```
1   #include "all03.h"
2
3   #define _VERSION_           "1.04"
4
5   //----------------------------
6   #define PIC_LOAD_CFG        0x00        // (16<-) switch to 0x2000 (0x4000)
7   #define PIC_INCREMENT       0x06        // next address
8
9   #define PIC_LOAD_CODE       0x02        // (16<-)
10  #define PIC_READ_CODE       0x04        // (->16)
11
12  #define PIC_START_PROG      0x08        // some devices ERASE & PROGRAM !
13
14  #define PIC_PROG_ONLY       0x18        //
15  #define PIC_END_PROG        0x0E        //
16
17  // Devices with ERASE function
18  //----------------------------
19  #define PIC_CHIP_ERASE      0x1F        // Chip Erase TYPE=1
20
21  #define PIC_CODE_ERASE      0x09        // Single mem Erase TYPE=1
22
23  #define PIC_BULK_SETUP1     0x01        // Erase Setup TYPE=0
24  #define PIC_BULK_SETUP2     0x07        // ChipErase if ADDR=2007
25
26  // Flash chips only
27  //----------------------------
28  #define PIC_LOAD_DATA       0x03        // (16<-)
29  #define PIC_READ_DATA       0x05        // (->16)
30
31  #define PIC_DATA_ERASE      0x0B        // works only if not protected
32
33  // Segment names
34  //----------------------------
35  #define CFG_SEG             PIC_LOAD_CFG
36  #define DATA_SEG            PIC_LOAD_DATA
37  #define CODE_SEG            PIC_LOAD_CODE
38
39  //----------------------------
40  int _MCLR, _DATA, _CLK, _PGM, _LVP;
41  int _VCC1, _VCC2, _GND1, _GND2;
42  int _UB, _VPP;
43
44  struct DEV {
45      char name[20];
46      unsigned Size, DataSize;    // ROM & EEPROM sizes in bytes(!)
47      char mclr,dat,clk, lvp;     // MCLR, DATA, CLOCK and LVP pin numbers (*)
48      char vcc1,vcc2, gnd1,gnd2;  // Pins numbers for VCC(s) and GND(s)    (*)
49      int UB, UBmin, UBmax;       // Norm, min. & max. Vcc
50      int VPP, VPP1st;            // VPP value, apply Vpp **BEFORE** Vcc
51      char erase_type;            // if electrically erasable, type of erase
52      char NumProgBytes;          // Number of words to load for 1 prog cycle
53      char prog_type;             // prog pulse type
54      unsigned cfg_mask;          // implemented bits in CONFIG word
55  };
56                                  // (*) Pin numbers on 40pin socket !!!
57  // erase types =
58  // 0 : device has no erase function
59  // 1 : LOAD_XXX with 3FFF, BULK_SETUP, START_PROG, BULK_SETUP
60  // 2 : XXX_ERASE, START_PROG
61  // 3 : LOAD_XXX with 3FFF, XXX_ERASE, START_PROG (all cmds with appended loads)
62
63  // prog types =
64  // 0 : max. 25 * (START_PROG / wait 100us / END_PROG) + 3*n postpulses
65  // 1 : 1 PROG_ONLY pulse, then 4ms delay
66  // 2 : 1 START_PROG pulse, then 8ms delay
67  // 3 : 1 PROG_ONLY pulse, then 20ms delay, END_PROG
68
69  struct DEV PIC_devs[] = {
70  //          TYPE        ROMSIZE EESIZE mclr dat clk lvp vcc vcc gnd gnd Ub  UbL UbH
    Vpp 1st ERA NUM PRG MASK
71  /* 0*/  { "PIC16C554",  0x0400, 0x0000, 15, 24, 23, 0,  25, 0,  16, 27, 50, 20, 65,
```

```
    120,0,  0,  1,  0,  0x3F3F },
 72 /* 1*/  { "PIC16C557",  0x1000, 0x0000, 34, 31, 30, 0,  8,  0,  10, 33, 50, 20, 65,
    120,0,  0,  1,  0,  0x3F3F },
 73 /* 2*/  { "PIC16C558",  0x1000, 0x0000, 15, 24, 23, 0,  25, 0,  16, 27, 50, 20, 65,
    120,0,  0,  1,  0,  0x3F3F },
 74 /* 3*/  { "PIC16C710",  0x0800, 0x0000, 15, 24, 23, 0,  25, 0,  16, 0,  50, 25, 55,
    120,0,  0,  1,  0,  0x001F },
 75 /* 4*/  { "PIC16C71",   0x1000, 0x0000, 15, 24, 23, 0,  25, 0,  16, 0,  50, 25, 55,
    120,0,  0,  1,  0,  0x001F },
 76 /* 5*/  { "PIC16C711",  0x1000, 0x0000, 15, 24, 23, 0,  25, 0,  16, 0,  50, 25, 55,
    120,0,  0,  1,  0,  0x001F },
 77 /* 6*/  { "PIC16C715",  0x2000, 0x0000, 15, 24, 23, 0,  25, 0,  16, 0,  50, 25, 55,
    120,0,  0,  1,  0,  0x001F },
 78 /* 7*/  { "PIC16F870",  0x1000, 0x0040, 7,  34, 33, 0,  26, 0,  14, 25, 50, 25, 55,
    120,0,  1,  1,  1,  0x3BFF },
 79 /* 8*/  { "PIC16F871",  0x1000, 0x0040, 1,  40, 39, 0,  11, 32, 12, 31, 50, 25, 55,
    120,0,  1,  1,  1,  0x3BFF },
 80 /* 9*/  { "PIC16F872",  0x1000, 0x0040, 7,  34, 33, 0,  26, 0,  14, 25, 50, 25, 55,
    120,0,  1,  1,  1,  0x3BFF },
 81 /*10*/  { "PIC16F873",  0x2000, 0x0080, 7,  34, 33, 0,  26, 0,  14, 25, 50, 25, 55,
    120,0,  1,  1,  1,  0x3BFF },
 82 /*11*/  { "PIC16F873A", 0x2000, 0x0080, 7,  34, 33, 0,  26, 0,  14, 25, 50, 20, 55,
    120,0,  2,  8,  2,  0x3BFF },
 83 /*12*/  { "PIC16F874",  0x2000, 0x0080, 1,  40, 39, 0,  11, 32, 12, 31, 50, 25, 55,
    120,0,  1,  1,  1,  0x3BFF },
 84 /*13*/  { "PIC16F874A", 0x2000, 0x0080, 1,  40, 39, 0,  11, 32, 12, 31, 50, 20, 55,
    120,0,  2,  8,  2,  0x3BFF },
 85 /*14*/  { "PIC16F876",  0x4000, 0x0100, 7,  34, 33, 0,  26, 0,  14, 25, 50, 25, 55,
    120,0,  1,  1,  1,  0x3BFF },
 86 /*15*/  { "PIC16F876A", 0x4000, 0x0100, 7,  34, 33, 0,  26, 0,  14, 25, 50, 20, 55,
    120,0,  2,  8,  2,  0x3BFF },
 87 /*16*/  { "PIC16F877",  0x4000, 0x0100, 1,  40, 39, 0,  11, 32, 12, 31, 50, 25, 55,
    120,0,  1,  1,  1,  0x3BFF },
 88 /*17*/  { "PIC16F877A", 0x4000, 0x0100, 1,  40, 39, 0,  11, 32, 12, 31, 50, 20, 55,
    120,0,  2,  8,  2,  0x3BFF },
 89 /*18*/  { "PIC16F83",   0x0400, 0x0040, 15, 24, 23, 0,  25, 0,  16, 27, 50, 20, 60,
    120,0,  1,  1,  3,  0x3FFF },
 90 /*19*/  { "PIC16F84",   0x0800, 0x0040, 15, 24, 23, 0,  25, 0,  16, 27, 50, 20, 60,
    120,0,  1,  1,  3,  0x3FFF },
 91 /*20*/  { "PIC16F84A",  0x0800, 0x0040, 15, 24, 23, 0,  25, 0,  16, 27, 50, 20, 60,
    120,0,  1,  1,  2,  0x3FFF },
 92 /*21*/  { "PIC16F627",  0x0800, 0x0080, 15, 24, 23, 21, 25, 0,  16, 27, 50, 30, 55,
    120,1,  3,  1,  2,  0x3DFF },
 93 /*22*/  { "PIC16F627A", 0x0800, 0x0080, 15, 24, 23, 21, 25, 0,  16, 27, 50, 30, 55,
    120,1,  2,  1,  2,  0x21FF },
 94 /*23*/  { "PIC16F628",  0x1000, 0x0080, 15, 24, 23, 21, 25, 0,  16, 27, 50, 30, 55,
    120,1,  3,  1,  2,  0x3DFF },
 95 /*24*/  { "PIC16F628A", 0x1000, 0x0080, 15, 24, 23, 21, 25, 0,  16, 27, 50, 30, 55,
    120,1,  2,  1,  2,  0x21FF },
 96 /*25*/  { "PIC16F648A", 0x2000, 0x0100, 15, 24, 23, 21, 25, 0,  16, 27, 50, 30, 55,
    120,1,  2,  1,  2,  0x21FF }
 97 };
 98
 99 struct {
100     char name[20];
101     int numdevs;
102     struct DEV *dev;
103 } mfr[] = {
104     { "MICROCHIP", sizeof(PIC_devs)/sizeof(struct DEV), PIC_devs }
105 };
106
107 int mfrno = 0, devno = 0;
108
109 #define BUFSIZE     0x8000U
110
111 long addr;
112 long DevStart= 0x0000;
113 long Counter = 0x0000;
114 int last_volt;
115
116 /*                          12345678901234567890123456 */
117 const char *memareas[] = {  "                            ",
```

```
