

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 map 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-FFFFFFh 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 [drive1:] [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 00000-000FFh (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 (80x88).  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 be a ;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

```

1      0000          .model small
2
3      0000          .data
4      org 0100h          ;MUST FOLLOW the
                        ;.data declaration
5          =03F8          inport equ 3f8h
6          =02F8          outport equ 2f8h
7          =0010          MaxBot equ 10h
8
9
10     0100          .code
11     org 0200h          ;MUST FOLLOW the
                        ;.code declaration
12     0200  90          init:      nop
13     0201  B8 FF10      mov     ax,0ff10h
14     0204  8B D0        mov     dx,ax
15     0206  B8 0000      mov     ax,0h
16     0209  8E D0        mov     ss,ax

```

```
17      020B   BC 00FF          mov  sp,00ffh
18
19      020E   B8 0000          mov  ax,0
20      0211   B9 0010          mov  cx,MaxBot
21
22      0214   90              start:  nop
23      0215   BA 03F8          mov  dx,inport
24      0218   ED              in   ax,dx
25      0219   3D 0000          cmp  ax,0
26      021C   74 F6              je   start
27
28      021E   E2 F4              loop start
29      0220   BA 02F8          mov  dx,outport
30      0223   B8 00FF          mov  ax,0ffh
31      0226   EF              out  dx,ax
32      0227   B9 0010          mov  cx,MaxBot
33      022A   EB E8              jmp  start
34
35
36
37
38      0FF0   90              org 0ff0h
39      0FF1   FA              startep:nop
40      0FF2   E9 F20B          cli
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
```