```
//*****************LCDG.c************************************
// implementation of the driver for the AGM1264F MODULE
// Jonathan W. Valvano 11/20/09

//  This example accompanies the books
//   "Embedded Microcomputer Systems: Real Time Interfacing",
//         Engineering, copyright (c) 2006,
//   "Introduction to Embedded Microcomputer Systems:
//    Motorola 6811 and 6812 Simulation", Thomson, copyright (c) 2002

// Copyright 2009 by Jonathan W. Valvano, valvano@mail.utexas.edu
//    You may use, edit, run or distribute this file
//    as long as the above copyright notice remains

// Hardware:
// gnd    =  1- AGM1264F ground
// +5V    =  2- AGM1264F Vcc (with 0.1uF cap to ground)
// pot    =  3- AGM1264F Vo  (center pin of 10k pot)
// PP2    =  4- AGM1264F D/I (0 for command, 1 for data)
// gnd    =  5- AGM1264F R/W (blind cycle synchronization)
// PP3    =  6- AGM1264F E   (1 to latch in data/command)
// PH0    =  7- AGM1264F DB0
// PH1    =  8- AGM1264F DB1
// PH2    =  9- AGM1264F DB2
// PH3    = 10- AGM1264F DB3
// PH4    = 11- AGM1264F DB4
// PH5    = 12- AGM1264F DB5
// PH6    = 13- AGM1264F DB6
// PH7    = 14- AGM1264F DB7
// PP0    = 15- AGM1264F CS1 (controls left half of LCD)
// PP1    = 16- AGM1264F CS2 (controls right half of LCD)
// +5V    = 17- AGM1264F RES (reset)
// pot    = 18- ADM1264F Vee (-10V)
// 10k pot from pin 18 to ground, with center to pin 3
// references   http://www.azdisplays.com/prod/g1264f.php
// sample code  http://www.azdisplays.com/PDF/agm1264f_code.pdf
// data sheet   http://www.azdisplays.com/PDF/agm1264f.pdf

// BUG NOTICE 11/11/09 -Valvano
// When changing from right to left or from left to right
//   the first write with data=0 goes to two places
// One can reduce the effect of this bug by
// 1) Changing sides less often
// 2) Ignore autoincrement, and set column and page address each time
// 3) Blanking the screen then write 1's to the screen
// GoTo bug fixed on 11/20/09

//*********************************************************************
#include <mc9s12dp512.h>     /* derivative information */
#include "LCDG.h"
#include "Timer.h"
#include "game.h"

#define E PTP_PTP3
#define DI PTP_PTP2
#define CS2 PTP_PTP1
#define CS1 PTP_PTP0
#define DATA PTH

// assuming TCNT is 1.5 MHz
#define T1usec 2
#define T4usec 6

static unsigned short OpenFlag=0;// 5 wide by 7 tall font

unsigned char Column1; // column position
unsigned char bLeft1;  // to be placed into CS1, in LCD_OutChar
unsigned char bRight1; // to be placed into CS2, in LCD_OutChar
unsigned char Page;
unsigned char bDown;    // true if want font shifted down

const unsigned char Font[96*5]={   // no numbers with bit7=1
  0,0,0,0,0,           // 32  space
  0,0,95,0,0,          // 33  !
  0,7,0,7,0,           // 34  "
  20,127,20,127,20,    // 35  #
  36,42,127,42,18,     // 36  $
  35,19,8,100,98,      // 37  %
```

```
  54,73,85,34,80,      // 38  &
  0,5,3,0,0,           // 39  quote
  0,28,34,65,0,        // 40  (
  0,65,34,28,0,        // 41  )
  20,8,62,8,20,        // 42  *
  8,8,62,8,8,          // 43  plus
  0,80,48,0,0,         // 44  ,
  8,8,8,8,8,           // 45  minus
  0,112,112,112,0,     // 46  .