                                "(Program memory)           ",
                                "(Data memory)              ",
                                "(Program & Data memory)    " };
int area = 3;

void dly2u( void )
{
    dly1u();
    dly1u();
}

void dly5u( void )
{
    dly2u();
    dly2u();
    dly1u();
}

void ShowCounter( long val )
{
    struct text_info ti;

    gettextinfo( &ti );

    textattr( ( BLUE << 4 ) + WHITE );
    locate( 5, 73 ); cprintf( "%04lX", val );

    textattr( ti.attribute );
    gotoxy( ti.curx, ti.cury );
}

void ShowConfig( void )
{
    struct text_info ti;
    int i;

    gettextinfo( &ti );

    textattr( ( BLUE << 4 ) + WHITE );
    locate( 8,54 );
    for( i = 0x4000; i < 0x4008; i += 2 )
        cprintf( "%02X ", buffer[i] );

    locate( 9,63 );
    cprintf( "%04X  ID : %04X",
        *(unsigned*)(buffer+0x400E), i = *(unsigned*)(buffer+0x400C) );

    switch( i & 0x3FE0 )    // mask off revision number (last 5bits)
    {
    case 0x0D00:    if( devno != 7 ) i = 1; break;  // PIC16F870
    case 0x0D20:    if( devno != 8 ) i = 1; break;  // PIC16F871
    case 0x08E0:    if( devno != 9 ) i = 1; break;  // PIC16F872
    case 0x0960:    if( devno != 10) i = 1; break;  // PIC16F873
    case 0x0E40:    if( devno != 11) i = 1; break;  // PIC16F873A
    case 0x0920:    if( devno != 12) i = 1; break;  // PIC16F874
    case 0x0E60:    if( devno != 13) i = 1; break;  // PIC16F874A
    case 0x09E0:    if( devno != 14) i = 1; break;  // PIC16F876
    case 0x0E00:    if( devno != 15) i = 1; break;  // PIC16F876A
    case 0x09A0:    if( devno != 16) i = 1; break;  // PIC16F877
    case 0x0E20:    if( devno != 17) i = 1; break;  // PIC16F877A

    case 0x07A0:    if( devno != 21) i = 1; break;  // PIC16F627
    case 0x1040:    if( devno != 22) i = 1; break;  // PIC16F627A
    case 0x07C0:    if( devno != 23) i = 1; break;  // PIC16F628
    case 0x1060:    if( devno != 24) i = 1; break;  // PIC16F628A

    case 0x1100:    if( devno != 25) i = 1; break;  // PIC16F648A

    case 0x3FE0:    break;                          // ID not read yet
    default:        i = 0; break;                   // unknown ID
    }

```

```
190         if( i == 1 )
191         {
192             locate( 0, 77 ); cprintf( "??" );
193         }
194
195         locate( 2, 69 ); cprintf( "%04X", mfr[mfrno].dev[devno].DataSize );
196
197         textattr( ( CYAN<< 4 ) + WHITE );
198         locate( 8, 75 ); cprintf( "OFF" );
199
200         textattr( ti.attribute );
201         gotoxy( ti.curx, ti.cury );
202     }
203
204     void ShowType( void )
205     {
206         if( mfrno < 0 || mfrno > sizeof(mfr) / sizeof(mfr[0]) )
207             mfrno = 0;
208         if( devno < 0 || devno >= mfr[mfrno].numdevs )
209             devno = 0;
210
211         bufsize = mfr[mfrno].dev[devno].Size;
212         BufEnd = bufsize - 1;
213
214         textattr( ( BLUE << 4 ) | WHITE );
215
216         locate( 0,40 ); cprintf( "*Mfr.: %s", mfr[mfrno].name );
217         locate( 0,60 ); cprintf( "*TYPE: %s", mfr[mfrno].dev[devno].name );
218
219         locate( 3,41 ); cprintf( "       end   addr.: %04lX", BufEnd );
220
221         textattr( ( CYAN << 4 ) | WHITE );
222     }
223
224     // Check, if Pin can be pulled LOW constantly
225     static int pinused( int pin )
226     {
227         return( _MCLR == pin || _DATA == pin || _CLK == pin ||
228                 _VCC1 == pin || _VCC2 == pin );
229     }
230
231     void DeviceSetup( void )
232     {
233         _MCLR = mfr[mfrno].dev[devno].mclr;
234         _DATA = mfr[mfrno].dev[devno].dat;
235         _CLK  = mfr[mfrno].dev[devno].clk;
236         _LVP  = mfr[mfrno].dev[devno].lvp;
237
238         _UB   = mfr[mfrno].dev[devno].UB;
239         _VPP  = mfr[mfrno].dev[devno].VPP;
240
241         _VCC1 = mfr[mfrno].dev[devno].vcc1;
242         _VCC2 = mfr[mfrno].dev[devno].vcc2;
243
244         _GND1 = mfr[mfrno].dev[devno].gnd1;
245         _GND2 = mfr[mfrno].dev[devno].gnd2;
246
247         // Can't use Pins 2,3,4,6 and 8 for neither Vop nor Vhh
248         if( _MCLR == 2 || _MCLR == 3 || _MCLR == 4 || _MCLR == 6 || _MCLR == 8 )
249         {
250             errbeep();
251             textattr( ( RED << 4 ) | WHITE );
252             locate( 23, 41 ); cprintf( "Connect socket pin 1 to IC pin %d", _MCLR );
253             textattr( ( CYAN << 4 ) | WHITE );
254             _MCLR = 1;                                  // force using Pin 1
255         }
256
257         // Check, if Pin20 is a pin we can pull LOW all the time
258         if( !pinused( 20 ) )
259             setport( OTHERENID, 0, 0 );          // Pin 20 = GND
260
261         // If not: check, if Pins 11 and 30 can be pulled LOW constantly
```

```
262         else if( !pinused( 11 ) && !pinused( 30 ) )
263             setport( OTHERENID, 0, 1 );          // Pin 11,30 = GND
264
265         // Else emit warning, force adapter usage with Pin20 low
266         else
267         {
268             errbeep();
269             textattr( ( RED << 4 ) | WHITE );
270             locate( 41, 23 ); cprintf( "Use adapter ADP-MPU4" );
271             textattr( ( CYAN << 4 ) | WHITE );
272             setport( OTHERENID, 0, 0 );              // Pin 20 = GND
273         }
274
275         if( mfr[mfrno].dev[devno].DataSize == 0 )
276             area = 0;
277         else
278             area = 3;
279     }
280
281     void power( int voltage, int vpp )
282     {
283         int i;
284
285         // we dont need VHH
286         setdac( VHHID, 0 );                                          // VHH = 0V
287         for( i = 0; i <= 4; ++i ) setport( VHHENID, i, 0 );          // no VHH
288         setport( VHHENCID,0, 0 );                                    // no VHHC
289         setport( VHHENCID,1, 0 );                                    // no VHHC
290
291         if( voltage )
292         {
293             // Set all pins LOW
294             for( i = 1; i <= 40; ++i )
295             {
296                 setpin( i, VOPENID, 0 );             // remove Vpp
297                 setpin( i, TTLID, 0 );               // set low
298             }
299
300             if( !vpp && !_LVP )
301             {
302                 errbeep();
303                 textattr( ( RED << 4 ) | WHITE );
304                 locate( 23, 41 ); cprintf( "ERROR: neither Vpp nor LVP mode" );
305                 textattr( ( CYAN << 4 ) | WHITE );
306             }
307
308             if( !_LVP && _VPP )                      // chip has no LVP mode
309                 vpp = _VPP;                          // or we don't want it...
