum RAM needed for this application alone is C0Fh (100Fh **-** 400h), and the minimum ROM needed for this application alone is 17Fh (E017Fh **-** E0000h).

```
Input file: SIEVE.ROM
Configuration file: SIEVE.CFG
Created by Paradigm LOCATE 5.00b on Sat Jul 29 15:22:41 1995
Command line options: C:\LOCATE\LOCATE.EXE sieve

Memory Address Map for Program SIEVE

Start     Stop      Length  Segment       Class
000400H   000401H   00002H  _DATA         DATA
000402H   000403H   00002H  XIHEAP        DATA
.......
.......
000404H   000C03H   00800H  C_COMMON      BSS
000C10H   00100FH   00400H  _STACK        STACK
0E0000H   0E0156H   00157H  _TEXT         CODE
0E0160H   0E0160H   00000H  _BRD          ROMDATA
.......
.......
0E0164H   0E0164H   00000H  HDR           ROMDATA
0E0170H   0E017FH   00010H  _ERD          ENDROMDATA
0F7FF0H   0F7FF8H   00009H  ??CPUINIT     ??LOCATE

Entry point:   F7FF:0000
Initial stack: 00C1:0400
```

Figure 12 **- sieve.loc** File

3**. Downloading a standalone program**: Downloading a standalone program is exactly the same as downloading PDREM. See the section “Downloading PDREM to SD186EM the board” for detailed information.