  32,16,8,4,2,         // 47  /
  62,81,73,69,62,      // 48  0
  0,66,127,64,0,       // 49  1
  66,97,81,73,70,      // 50  2
  33,65,69,75,49,      // 51  3
  24,20,18,127,16,     // 52  4
  39,69,69,69,57,      // 53  5
  60,74,73,73,48,      // 54  6
  3,1,113,9,7,         // 55  7
  54,73,73,73,54,      // 56  8
  6,73,73,41,30,       // 57  9
  0,54,54,0,0,         // 58  :
  0,86,54,0,0,         // 59  ;
  8,20,34,65,0,        // 60  <
  20,20,20,20,20,      // 61  equals
  65,34,20,8,0,        // 62  >
  2,1,81,9,6,          // 63  ?
  50,73,121,65,62,     // 64  @
  126,17,17,17,126,    // 65  A
  127,73,73,73,54,     // 66  B
  62,65,65,65,34,      // 67  C
  127,65,65,65,62,     // 68  D
  127,73,73,73,65,     // 69  E
  127,9,9,9,1,         // 70  F
  62,65,73,73,122,     // 71  G
  127,8,8,8,127,       // 72  H
  65,65,127,65,65,     // 73  I
  32,64,65,63,1,       // 74  J
  127,8,20,34,65,      // 75  K
  127,64,64,64,64,     // 76  L
  127,2,12,2,127,      // 77  M
  127,6,24,96,127,     // 78  N
  62,65,65,65,62,      // 79  O
  127,9,9,9,6,         // 80  P
  62,65,81,33,94,      // 81  Q
  127,9,25,41,70,      // 82  R
  70,73,73,73,49,      // 83  S
  1,1,127,1,1,         // 84  T
  63,64,64,64,63,      // 85  U
  31,32,64,32,31,      // 86  V
  63,64,56,64,63,      // 87  W
  99,20,8,20,99,       // 88  X
  7,8,112,8,7,         // 89  Y
  97,81,73,69,67,      // 90  Z
  0,127,65,65,0,       // 91  [
  2,4,8,16,32,         // 92  back slash
  0,65,65,127,0,       // 93  ]
  4,2,1,2,4,           // 94  ^
  64,64,64,64,64,      // 95  _
  0,1,2,4,0,           // 96  quote
  32,84,84,84,120,     // 97  a
  127,72,68,68,56,     // 98  b
  56,68,68,68,32,      // 99  c
  56,68,68,72,127,     // 100  d
  56,84,84,84,24,      // 101  e
  8,126,9,1,2,         // 102  f
  8,84,84,84,60,       // 103  g
  127,8,4,4,120,       // 104  h
  0,72,125,64,0,       // 105  i
  32,64,68,61,0,       // 106  j
  127,16,40,68,0,      // 107  k
  0,65,127,64,0,       // 108  l
  124,4,24,4,120,      // 109  m
  124,8,4,4,120,       // 110  n
  56,68,68,68,56,      // 111  o
  124,20,20,20,8,      // 112  p
  12,18,18,20,126,     // 113  q
  124,8,4,4,8,         // 114  r
  72,84,84,84,36,      // 115  s
```

```
  4,63,68,64,32,       // 116  t
  60,64,64,32,124,     // 117  u
  28,32,64,32,28,      // 118  v
  60,64,48,64,60,      // 119  w
  68,40,16,40,68,      // 120  x
  12,80,80,80,60,      // 121  y
  68,100,84,76,68,     // 122  z
  0,65,54,8,0,         // 123  }
  0,0,127,0,0,         // 124  |
  0,8,54,65,0,         // 125  {
  8,4,8,16,8,          // 126  ~
  31,36,124,36,31      // 127  UT sign
};


// ********* lcdCmd***********
// Output command to AGM1264F 128-bit by 64-bit graphics display
// Inputs: 8-bit instruction
// Outputs: none
void lcdCmd(unsigned char instruction){
  // R/W=0, write mode  default, R/W=0 always
  // normally D/I will be left at D/I=1 for data
  DI = 0;        // D/I=0, COMMAND WRITE
  Timer_Wait(T1usec);
  E = 1;         // E pulse width > 450ns
  DATA = instruction;
  Timer_Wait(T1usec);
  E = 0;         // falling edge latch, setup time 200ns