310
311             if( mfr[mfrno].dev[devno].VPP1st &&      // apply VPP before VCC
312                 vpp )
313             {
314                 setdac( VOPID, vpp );                // raise Vpp to xV
315                 setdac( VCCID, voltage );            // raise Vcc to xV
316                 last_volt = voltage;
317                 delay(100);                          // let stabilize
318
319                 setpin( _MCLR, TTLID, 1 );           // remove reset from MCLR
320                 setpin( _MCLR, VOPENID, 1 );         // apply Vpp to it
321                 dly10u();
322
323                 setpin( _VCC1, TTLID, 1 );           // apply Vcc1 & Vcc2
324                 setpin( _VCC2, TTLID, 1 );
325                 setpin( _VCC1, VCCENID, 1 );
326                 setpin( _VCC2, VCCENID, 1 );
327
328                 dly10m();
329             }
330             else                                     // apply VCC, then VPP
331             {
332                 setdac( VCCID, voltage );            // raise Vcc to xV
333                 setdac( VOPID, vpp );                // raise Vpp to xV
```

```
334             last_volt = voltage;
335             delay( 100 );                           // let stabilize
336
337             setpin( _VCC1, TTLID, 1 );              // apply Vcc1 & Vcc2
338             setpin( _VCC2, TTLID, 1 );
339             setpin( _VCC1, VCCENID, 1 );
340             setpin( _VCC2, VCCENID, 1 );
341             dly10u();
342
343             if( vpp )
344             {
345                 setpin( _MCLR, TTLID, 1 );          // remove reset from MCLR
346                 setpin( _MCLR, VOPENID, 1 );        // .. and apply Vpp to it
347                 dly10u();
348             }
349             else
350             {
351                 setpin( _MCLR, TTLID, 1 );          // remove reset from MCLR
352                 setpin( _LVP, TTLID, 1 );           // activate LVP mode
353             }
354         }
355
356         locate( 2, 75 );
357         textattr( ( BLUE << 4 ) | WHITE );
358         cprintf( "%s", (!vpp && _LVP) ? "LVP" : ( vpp ? "HVP" : "???" ) );
359         textattr( ( CYAN << 4 ) | WHITE );
360
361         delay( 100 );                               // wait to stabilize
362     }
363     else
364     {
365         setpin( _LVP,  TTLID, 0 );                  // remove LVP voltage
366
367         setpin( _MCLR, VOPENID, 0 );                // remove Vpp
368         setpin( _MCLR, TTLID, 0 );                  // and reset chip
369
370         setdac( VOPID, 0 );                         // Vpp = 0V
371         setdac( VCCID, 0 );                         // Vcc = 0V
372
373         setpin( _VCC1, VCCENID, 0 );                // disable Vcc(s)
374         setpin( _VCC2, VCCENID, 0 );
375
376         // Set all pins LOW
377         for( i = 1; i <= 40; ++i )
378             setpin( i, TTLID, 0 );
379     }
380 }
381
382 void pic_command( int c )
383 {
384     int i;
385
386     for( i = 0; i < 6; ++i )            // 6 bits, LSB first
387     {
388         setpin( _CLK, TTLID, 1 ); dly10u();
389         setpin( _DATA, TTLID, c & 1 ); dly10u();
390         c >>= 1;
391         setpin( _CLK, TTLID, 0 ); dly10u();
392     }
393     dly50u();
394 }
395
396 void pic_data( unsigned d )
397 {
398     int i;
399
400     d = ( d & 0x3FFF ) << 1;
401
402     for( i = 0; i < 16; ++i )           // START=L, 14 bits (LSB first), STOP=L
403     {
404         setpin( _CLK, TTLID, 1 ); dly10u();
405         setpin( _DATA, TTLID, d & 1 ); dly10u();
```

```
406             d >>= 1;
407             setpin( _CLK, TTLID, 0 ); dly10u();
408         }
409         dly50u();
410     }
411
412     unsigned pic_read( int seg /* 0: CODE, 1: DATA(EEPROM) */ )
413     {
414         unsigned i, d = 0;
415
416         pic_command( seg ? PIC_READ_DATA : PIC_READ_CODE );
417
418         setpin( _DATA, TTLID, 1 ); dly10u(); // make pin an input and wait a bit
419
420         for( i = 0; i < 16; ++i )
421         {
422             setpin( _CLK, TTLID, 1 ); dly10u();
423             d |= ( getpin( _DATA ) ? 0x8000 : 0 );      // LSB first
424             dly10u();
425             d >>= 1;
426             setpin( _CLK, TTLID, 0 ); dly10u();
427         }
428         dly50u();
429         return d & 0x3FFF;              // Bit0 and 15 where read from HiZ...
430     }
431
432     void reset( int vpp )
433     {
434         power( 0, 0 );
435         delay( 100 );
436         power( last_volt, vpp );
437     }
438
439     void do_increment( int num )
440     {
441         while( num-- )
442         {
443             pic_command( PIC_INCREMENT );
444             if( addr < 0x4200 ) addr += 2;
445             else ++addr;
446             dly50u();
447         }
448     }
449
450     void do_load( int seg, unsigned val )
451     {
452         pic_command( seg ); pic_data( val );
453     }
454
455     int do_prog1( unsigned lastval )
456     {
457         int num, end = 25, step = 1;
458         unsigned mask;
459
460         mask = mfr[mfrno].dev[devno].cfg_mask;
461
462         switch( mfr[mfrno].dev[devno].prog_type )
463         {
464         case 0:     // max. 25 pulses of 100us + 3*n additional pulses
465
466             for( num = 1; num != end; num += step )
467             {
468                 pic_command( PIC_START_PROG );
469                 dly100u();
470                 pic_command( PIC_END_PROG );
471
472                 if( step == 1 )
473                 {
474                     if( addr >= 0x4200 )        // DATA
475                     {
476                         if( ( pic_read( 1 ) & 0xFF ) == lastval )
477                         {
```

```
                        num *= 3; end = 0; step = -1;
                    }
                }
                else if( addr == 0x400E )    // CONFIG word
                {
                    if( ( ( pic_read( 0 ) ^ lastval ) & mask ) == 0 )
                    {
                        num *= 3; end = 0; step = -1;
                    }
                }
                else if( pic_read( 0 ) == lastval ) // CODE & ID
                {
                    num *= 3; end = 0; step = -1;
                }
            }
        }
        return num;

    case 1:     // PROG_ONLY, wait 4ms (10ms), no END (program only)
        pic_command( PIC_PROG_ONLY );
        delay(20);
        if( addr >= 0x4200 )
            return( ( pic_read( 1 ) & 0xFF ) == lastval );
        else if( addr == 0x400E )
            return( ( pic_read( 0 ) ^ lastval ) & mask ) == 0;
        else
            return( pic_read( 0 ) == lastval );

    case 2:     // START_PROG, wait 8ms (10ms), no END (erase/program)
        pic_command( PIC_START_PROG );
        delay(20);
        if( addr >= 0x4200 )
            return( ( pic_read( 1 ) & 0xFF ) == lastval );
        else if( addr == 0x400E )
            return( ( pic_read( 0 ) ^ lastval ) & mask ) == 0;
        else
            return( pic_read( 0 ) == lastval );

    case 3:     // PROG_ONLY, wait 20ms, END_PROG
        pic_command( PIC_PROG_ONLY );
        dly20m();
        pic_command( PIC_END_PROG );
        if( addr >= 0x4200 )
            return( ( pic_read( 1 ) & 0xFF ) == lastval );
        else if( addr == 0x400E )
            return( ( pic_read( 0 ) ^ lastval ) & mask ) == 0;
        else
            return( pic_read( 0 ) == lastval );
    }

    return 0;
}

int erase( void )
{
//  reset();

    switch( mfr[mfrno].dev[devno].erase_type )
    {
    case 0: // Ooops, no ERASE function

        return 0;

    case 1:

        switch( area )
        {
        case 0: /* none */
        case 1: /* CODE only */
            do_load( CODE_SEG, 0x3FFF );
            break;

```

```
550             case 2: /* DATA only */
551                 do_load( DATA_SEG, 0x3FFF );
552                 break;
553 
554             case 3: /* CHIP */
555                 do_load( CFG_SEG, 0x3FFF );
556                 do_increment( 7 );          // advance to 0x2007 = CFG WORD
557                 break;
558             }
559 
560             pic_command( PIC_BULK_SETUP1 );
561             pic_command( PIC_BULK_SETUP2 );
562             pic_command( PIC_START_PROG ); dly10m();
563             pic_command( PIC_BULK_SETUP1 );
564             pic_command( PIC_BULK_SETUP2 );
565             break;
566 
567         case 2:
568 
569             switch( area )
570             {
571             case 0: /* none */
572             case 1: /* CODE only */
573                 pic_command( PIC_CODE_ERASE );
574                 break;
575 
576             case 2: /* DATA only */
577                 pic_command( PIC_DATA_ERASE );
578                 break;
579 
580             case 3: /* CHIP */
581                 pic_command( PIC_CHIP_ERASE );
582                 break;
583             }
584             pic_command( PIC_START_PROG );
585             dly10m();
586             break;
587 
588         case 3:
589 
590             switch( area )
591             {
592             case 0: /* none */
593             case 1: /* CODE only */
594 
595                 do_load( CODE_SEG, 0x3FFF );
