Creating an EPROM Program

Ali Zahedi

Introduction

Cost, flexibility and performance are key issues when considering system design. Cost is minimized and flexibility is maximized when using dynamic RAM for main memory. However, better performance and greater reliability is achieved when ROM is used. Depending on the specific application, the optimum memory complement is a compromise between RAM and ROM. Within real time systems (embedded applications), as both performance and reliability are of great importance, usually flexibility is sacrificed and systems are developed mainly on ROM.

The method by which most programs are transferred to ROM (PROM, EPROM, EEPROM) is via a specific ROM programmer. The code that is down loaded into a ROM is usually known as "firmware". The major differences between a firmware coded application and a commonly encountered .com or .exe application are the lack of code relocatability and the difficulty of changing parts of the firmware code once it has been transferred to the ROM. As a result, embedded programs are usually written so that they do not require any changes to the code once burned into ROM. It should be mentioned that most embedded programs today use a combination of DRAM for data and volatile storage and ROM for program and constant storage. A common and classic example of this type of system is a Laserjet Printer where the program running the Printer resides in ROM and the bit **m**ap of the page to print is loaded and processed in DRAM.

The purpose of this document is specifically to explain how to create an Assembly Language (or TurboC) program that can work in the 8088 environment. Specific issues addressed by this document include:

- how to create the RESTART jump code
- how to initialize segment registers
- how to access RAM and use variables in RAM, and
- how to burn an EPROM for use in an embedded system.

Initial Assumptions

The first step in structuring a program for burning into a ROM is to determine the ROM size, the memory addressing space and the restart address of the processor. In the following example a 2732 EPROM is used. This EPROM has 32 Kbits(4KX8) of memory. We will make the following assumptions (8086-88 Microprocessor):

- The startup address of the 8086/88 is FFFF0H
- The EPROM has an address range of 000-FFFh
- The EPROM must be mapped to the address range FF000-FFFFFh so that it is in the restart

address space of the 8086/88.

Assembly language programming

The most direct approach when using assembly language is to create an .EXE file with correct restart addressing. Unfortunately, an EXE program has header and trailer bits that specify how the program is loaded into memory under control of the DOS operating system. This additional code is useful, for example, when someone interrupts a program by hitting Ctrl-C and control is returned to the operating system. When we want to program an EPROM there is no need for these operating system or loader links because there is no operating system. Thus the control bits must be removed. This is done by simply converting an EXE file to a Binary file (.BIN).

DOS has a utility program named EXE2BIN.EXE which is used to convert *fname.EXE* files to binary format *fname.BIN files*. The command line prompt is as follows:

EXE2BIN [drivel:] [path1] inputfile [[drive2:] [path2] outputfile)

where:

inputfile Specifies the EXE file to be converted.

output-file Specifies the binary file to create(the same name with a bin extension is used if not specified)

Burning An EPROM

Once your assembly language program is in a binary format, it can be loaded into an EPROM. For programming an EPROM the following steps are required:

1. Write your program in assembly language. Make sure you include a hook (Jump Command) to intercept the restart address of the microprocessor so that your program starts up correctly.
2. Assemble and link the file to generate an EXE program(use TLINK.EXE).
3. Use EXE2BIN.EXE to convert the program to a binary format.
4. Use an EPROM programmer to download the binary file into the EPROM.
5. There are many options for creating a program. These include using a compiler for high level languages like C or Pascal, or using low assembly language and an assembler. C compilers are more common and. some compilers from Microsoft(QuickC, MS C/C++) and Borland(TurboC, Borland C) are readily available and easy to use. If one wants to use assembly language, Microsoft MASM and Borland TASM are common.