  DI = 1;        // D/I=1 default state is data
  Timer_Wait(T4usec);
}

// ********* lcdData***********
// Output data to AGM1264F 128-bit by 64-bit graphics display
// Inputs: 8-bit data
// Outputs: none
void lcdData(unsigned char data){
  // R/W=0, write mode  default, R/W=0 always
  // normally D/I will be left at D/I=1 for data
  E = 1;         // E pulse width > 450ns
  DATA = data;
  Timer_Wait(T1usec);
  E = 0;         // falling edge latch, setup time 200ns
  Timer_Wait(T4usec);
}

// ********* LCD_Init***********
// Initialize AGM1264F 128-bit by 64-bit graphics display
// activates TCNT at 1.5 MHz, assumes PLL active
// Input: none
// Output: none
// does not clear the display
void LCD_Init(void){
  Timer_Init();   // TCNT at 1.5 MHz
  DDRH = 0xFF;    // PH7-PH0 outputs to DB7-DB0, PT3=E
  DDRP |= 0x0F;   // PP3-PP0 outputs to E,DI,CS1,CS2
  CS2 = 1;         // talk to both LCD controllers
  CS1 = 1;
  DI = 1;          // default mode is data
  E = 0;           // inactive
  Timer_Wait1ms(100); // let it warm up
  lcdCmd(0x3F);    // display=ON
  lcdCmd(0xB8);    // Page address (0 to 7) is 0
  lcdCmd(0x40);    // Column address (0 to 63) is 0
  lcdCmd(0xC0);    // Y=0 is at top
  OpenFlag = 1;    // device openopen
  Column1 = 0x41;  // column position
  bLeft1 = 1;
  bRight1 = 0;
  Page = 0xB8;
  bDown = 0;    // true if want font shifted down

}


// ********* LCD_Clear***********
// Clear the entire 1024 byte (8192 bit) image on the
//    AGM1264F 128-bit by 64-bit graphics display
```

```
// Input: value to write into all bytes of display RAM
// Output: none
// e.g., LCD_Clear(0);  // makes all pixels off
void LCD_Clear(unsigned char data){
unsigned char page;
  int i;
  if(OpenFlag == 0) return;
  for(page = 0xB8; page< 0xB8+8; page++){ // pages 0 to 7
    CS2 = 1;          // right enable
    CS1 = 0;
    lcdCmd(page);    // Page address (0 to 7)
    lcdCmd(0x40);    // Column = 0
    for(i=64; i>0; i--){
      lcdData(data);   // copy one byte to right side
    }
    CS2 = 0;
    CS1 = 1;         // left enable
    lcdCmd(page);    // Page address (0 to 7)
    lcdCmd(0x40);    // Column = 0
    for(i=64; i>0; i--){
      lcdData(data);   // copy one byte to left side
    }
  }
}

// page   is 0xB8 to 0xBF for pages 0 to 7
// column is 0x40 to 0x7F for columns 0 to 63
void OutByte(unsigned char page, unsigned char column,unsigned char data){
  lcdCmd(page);    // Page address (0 to 7)
  lcdCmd(column);  // Column = 0 to 63
  lcdData(data);   // data
}

int pixelOn(int type, int x, int y) {
  switch(type) {
    case SHIPEND_UP:
      if((x == 2 && y == 3) ||
         (x == 3 && y == 2) ||
         (x == 3 && y == 3) ||
         (x == 3 && y == 4) ||
         (x == 4 && y == 2) ||
         (x == 4 && y == 3) ||
         (x == 4 && y == 4) ||
         (x == 5 && y == 2) ||
         (x == 5 && y == 3) ||
         (x == 5 && y == 4)) {

        return 1;
      }
      break;
    case SHIPEND_DOWN:
      if((x == 1 && y == 2) ||
         (x == 1 && y == 3) ||
         (x == 1 && y == 4) ||
         (x == 2 && y == 2) ||
         (x == 2 && y == 3) ||