596                 pic_command( PIC_CODE_ERASE );
597                 pic_command( PIC_START_PROG );
598                 dly10m();
599                 break;
600 
601             case 2: /* DATA only */
602 
603                 do_load( DATA_SEG, 0x3FFF );
604                 pic_command( PIC_DATA_ERASE );
605                 pic_command( PIC_START_PROG );
606                 dly10m();
607                 break;
608 
609             case 3: /* CHIP */
610 
611                 do_load( CFG_SEG, 0x3FFF );
612                 do_increment( 7 );          // advance to 0x2007 = CFG WORD
613                 pic_command( PIC_BULK_SETUP1 );
614                 pic_command( PIC_BULK_SETUP2 );
615                 pic_command( PIC_START_PROG ); dly20m();
616                 pic_command( PIC_BULK_SETUP1 );
617                 pic_command( PIC_BULK_SETUP2 );
618 
619                 reset( _VPP );
620 
621                 do_load( CODE_SEG, 0x3FFF );
```

```
622                 pic_command( PIC_CODE_ERASE );
623                 pic_command( PIC_START_PROG );
624                 dly10m();
625
626                 do_load( DATA_SEG, 0x3FFF );
627                 pic_command( PIC_DATA_ERASE );
628                 pic_command( PIC_START_PROG );
629                 dly10m();
630
631                 break;
632             }
633             break;
634         }
635         return 1;
636     }
637
638     int check( void )       // BLANK check, return 1 on success
639     {
640         unsigned end, val;
641
642         if( ( end = mfr[mfrno].dev[devno].DataSize ) != 0 )
643             end += 0x4200;
644         else
645             end = 0x4010;
646
647         for( addr = BufStart; addr < end; )
648         {
649             if( ( addr & 0xFF ) == 0 ) ShowCounter( addr );
650
651             if( addr < 0x4200 )                 // CODE & CONFIG
652             {
653                 val = pic_read(0) ^ 0x3FFF;
654
655                 if( addr == 0x400E )
656                     val &= mfr[mfrno].dev[devno].cfg_mask;
657
658                 if( val )
659                 {
660                     ShowCounter( addr );
661                     return 0;
662                 }
663             }
664             else if( ( pic_read(1) & 0xFF ) != 0xFF )   // DATA
665             {
666                 ShowCounter( addr );
667                 return 0;
668             }
669
670             do_increment( 1 );
671
672             if( addr == bufsize )               // continue in CONFIG space
673             {
674                 do_load( CFG_SEG, 0x3FFF );     // goto 0x2000 (0x4000)
675                 addr = 0x4000;
676             }
677
678             else if( addr == 0x4008 )           // ID locations checked
679                 do_increment( 3 );              // advance to CFG WORD location
680
681             else if( addr == 0x4010 )           // config space checked,
682             {
683                 reset( _VPP );                  // reset device and check DATA
684                 addr = 0x4200;
685             }
686
687         }
688         ShowCounter( addr );
689         return 1;
690     }
691
692     int read_verify( int verify )
693     {
```

```
694         unsigned val, end, res = 1;
695
696         if( ( end = mfr[mfrno].dev[devno].DataSize ) != 0 )
697             end += 0x4200;
698         else
699             end = 0x4010;
700
701         for( addr = BufStart; addr < end; )
702         {
703             if( ( addr & 0xFF ) == 0 ) ShowCounter( addr );
704
705             if( verify )
706             {
707                 if( addr < 0x4200 )
708                 {
709                     val = buffer[addr];
710                     val |= ( buffer[addr+1] & 0x3F ) << 8;
711                     val ^= pic_read(0);
712
713                     if( addr == 0x400E )
714                         val &= mfr[mfrno].dev[devno].cfg_mask;
715
716                     if( val )
717                     {
718                         ShowCounter( addr );
719                         res = 0;
720                         break;
721                     }
722                 }
723                 else if( ( pic_read(1) & 0xFF ) != buffer[addr] )
724                 {
725                     ShowCounter( addr );
726                     res = 0;
727                     break;
728                 }
729             }
730             else
731             {
732                 if( addr < 0x4200 )
733                 {
734                     val = pic_read(0);
735                     Chks += ( buffer[addr] = val & 0xFF );
736                     Chks += ( buffer[addr+1] = ( val >> 8 ) & 0x3F );
737                 }
738                 else
739                     Chks += ( buffer[addr] = pic_read(1) & 0xFF );
740             }
741
742             do_increment( 1 );
743
744             if( addr == bufsize )               // continue in CONFIG space
745             {
746                 do_load( CFG_SEG, 0x3FFF );     // goto 0x2000 (0x4000)
747                 addr = 0x4000;
748             }
749
750             else if( addr == 0x4008 )           // ID locations done
751                 do_increment( verify ? 3 : 2 ); // advance to DEVID or CFG word
752
753             else if( addr == 0x4010 )           // config space done,
754             {
755                 reset( _VPP );                  // reset device and check DATA
756                 addr = 0x4200;
757             }
758         }
759         ShowCounter( addr );
760
761         return res;
762     }
763
764     int program( void )
765     {
```

```
766         int i, n, val;
767         unsigned end;
768
769         if( ( end = mfr[mfrno].dev[devno].DataSize ) != 0 )
770             end += 0x4200;
771         else
772             end = 0x4010;
773
774         n = mfr[mfrno].dev[devno].NumProgBytes;
775
776         for( addr = BufStart; addr < end; )
777         {
778             if( ( addr & 0xFF ) == 0 ) ShowCounter( addr );
779
780             i = n;
781
782             while( i-- )
783             {
784                 if( addr < 0x4200 )             // CODE & CONFIG space
785                 {
786                     val = buffer[addr];
787                     val |= ( buffer[addr+1] & 0x3F ) << 8;
788                     do_load( CODE_SEG, val );
789                 }
790                 else
791                 {
792                     val = buffer[addr];
793                     do_load( DATA_SEG, val );
794                 }
795
796                 if( i ) do_increment( 1 );      // don't increment after n-th load
797             }
798
799             if( !do_prog1( val ) )
800                 return 0;
801
802             do_increment( 1 );
803
804             if( addr == bufsize )               // CODE space done
805             {
806                 do_load( CFG_SEG, 0x3FFF );     // goto 0x2000 (0x4000)
807                 addr = 0x4000;
808             }
809
810             else if( addr == 0x4008 )           // ID locations checked
811                 do_increment( 3 );              // advance to CFG WORD location
812
813             else if( addr == 0x4010 )           // CONFIG space done,
814             {
815                 reset( _VPP );                  // reset device and do DATA
816                 addr = 0x4200;
817             }
818         }
819         ShowCounter( addr );
820         return 1;
821     }
822
823     int config( void )
824     {
825         int i, n, val;
826
827         if( ( n = mfr[mfrno].dev[devno].NumProgBytes ) > 4 )
828             n = 4;
829
830         do_load( CFG_SEG, 0x3FFF );     // goto 0x2000 (0x4000)
831
832         for( addr = 0x4000; addr < 0x4010; )
833         {
834             i = n;
835             while( i-- )
836             {
837                 val = buffer[addr];
```

```
838             val |= ( buffer[addr+1] & 0x3F ) << 8;
839             do_load( CODE_SEG, val );
840             if( i ) do_increment( 1 );  // don't increment after n-th load
841         }
842
843         if( !do_prog1( val ) )
844             return 0;
845
846         do_increment( 1 );
847
848         if( addr == 0x4008 )
849             do_increment( 3 );      // advance to CFG word
850     }
851     return 1;
852 }
853
854 int flash_check( void )
855 {
856     int done;
857
858     textattr( ( CYAN << 4 ) | WHITE );
859     _window( 12,40, 23,79 );
860
861     textattr( ( BLUE << 4 ) | WHITE );
862     locate( 12, 45 ); cprintf( "   BLANK CHECK device:" );
863
864     for(;;)
865     {
866         textattr( ( CYAN << 4 ) | WHITE );
867         locate( 13, 41 ); cprintf( "Ready to check (Y/<CR>)? " );
868
869         for( done = 0; !done; )
870         {
871             switch( getch() )
872             {
873             case 0:
874                 getch();
875                 break;
876
877             case '\n':
878             case '\r':
879             case 0x1B:
880                 return;
881
882             case 'y':
883             case 'Y':
884                 done = 1;
885             }
886         }
887