Sample ASM Program

Below is a simple ASM program. This program illustrates how to initialize segment registers, how to hook into the 8086/88 restart vector, and how to access both RAM (assumed to be in low memory) and ROM (assumed to be the last 4Kbytes in high memory).

```
;ASM Example
;assume that the EPROM is mapped starting at FF000h
;and is 4Kbytes wide.(2732 type of device)
;
;assume that there is SRAM at 00000-OOOFFh (256 bytes)
;
;
;------------------------------------------------------*
;                  Packaging Program                   *
;This program is designed for bottle packaging in a    *
;factory.  The bottles pass across a sensor and for    *
;each bottle the sensor sends a signal to one of the   *
;input ports of the microprocessor (8Ox88).  The       *
;microprocessor checks to see if number of bottles has *
;reached 16.  If so, the program sends another signal  *
;to the packaging machine.                             *
;------------------------------------------------------*

.Model small                       ;64K Max. size

;-------------------------------------------------------
;Declare some useful constants.

inport     equ     3F8h            ;Input port address
outport    equ     2F8h            ;Output port address
MaxBot     equ     10h             ;set maximum number
                                   ;of bottles to 16

;-------------------------------------------------------
; DO NOT use a data segment.  All fixed data that you
; want in ROM can be put in the code segment using the
; same compiler directives you used in the data segment
;-------------------------------------------------------
.CODE
        ORG     0100h              ;put permanent data here.
                                   ;start,of fixed data(rather arbitrary,
                                   ;only must not be at the high end of
                                   ;EPROM). The ORG statement MUST FOLLOW
                                   ;the segment declaration.

tblstrt:  db           0FFh      ;Just defining some useless data.
          db           0EEh
          db           0DDh
tblend:   db           0A5h      ;End of useless data

;---------------------------------------------------------
;This is where the code "really" starts!

          ORG       0200h            ;keep data and code areas separate

init:     nop
          mov          ax,0FF10h     ;init DS register, for ease data request
                                     ;to start at FF100h absolute

          mov          dx,ax
          mov          ax,0000h      ;init SS register (start of SRAM)
```

```
          mov            ss,ax
          mov            sp,00FFh      ;init SP register (TOP of SRAM)

          mov            ax,0          ;initialize bottle flag
          mov            cx,MaxBot     ;initialize number of passed bottle

start:    nop                          ;
          mov            dx,inport     ;set address for input command
          in             ax,dx         ;read from input port
          cmp            ax,0          ;if no bottle arrived
          je             start         ;wait for one
                                       ;if bottle arrived increment bottle number
                                       ;if number of bottles < 16
          loop           start         ;wait for another one
          mov            dx,outport    ;if number of bottles = 16
          mov            ax,0FFh       ;command to packer
          out            dx,ax         ;then issue the command for packaging
          mov            cx,MaxBot     ;initializing number of passed bottle
          jmp            start         ;START again

;***********************************************************************
;***********************************************************************
;EPROM goes from XX000-XXFFFh. But, because of memory mapping,
;the XX is actually Ffh address. The jump must bea ;near jump (2 byte,
;relative address) so that the jump is relocatable. Otherwise
;a 4byte absolute address is put in.
;
org       0FF0h          ;restart address

startep:  nop
          cli            ;make sure interrupts are OFF!
          jmp   init     ;jump to start of program
          end
```

Detailed Assembler MAP

```
0000                 .model small

0000                 .data
                     org 0100h                          ;MUST FOLLOW the
                                                        ;.data declaration
       =03F8                           inport equ 3f8h
       =02F8                           outport equ 2f8h
       =0010                           MaxBot equ 10h


0100                 .code
                     org 0200h                          ;MUST FOLLOW the
                                                        ;.code declaration
0200   90            init:             nop
0201   B8 FF10                         mov  ax,0ff10h
0204   8B D0                           mov  dx,ax
0206   B8 0000                         mov  ax,0h
0209   8E D0                           mov  ss,ax
```

```
020B   BC 00FF                  mov  sp,00ffh
020E   B8 0000                  mov  ax,0
0211   B9 0010                  mov  cx,MaxBot
0214   90        start:         nop
0215   BA 03F8                  mov  dx,inport
0218   ED                       in   ax,dx
0219   3D 0000                  cmp  ax,0
021C   74 F6                    je   start
021E   E2 F4                    loop start
0220   BA 02F8                  mov  dx,outport
0223   B8 00FF                  mov  ax, 0ffh
0226   EF                       out  dx,ax
0227   B9 0010                  mov  cx,MaxBot
022A   EB E8                    jmp  start
                 org 0ff0h
0FF0   90        startep:nop
0FF1   FA                       cli
0FF2   E9 F20B                  jmp  init
                                end
```