888         clscrn( 14,41, 22,78 );
889
890         setport( USERBITS, 0, 0 );
891
892         textattr( BLINK | ( LIGHTGREEN << 4 ) | WHITE );
893         locate( 14, 41 ); cprintf( "Blank checking now... " );
894
895         power( _UB, _VPP ); done = check(); power( 0, 0 );
896
897         textattr( ( CYAN << 4 ) | WHITE );
898         locate( 14, 41 ); cprintf( "Blank checking now... " );
899
900         locate( 15, 41 );
901         if( done )
902         {
903             ShowCounter( BufEnd );
904             putchar( 7 );
905             cprintf( " OK !" );
906             setport( USERBITS, 0, 8 );
907         }
908         else
909         {
```

```
910                 errbeep();
911                 textattr( ( RED << 4 ) | WHITE );
912                 cprintf( "Blank check error at %04lX", addr );
913                 textattr( ( CYAN << 4 ) | WHITE );
914             }
915         }
916     }
917
918     void flash_program( void )
919     {
920         int done;
921
922         textattr( ( CYAN << 4 ) | WHITE );
923         _window( 12,40, 23,79 );
924
925         textattr( ( BLUE << 4 ) | WHITE );
926         locate( 12, 45 ); cprintf( "   PROGRAM :" );
927
928         for(;;)
929         {
930             textattr( ( CYAN << 4 ) | WHITE );
931             locate( 13, 41 ); cprintf( "Ready to program (Y/<CR>)? " );
932
933             for( done = 0; !done; )
934             {
935                 switch( getch() )
936                 {
937                 case 0:
938                     getch();
939                     break;
940
941                 case '\n':
942                 case '\r':
943                 case 0x1B:
944                     return;
945
946                 case 'y':
947                 case 'Y':
948                     done = 1;
949                 }
950             }
951
952             clscrn( 14,41, 22,78 );
953
954             setport( USERBITS, 0, 0 );
955
956             textattr( BLINK | ( LIGHTGREEN << 4 ) | WHITE );
957             locate( 14, 41 ); cprintf( "Programming now... " );
958
959             power( _UB, _VPP ); done = program(); power( 0, 0 );
960
961             textattr( ( CYAN << 4 ) | WHITE );
962             locate( 14, 41 ); cprintf( "Programming now... " );
963
964             locate( 15, 41 );
965             if( done )
966             {
967                 ShowCounter( BufEnd );
968                 putchar( 7 );
969                 setport( USERBITS, 0, 8 );
970                 cprintf( " OK !" );
971             }
972             else
973             {
974                 errbeep();
975                 textattr( ( RED << 4 ) | WHITE );
976                 cprintf( "Program error ! at %04lX", addr );
977                 textattr( ( CYAN << 4 ) | WHITE );
978             }
979         }
980     }
981
```

```
 982  void flash_config( void )
 983  {
 984      int done;
 985
 986      textattr( ( CYAN << 4 ) | WHITE );
 987      _window( 12,40, 23,79 );
 988
 989      textattr( ( BLUE << 4 ) | WHITE );
 990      locate( 12, 42 ); cprintf( " Program ID & CFG & protect bits: " );
 991
 992      for(;;)
 993      {
 994          textattr( ( CYAN << 4 ) | WHITE );
 995          locate( 13, 41 ); cprintf( "Ready to program (Y/<CR>)? " );
 996
 997          for( done = 0; !done; )
 998          {
 999              switch( getch() )
1000              {
1001              case 0:
1002                  getch();
1003                  break;
1004
1005              case '\n':
1006              case '\r':
1007              case 0x1B:
1008                  return;
1009
1010              case 'y':
1011              case 'Y':
1012                  done = 1;
1013              }
1014          }
1015
1016          clscrn( 14,41, 22,78 );
1017
1018          setport( USERBITS, 0, 0 );
1019
1020          textattr( BLINK | ( LIGHTGREEN << 4 ) | WHITE );
1021          locate( 14, 41 ); cprintf( "Programming now... " );
1022
1023          power( _UB, _VPP ); done = config(); power( 0, 0 );
1024
1025          textattr( ( CYAN << 4 ) | WHITE );
1026          locate( 14, 41 ); cprintf( "Programming now... " );
1027
1028          locate( 15, 41 );
1029          if( done )
1030          {
1031              ShowCounter( BufEnd );
1032              putchar( 7 );
1033              setport( USERBITS, 0, 8 );
1034              cprintf( " OK !" );
1035          }
1036          else
1037          {
1038              errbeep();
1039              textattr( ( RED << 4 ) | WHITE );
1040              cprintf( "Program error ! at %04lX", addr );
1041              textattr( ( CYAN << 4 ) | WHITE );
1042          }
1043      }
1044  }
1045
1046  void flash_read( void )
1047  {
1048      int done;
1049
1050      textattr( ( CYAN << 4 ) | WHITE );
1051      _window( 12,40, 23,79 );
1052
1053      textattr( ( BLUE << 4 ) | WHITE );
```

```
1054        locate( 12, 45 ); cprintf( "   READ to buffer :" );
1055
1056        for(;;)
1057        {
1058            textattr( ( CYAN << 4 ) | WHITE );
1059            locate( 13, 41 ); cprintf( "Ready to start (Y/Even/Odd/<CR>)? " );
1060
1061            for( done = 0; !done; )
1062            {
1063                switch( getch() )
1064                {
1065                case 0:
1066                    getch();
1067                    break;
1068
1069                case '\n':
1070                case '\r':
1071                case 0x1B:
1072                    return;
1073
1074                case 'y':
1075                case 'Y':
1076                    done = 1;
1077
1078                case 'e':
1079                case 'E':
1080                case 'o':
1081                case 'O':
1082                    done = 1;
1083                }
1084            }
1085
1086            clscrn( 14,41, 22,78 );
1087
1088            setport( USERBITS, 0, 0 );
1089
1090            textattr( BLINK | ( LIGHTGREEN << 4 ) | WHITE );
1091            locate( 14, 41 ); cprintf( "Reading now... " );
1092
1093            Chks = 0;
1094            power( _UB, _VPP ); read_verify(0); power( 0, 0 );
1095
1096            textattr( ( CYAN << 4 ) | WHITE );
1097            locate( 14, 41 ); cprintf( "Reading now... " );
1098
1099            locate( 15, 41 );
1100            putchar( 7 );
1101            cprintf( " OK !" );
1102
1103            textattr( ( BLUE << 4 ) | WHITE );
1104            locate( 4, 41 ); cprintf( "       Check Sum  : %04X", Chks );
1105
1106            ShowConfig();
1107        }
1108    }
1109
1110    void flash_verify( void )
1111    {
1112        int done;
1113
1114        textattr( ( CYAN << 4 ) | WHITE );
1115        _window( 12,40, 23,79 );
1116
1117        textattr( ( BLUE << 4 ) | WHITE );
1118        locate( 12, 45 ); cprintf( "   VERIFY with buffer :" );
1119
1120        for(;;)
1121        {
1122            textattr( ( CYAN << 4 ) | WHITE );
1123            locate( 13, 41 ); cprintf( "Ready to verify (Y/Even/Odd/<CR>)? " );
1124
1125            for( done = 0; !done; )
```

```
1126            {
1127                switch( getch() )
1128                {
1129                case 0:
1130                    getch();
1131                    break;
1132
1133                case '\n':
1134                case '\r':
1135                case 0x1B:
1136                    return;
1137
1138                case 'y':
1139                case 'Y':
1140                    done = 1;
1141
1142                case 'e':
1143                case 'E':
1144                case 'o':
1145                case 'O':
1146                    done = 1;
1147                }
1148            }
1149
1150            clscrn( 14,41, 22,78 );
1151
1152            setport( USERBITS, 0, 0 );
1153
1154            textattr( BLINK | ( LIGHTGREEN << 4 ) | WHITE );
1155            locate( 14, 41 ); cprintf( "Verifying now @ VDDmin... " );
1156
1157            power( mfr[mfrno].dev[devno].UBmin, _VPP );
1158            done = read_verify(1);
1159            power( 0, 0 );
1160
1161            textattr( ( CYAN << 4 ) | WHITE );
1162            locate( 14, 41 ); cprintf( "Verifying now @ VDDmin... " );
1163
1164            locate( 15, 41 );
1165            if( done )
1166            {
1167                ShowCounter( BufEnd );
1168                putchar( 7 );
1169                cprintf( " OK !" );
1170                setport( USERBITS, 0, 8 );
1171            }
1172            else
1173            {
1174                errbeep();
1175                textattr( ( RED << 4 ) | WHITE );
1176                cprintf( " VERIFY ERROR ! at %04lX", addr );
1177                textattr( ( CYAN << 4 ) | WHITE );
1178                continue;
1179            }
1180
1181            textattr( BLINK | ( LIGHTGREEN << 4 ) | WHITE );
1182            locate( 16, 41 ); cprintf( "Verifying now @ VDDmax... " );
1183
1184            power( mfr[mfrno].dev[devno].UBmax, _VPP );
1185            done = read_verify(1);
1186            power( 0, 0 );
1187
1188            textattr( ( CYAN << 4 ) | WHITE );
1189            locate( 16, 41 ); cprintf( "Verifying now @ VDDmax... " );
1190
1191            locate( 17, 41 );
1192            if( done )
1193            {
1194                ShowCounter( BufEnd );
1195                putchar( 7 );
1196                cprintf( " OK !" );
1197                setport( USERBITS, 0, 8 );
```

```
1198            }
1199            else
1200            {
1201                errbeep();
1202                textattr( ( RED << 4 ) | WHITE );
1203                cprintf( " VERIFY ERROR ! at %04lX", addr );
1204                textattr( ( CYAN << 4 ) | WHITE );
1205            }
1206        }
1207    }
1208
1209    void flash_erase( void )
1210    {
1211        int done;
1212
1213        textattr( ( CYAN << 4 ) | WHITE );
1214        _window( 12,40, 23,79 );
1215
1216        textattr( ( BLUE << 4 ) | WHITE );
1217        locate( 12, 45 ); cprintf( "  EEPROM Erase:" );
1218
1219        if( !mfr[mfrno].dev[devno].erase_type )
1220        {
1221            errbeep();
1222            textattr( ( RED << 4 ) | WHITE );
1223            locate( 13, 41 ); cprintf( "No ERASE function" );
1224            textattr( ( CYAN << 4 ) | WHITE );
1225
1226            locate( 21, 41 ); cprintf( "press any key to continue" );
1227            if( getch() == 0 ) getch();
1228            return;
1229        }
1230
1231        for(;;)
1232        {
1233            textattr( ( CYAN << 4 ) | WHITE );
1234            locate( 13, 41 ); cprintf( "Ready to erase (Y/<CR>)? " );
1235
1236            for( done = 0; !done; )
1237            {
1238                switch( getch() )
1239                {
1240                case 0:
1241                    getch();
1242                    break;
1243
1244                case '\n':
1245                case '\r':
1246                case 0x1B:
1247                    return;
1248
1249                case 'y':
1250                case 'Y':
1251                    done = 1;
1252                }
1253            }
1254
1255            clscrn( 14,41, 22,78 );
1256
1257            setport( USERBITS, 0, 0 );
1258
1259            textattr( BLINK | ( LIGHTGREEN << 4 ) | WHITE );
1260            locate( 14, 41 ); cprintf( "Erase now... " );
1261
1262            power( _UB, _VPP ); done = erase(); power( 0, 0 );
1263
1264            textattr( ( CYAN << 4 ) | WHITE );
1265            locate( 14, 41 ); cprintf( "Erase now... " );
1266
1267            locate( 15, 41 );
1268            if( done )
1269            {
```

```
1270                putchar( 7 );
1271                cprintf( " OK !" );
1272                setport( USERBITS, 0, 8 );
1273            }
1274            else
1275            {
1276                errbeep();
1277                textattr( ( RED << 4 ) | WHITE );
1278                cprintf( " ERROR ! " );
1279                textattr( ( CYAN << 4 ) | WHITE );
1280
1281                locate( 21, 41 ); cprintf( "press any key to continue" );
1282                if( getch() == 0 ) getch();
1283            }
1284
1285            clscrn( 13,41, 22,78 );
1286        }
1287    }
1288
1289    void flash_auto( void )
1290    {
1291        int done, l;
1292
1293        textattr( ( CYAN << 4 ) | WHITE );
1294        _window( 12,40, 23,79 );
1295
1296        textattr( ( BLUE << 4 ) | WHITE );
1297        locate( 12, 45 ); cprintf( "   AUTO :" );
1298
1299        for(;;)
1300        {
1301            textattr( ( CYAN << 4 ) | WHITE );
1302            locate( 13, 41 ); cprintf( "Ready to start (Y/Even/Odd/<CR>)? " );
1303
1304            for( done = 0; !done; )
1305            {
1306                switch( getch() )
1307                {
1308                case 0:
1309                    getch();
1310                    break;
1311
1312                case '\n':
1313                case '\r':
1314                case 0x1B:
1315                    return;
1316
1317                case 'y':
1318                case 'Y':
1319                    done = 1;
1320
1321                case 'e':
1322                case 'E':
1323                case 'o':
1324                case 'O':
1325                    done = 1;
1326                }
1327            }
1328
1329            clscrn( 14,41, 22,78 );
1330
1331            setport( USERBITS, 0, 0 );
1332
1333            l = 14;
1334
1335            // BLANK CHECK
1336
1337            textattr( BLINK | ( LIGHTGREEN << 4 ) | WHITE );
1338            locate( l, 41 ); cprintf( "Blank checking now... " );
1339
1340            power( _UB, _VPP ); done = check(); power( 0, 0 );
1341
```

```
1342            textattr( ( CYAN << 4 ) | WHITE );
1343            locate( l++, 41 ); cprintf( "Blank checking now... " );
1344
1345            locate( l++, 41 );
1346
1347            if( done )
1348            {
1349                ShowCounter( BufEnd );
1350                cprintf( " OK !" );
1351            }
1352            else
1353            {
1354                errbeep();
1355                textattr( ( RED << 4 ) | WHITE );
1356                cprintf( "Blank check error at %04lX", addr );
1357                textattr( ( CYAN << 4 ) | WHITE );
1358
1359            // ERASE
1360
1361                l -= 2;
1362
1363                clscrn( l, 41, l+1, 78 );
1364
1365                if( !mfr[mfrno].dev[devno].erase_type )
1366                {
1367                    errbeep();
1368                    textattr( ( RED << 4 ) | WHITE );
1369                    cprintf( "No ERASE function" );
1370                    textattr( ( CYAN << 4 ) | WHITE );
1371
1372                    continue;
1373                }
1374
1375                textattr( BLINK | ( LIGHTGREEN << 4 ) | WHITE );
1376                locate( l, 41 ); cprintf( "Erase now... " );
1377
1378                power( _UB, _VPP ); done = erase(); power( 0, 0 );
1379
1380                textattr( ( CYAN << 4 ) | WHITE );
1381                locate( l++, 41 ); cprintf( "Erase now... " );
1382
1383                locate( l++, 41 );
1384
1385                if( done )
1386                    cprintf( " OK !" );
1387                else
1388                {
1389                    errbeep();
1390                    textattr( ( RED << 4 ) | WHITE );
1391                    cprintf( " ERROR" );
1392                    textattr( ( CYAN << 4 ) | WHITE );
1393
1394                    continue;
1395                }
1396            }
1397
1398            // PROGRAM
1399
1400            textattr( BLINK | ( LIGHTGREEN << 4 ) | WHITE );
1401            locate( l, 41 ); cprintf( "Programming now... " );
1402
1403            power( _UB, _VPP ); done = program(); power( 0, 0 );
1404
1405            textattr( ( CYAN << 4 ) | WHITE );
1406            locate( l++, 41 ); cprintf( "Programming now... " );
1407
1408            locate( l++, 41 );
1409            if( done )
1410            {
1411                ShowCounter( BufEnd );
1412                cprintf( " OK !" );
1413            }
```

```
1414            else
1415            {
1416                errbeep();
1417                textattr( ( RED << 4 ) | WHITE );
1418                cprintf( "Program error ! at %04lX", addr );
1419                textattr( ( CYAN << 4 ) | WHITE );
1420
1421                continue;
1422            }
1423
1424            // VERIFY
1425
1426            textattr( BLINK | ( LIGHTGREEN << 4 ) | WHITE );
1427            locate( l, 41 ); cprintf( "VDD max verifying now..." );
1428
1429            power( mfr[mfrno].dev[devno].UBmax, _VPP );
1430            done = read_verify(1);
1431            power( 0, 0 );
1432
1433            textattr( ( CYAN << 4 ) | WHITE );
1434            locate( l++, 41 ); cprintf( "VDD max verifying now..." );
1435
1436            locate( l++, 41 );
1437            if( done )
1438            {
1439                ShowCounter( BufEnd );
1440                putchar( 7 );
1441                cprintf( " OK !" );
1442                setport( USERBITS, 0, 8 );
1443            }
1444            else
1445            {
1446                errbeep();
1447                textattr( ( RED << 4 ) | WHITE );
1448                cprintf( " VERIFY ERROR ! at %04lX", addr );
1449                textattr( ( CYAN << 4 ) | WHITE );
1450                continue;
1451            }
1452
1453            textattr( BLINK | ( LIGHTGREEN << 4 ) | WHITE );
1454            locate( l, 41 ); cprintf( "VDD min verifying now..." );
1455
1456            power( mfr[mfrno].dev[devno].UBmin, _VPP );
1457            done = read_verify(1);
1458            power( 0, 0 );
1459
1460            textattr( ( CYAN << 4 ) | WHITE );
1461            locate( l++, 41 ); cprintf( "VDD min verifying now..." );
1462
1463            locate( l, 41 );
1464            if( done )
1465            {
1466                ShowCounter( BufEnd );
1467                cprintf( " OK !" );
1468                setport( USERBITS, 0, 8 );
1469                putchar( 7 );
1470            }
1471            else
1472            {
1473                textattr( ( RED << 4 ) | WHITE );
1474                cprintf( " VERIFY ERROR ! at %04lX", addr );
1475                textattr( ( CYAN << 4 ) | WHITE );
1476                errbeep();
1477            }
1478        }
1479    }
1480
1481    void change_area( void )
1482    {
1483        if( mfr[mfrno].dev[devno].DataSize == 0 )
1484            area = 0;
1485        else if( area >= 3 )
```

```
1486            area = 1;       /* CODE only */
1487        else if( area == 1 )
1488            area = 2;       /* DATA only */
1489        else if( area == 2 )
1490            area = 3;       /* CODE+DATA */
1491    }
1492
1493    void edit_config( void )
1494    {
1495        static char *osctype[4] = { "LP","XT","HS","RC" };
1496        int i, done;
1497        unsigned val, cfg = *(unsigned*)( buffer + 0x400E );
1498
1499        textattr( ( CYAN << 4 ) | WHITE );
1500        _window(  1,0, 10,39 );
1501        _window( 11,1, 24,78 );
1502
1503        locate(  2, 1 ); cprintf( "Current Oscillator Selection :" );
1504        locate(  3, 1 ); cprintf( "      Watchdog :" );
1505        locate(  4, 1 ); cprintf( "Power-up timer :" );
1506        locate(  5, 1 ); cprintf( "CODE Protection:" );
1507        locate(  6, 1 ); cprintf( "DATA Protection:" );
1508        locate(  7, 1 ); cprintf( "User FLASH pgm :" );
1509        locate(  8, 1 ); cprintf( "Brownout reset :" );
1510        locate(  9, 1 ); cprintf( "Low voltage pgm:" );
1511
1512        locate( 12, 4 ); cprintf( "A : Oscillator option toggle" );
1513        locate( 12,43 ); cprintf( "B : Watchdog Option toggle" );
1514        locate( 13, 4 ); cprintf( "C : Power-up timer Option toggle" );
1515        locate( 13,43 ); cprintf( "D : Brown-out reset Option toggle" );
1516        locate( 14, 4 ); cprintf( "E : Low voltage pgm Option toggle" );
1517
1518        locate( 16, 4 ); cprintf( "K : CODE protection toggle" );
1519        locate( 16,43 ); cprintf( "L : DATA protection toggle" );
1520        locate( 17, 4 ); cprintf( "M : FLASH User write Option toggle" );
1521        locate( 17,43 ); cprintf( "N : DEBUG enable toggle" );
1522
1523        locate( 19, 4 ); cprintf( "1 : edit ID code" );
1524        locate( 19,43 ); cprintf( "2 : set serial on of off" );
1525        locate( 20, 4 ); cprintf( "3 : read ID code" );
1526        locate( 20,43 ); cprintf( "4 : program ID code" );
1527        locate( 21, 4 ); cprintf( "5 : read configuration code" );
1528        locate( 21,43 ); cprintf( "6 : program configuration code" );
1529
1530        locate( 23, 4 ); cprintf( "Select options or <CR><ESC> to go back to the main
            menu ?" );
1531
1532        textattr( ( BLUE << 4 ) | WHITE );
1533        locate(  1, 5 ); cprintf( " Configuration Bit Setting :" );
1534        locate( 11,28 ); cprintf( " Configuration Options :" );
1535
1536        for(;;)
1537        {
1538            textattr( ( BLUE << 4 ) | WHITE );
1539
1540            locate( 9,63 ); cprintf( "%04X", cfg );
1541
1542            locate( 2,32 ); /* OSC */ cprintf( osctype[ cfg & 0x003 ] );
1543            locate( 3,18 ); /* WDT */ cprintf( (cfg & 0x004) ? "Enabled " : "Disabled" );
1544            locate( 4,18 ); /* PWT */ cprintf( (cfg & 0x008) ? "Disabled" : "Enabled " );
1545            locate( 5,18 ); /* CPT */
1546            switch( ( cfg & 0x030 ) >> 4 )
1547            {
1548            case 0: cprintf( "All protected" ); break;
1549            case 1: cprintf( " %04lX - %04lX ", bufsize/2, bufsize-1 ); break;
1550            case 2: cprintf( " %04X - %04lX ", 0, bufsize/2 - 1 ); break;
1551            case 3: cprintf( "Not protected" ); break;
1552            }
1553            locate( 6,18 ); cprintf( (cfg & 0x100) ? "Disabled" : "Enabled " );
1554            locate( 7,18 ); cprintf( (cfg & 0x200) ? "Enabled " : "Disabled" );
1555            locate( 8,18 ); cprintf( (cfg & 0x040) ? "Enabled " : "Disabled" );
1556            locate( 9,18 ); cprintf( (cfg & 0x080) ? "Enabled " : "Disabled" );
```

```
1557
1558            for( done = 0; !done; )
1559            {
1560                done = 1;
1561
1562                switch( toupper( getch() ) )
1563                {
1564                case 0: getch(); done = 0; break;
1565
1566                case '\n':
1567                case '\r':
1568                case 0x1B:
1569                    *(unsigned*)( buffer + 0x400E ) = cfg;
1570                    return;
1571
1572                case 'A' : // Oscillator option toggle
1573                    cfg = ( cfg & ~0x003 ) | ( (cfg + 1) & 0x003 ); break;
1574
1575                case 'B' : // Watchdog Option toggle
1576                    cfg ^= 0x004; break;
1577
1578                case 'C' : // Power-up timer Option toggle
1579                    cfg ^= 0x008; break;
1580
1581                case 'D' : // Brown-out reset Option toggle
1582                    cfg ^= 0x040; break;
1583
1584                case 'E' : // Low voltage pgm Option toggle
1585                    cfg ^= 0x080; break;
1586
1587                case 'K' : // CODE protection toggle
1588                    cfg = ( cfg & ~0x3030 ) | ( ( (cfg&0x3030) + 0x1010) & 0x3030 );
                    break;
1589
1590                case 'L' : // DATA protection toggle
1591                    cfg ^= 0x100; break;
1592
1593                case 'M' : // FLASH User write Option toggle
1594                    cfg ^= 0x200; break;
1595
1596                case 'N' : // DEBUG Option toggle
1597                    cfg ^= 0x400; break;
1598
1599                case '1' : // edit ID code
1600                case '2' : // set serial on of off
1601
1602                    done = 0; break;
1603
1604                case '3' : // read ID code
1605                    power( _UB, _VPP );
1606                    do_load( CFG_SEG, 0x3FFF );     // goto 0x2000 (0x4000)
1607                    for( i = 0x4000; i < 0x4008; i += 2 )
1608                    {
1609                        val = pic_read( 0 );
1610                        buffer[ i ] = val & 0xFF;
1611                        buffer[i+1] = ( val >> 8 ) & 0xFF;
1612                        do_increment( 1 );
1613                    }
1614                    power( 0, 0 ); break;
1615
1616                case '4' : // program ID code
1617                    power( _UB, _VPP );
1618                    do_load( CFG_SEG, 0x3FFF );     // goto 0x2000 (0x4000)
1619                    for( i = 0x4000; i < 0x4008; i += 2 )
1620                    {
1621                        int n = mfr[mfrno].dev[devno].NumProgBytes;
1622                        while( n-- )
1623                        {
1624                            val = buffer[ i ];
1625                            val |= ( buffer[i+1] << 8 );
1626                            do_load( CODE_SEG, val );
1627                            if( n ) do_increment( 1 );
```

```
                    }
                    do_prog1( val );
                    do_increment( 1 );
                }
                power( 0, 0 ); break;

            case '5' : // read configuration code
                power( _UB, _VPP );
                do_load( CFG_SEG, 0x3FFF );     // goto 0x2000 (0x4000)
                do_increment( 7 );
                cfg = pic_read( 0 );
                power( 0, 0 ); break;

            case '6' : // program configuration code
                power( _UB, _VPP );
                do_load( CFG_SEG, cfg );        // goto 0x2000 (0x4000)
                do_increment( 7 );
                do_prog1( cfg );
                power( 0, 0 ); break;

            default: done = 0;
            }
        }
    }
}


void type_select( void )
{
    int done, i, num, len=15, left = 40;
    char no[10];

    if( ( num = mfr[mfrno].numdevs ) > 14 )
    {
        left = 0;
        for( i = len = 0; i < num; ++i )
            if( ( done = strlen( mfr[mfrno].dev[i].name ) ) > len )
                len = done;
    }

    textattr( ( CYAN << 4 ) | WHITE );
    _window( 12,left, 23,79 );

    textattr( ( BLUE << 4 ) | WHITE );
    locate( 12, left+5 ); cprintf( "  TYPE SELECT:" );

    textattr( ( CYAN << 4 ) | WHITE );

    for( i = 0; i < num; ++i )
    {
        locate( 13+(i%7), left + 1 + (i/7)*(len+4) );
        cprintf( "%d.%s", i, mfr[mfrno].dev[i].name );
    }

    locate( 21, left+1 ); cprintf( "<CR> back to main menu." );
    locate( 22, left+1 ); cprintf( "SELECT NUMBER ?" );

    for(;;)
    {
        for( no[0] = done = 0; !done; )
        {
            locate( 22, left+16 ); cprintf( "%s ", no );
            locate( 22, left+16+strlen(no) );

            switch( i = getch() )
            {
            case 0:
                getch();
                break;

            case 8:
                if( ( i = strlen(no) ) != 0 )
```

```
                    no[i-1] = 0;
                break;

            case '\n':
            case '\r':
                if( strlen( no ) &&
                    ( i = atoi(no) ) >= 0 && i < mfr[mfrno].numdevs )
                {
                    devno = i;
                    ShowType();
                    return;
                }
                break;

            case 0x1B:
                return;

            default:
                if( isdigit( i ) )
                    strcat( no, (char*)&i );
                break;
            }
        }
    }
}

void mfr_select( void )
{
    int done, i;
    char no[10];

    textattr( ( CYAN << 4 ) | WHITE );
    _window( 12,40, 23,79 );

    textattr( ( BLUE << 4 ) | WHITE );
    locate( 12, 45 ); cprintf( "  MFR SELECT:" );

    textattr( ( CYAN << 4 ) | WHITE );

    for( i = 0; i < sizeof(mfr) / sizeof(mfr[0]); ++i )
    {
        locate( 13+i, 41 ); cprintf( "%d.%s", i, mfr[i].name );
    }

    locate( 21, 41 ); cprintf( "<CR> back to main menu." );
    locate( 22, 41 ); cprintf( "SELECT NUMBER ?" );

    for(;;)
    {
        for( no[0] = done = 0; !done; )
        {
            locate( 22, 56 ); cprintf( "%s ", no );
            locate( 22, 56+strlen(no) );

            switch( i = getch() )
            {
            case 0:
                getch();
                break;

            case 8:
                if( ( i = strlen(no) ) != 0 )
                    no[i-1] = 0;
                break;

            case '\n':
            case '\r':
                if( strlen( no ) &&
                    ( i = atoi(no) ) >= 0 && i < sizeof(mfr) / sizeof(mfr[0]) )
                {
                    mfrno = i;
                    if( devno >= mfr[mfrno].numdevs )
```

```
1772                        devno = 0;
1773                    ShowType();
1774                    return;
1775                }
1776                break;
1777
1778            case 0x1B:
1779                return;
1780
1781            default:
1782                if( isdigit( i ) )
1783                    strcat( no, (char*)&i );
1784                break;
1785            }
1786        }
1787    }
1788  }
1789
1790  /*=========================================================*/
1791  int main( void )
1792  {
1793      int ch = 0, redraw, first = 1;
1794      long tmpval;
1795
1796      /*---------------------------------------------------------*/
1797      /* main program starts here                                */
1798      /*---------------------------------------------------------*/
1799
1800      getcwd( oldpath, 260 );
1801      strcpy( path, oldpath );
1802
1803      if( ( buffer = farmalloc( BUFSIZE ) ) == NULL )
1804          return -1;
1805
1806      memset( (void far *)buffer, 0, BUFSIZE );
1807
1808      ReadConfig();
1809      delay(0);
1810
1811      for( ;; )
1812      {
1813          if( first )
1814          {
1815              first = 0;
1816
1817              area = 3;
1818
1819              init_hw();
1820              initdacs();
1821              setport( USERBITS, 0, 0 );
1822
1823              DeviceSetup();
1824          }
1825
1826          textattr( ( LIGHTGRAY << 4 ) | YELLOW ); clscrn( 0,0, 24,79 );
1827
1828          locate( 0,0 ); cprintf( "Universal   Programmer" );
1829          locate( 1,0 ); cprintf( "MODEL: PC Based" );
1830          locate( 2,0 ); cprintf( "MPU PIC16 section  " _VERSION_ );
1831
1832          textattr( ( BLUE << 4 ) + WHITE ); clscrn( 0,40, 6,79 );
1833
1834          ShowType();
1835
1836          textattr( ( BLUE << 4 ) + WHITE ); _window( 1,40, 6,79 );
1837          locate( 1,53 ); cprintf( " TARGET ZONE " );
1838          locate( 2,41 ); cprintf( "Buffer start addr.: %04lX", BufStart );
1839          locate( 3,41 ); cprintf( "       end   addr.: %04lX", BufEnd );
1840          locate( 4,41 ); cprintf( "       Check Sum  : %04X", Chks );
1841          locate( 5,41 ); cprintf( "Device start addr.: %04lX", DevStart );
1842
1843          _window( 3,69, 6,79 );
```

```
1844            locate( 4,71 ); cprintf( "COUNTER" );
1845            ShowCounter( 0 );
1846
1847            _window( 7,40, 10,79 );
1848            locate( 7,43 ); cprintf( " Device ID & Configuration bits " );
1849            locate( 8,42 ); cprintf( "ID3 - ID0 :" );
1850            locate( 8,42 ); cprintf( "Lock Bits :" );
1851            locate( 8,67 ); cprintf( "*Fuses :" );
1852            locate( 9,42 ); cprintf( "Configuration bits : " );
1853            ShowConfig();
1854
1855            textattr( ( CYAN << 4 ) | WHITE ); clscrn( 3,0, 23,38 );
1856
1857            locate( 3,0 ); cprintf( "-------------  Main Menu  -------------" );
1858            locate( 4,0 ); cprintf( "1. DOS SHELL                          " );
1859            locate( 5,0 ); cprintf( "2. Load BIN or HEX file to buffer     " );
1860            locate( 6,0 ); cprintf( "3. Save buffer to disk                " );
1861            locate( 7,0 ); cprintf( "4. Edit buffer     7. Display buffer  " );
1862            locate( 8,0 ); cprintf( "5. Change I/O base address           " );
1863            locate( 9,0 ); cprintf( "6. Display loaded file history        " );
1864            locate(10,0 ); cprintf( "W. Swap hi-low bytes in buffer        " );
1865            locate(11,0 ); cprintf( "T. Type select     Z. Target zone     " );
1866            locate(12,0 ); cprintf( "B. Blank check     D. Display         " );
1867            locate(13,0 );
1868            if( mfr[mfrno].dev[devno].erase_type == 0 )
1869                           cprintf( "                                     " );
1870            else
1871                           cprintf( "S. Erase %s   ", memareas[area] );
1872
1873            locate(14,0 ); cprintf( "P. Program %s ", memareas[area] );
1874            locate(15,0 );
1875            if( mfr[mfrno].dev[devno].erase_type == 0 )
1876                           cprintf( "A. Auto(B&P&V&L)   " );
1877            else
1878                           cprintf( "A. Auto(B&S&P&V&L) " );
1879
1880            locate(15,19);
1881            if( mfr[mfrno].dev[devno].DataSize != 0 )
1882                           cprintf(                   "X. Change Mem area  " );
1883            else
1884                           cprintf(                   "                    " );
1885
1886            locate(16,0 ); cprintf( "R. Read            V. Verify          " );
1887            locate(17,0 ); cprintf( "C. Compare and display error          " );
1888            locate(18,0 ); cprintf( "E. Configuration & ID code function   " );
1889            locate(19,0 ); cprintf( "L. Program ID & config. & protect bits " );
1890            locate(20,0 ); cprintf( "Q. Quit                               " );
1891            locate(21,0 ); cprintf( "--------------------------------------" );
1892            locate(22,0 ); cprintf( "Allocation Buffer size : %uK bytes", BUFSIZE/1024 );
1893            if( ( ch = mfr[mfrno].dev[devno].DataSize ) != 0 ) {
1894            locate(23,0 ); cprintf( "Data memory buffer at 4200 ~ %04X", 0x4200 + ch - 1
               );
1895            }
1896
1897            for( redraw = 0; !redraw; )
1898            {
1899                textattr( ( BLUE << 4 ) + WHITE ); clscrn( 24,0, 24,38 );
1900
1901                locate( 24,0 ); cprintf( "Select function ? " );
1902
1903                for(;;)
1904                {
1905                    if( ( ch = getch() ) != 0 )
1906                        break;
1907                    getch();                /* neglect extended code */
1908                }
1909
1910                switch( ch = toupper(ch) )
1911                {
1912                case '1': dos_shell( "" ); redraw = 1; break;
1913                case '2': tmpval = bufsize; bufsize = 0x8000;
1914                          memset( (void far *)buffer, 0, BUFSIZE );
```

```
1915                        load_file();
1916                        bufsize= tmpval;
1917                        redraw = 1; ShowConfig(); break;
1918                case '3': save_file(); break;
1919                case '4': tmpval = bufsize; bufsize = 0x8000;
1920                        edit_buffer();
1921                        bufsize= tmpval;
1922                        redraw = 1; break;
1923                case '5': set_io_adr(); first = redraw = 1; break;
1924                case '7': disp_buffer(); redraw = 1; break;
1925
1926                case 'M': mfr_select(); redraw = first = 1; break;
1927                case 'T': type_select(); redraw = first = 1; break;
1928                case 'E': edit_config(); redraw = 1; break;
1929
1930                case 'X': change_area(); redraw = 1; break;
1931                case 'R': flash_read(); break;
1932                case 'B': flash_check(); break;
1933                case 'S': flash_erase(); break;
1934                case 'P': flash_program(); break;
1935                case 'V': flash_verify(); break;
1936                case 'L': flash_config(); break;
1937                case 'A': flash_auto(); break;
1938
1939    //          case '\n':
1940    //          case '\r': redraw = 1; break;       // refresh
1941                }
1942
1943                setport( USERBITS, 0, 0 );
1944
1945                if( ch == 'Q' )
1946                {
1947                    WriteConfig();
1948                    textattr( LIGHTGRAY ); clrscr();
1949                    chdir( oldpath );
1950                    if( buffer ) farfree( (void far *)buffer );
1951                    return( 0 );
1952                }
1953
1954                textattr( ( LIGHTGRAY << 4 ) | YELLOW ); clscrn( 11,40, 23,79 );
1955            }
1956        }
1957    }
1958
```