         (x == 2 && y == 4) ||
         (x == 3 && y == 2) ||
         (x == 3 && y == 3) ||
         (x == 3 && y == 4) ||
         (x == 4 && y == 3)) {

        return 1;
      }
      break;
    case SHIPEND_LEFT:
      if((x == 2 && y == 3) ||
         (x == 2 && y == 4) ||
         (x == 2 && y == 5) ||
         (x == 3 && y == 2) ||
         (x == 3 && y == 3) ||
         (x == 3 && y == 4) ||
         (x == 3 && y == 5) ||
         (x == 4 && y == 3) ||
         (x == 4 && y == 4) ||
         (x == 4 && y == 5)) {

        return 1;
```

```
    }
    break;
  case SHIPEND_RIGHT:
    if((x == 2 && y == 1) ||
       (x == 2 && y == 2) ||
       (x == 2 && y == 3) ||
       (x == 3 && y == 1) ||
       (x == 3 && y == 2) ||
       (x == 3 && y == 3) ||
       (x == 3 && y == 4) ||
       (x == 4 && y == 1) ||
       (x == 4 && y == 2) ||
       (x == 4 && y == 3)) {

      return 1;
    }
    break;
  case SHIP_VERT:
    if((x == 1 && y == 2) ||
       (x == 1 && y == 3) ||
       (x == 1 && y == 4) ||
       (x == 2 && y == 2) ||
       (x == 2 && y == 3) ||
       (x == 2 && y == 4) ||
       (x == 3 && y == 2) ||
       (x == 3 && y == 3) ||
       (x == 3 && y == 4) ||
       (x == 4 && y == 2) ||
       (x == 4 && y == 3) ||
       (x == 4 && y == 4) ||
       (x == 5 && y == 2) ||
       (x == 5 && y == 3) ||
       (x == 5 && y == 4)) {

      return 1;
    }
    break;
  case SHIP_HORIZ:
    if((x == 2 && y == 1) ||
       (x == 2 && y == 2) ||
       (x == 2 && y == 3) ||
       (x == 2 && y == 4) ||
       (x == 2 && y == 5) ||
       (x == 3 && y == 1) ||
       (x == 3 && y == 2) ||
       (x == 3 && y == 3) ||
       (x == 3 && y == 4) ||
       (x == 3 && y == 5) ||
       (x == 4 && y == 1) ||
       (x == 4 && y == 2) ||
       (x == 4 && y == 3) ||
       (x == 4 && y == 4) ||
       (x == 4 && y == 5)) {

      return 1;
    }
    break;
  case HIT:
    if((x == 1 && y == 3) ||
       (x == 2 && y == 2) ||
       (x == 2 && y == 3) ||
       (x == 2 && y == 4) ||
       (x == 3 && y == 1) ||
       (x == 3 && y == 2) ||
       (x == 3 && y == 4) ||
       (x == 3 && y == 5) ||
       (x == 4 && y == 2) ||
       (x == 4 && y == 3) ||
       (x == 4 && y == 4) ||
       (x == 5 && y == 3)) {

      return 1;
    }
    break;
  case MISS:
    if((x == 1 && y == 1) ||
       (x == 1 && y == 5) ||
       (x == 2 && y == 2) ||
```

```
          (x == 2 && y == 4) ||
          (x == 3 && y == 3) ||
          (x == 4 && y == 2) ||
          (x == 4 && y == 4) ||
          (x == 5 && y == 1) ||
          (x == 5 && y == 5)) {

        return 1;
      }
      break;
  }
  return 0;
}

void LCD_DrawGrid(unsigned char field[10][10]) {
  int i, j, k;

  //PTP |= 0x80;

  CS1 = 0;
  CS2 = 1;

  for(i=0; i<8; i++) {
    for(j=0; j<61; j++) {
      unsigned char pixels = 0;
      if(!(j%6)) {
        if(i<7) {
          pixels = 0xFF;
        }
        else {
          pixels = 0x1F;
        }
      }
      else {
        switch(i) {
          case 0:
          case 3:
          case 6:
            pixels = 0x41;
            break;
          case 1:
          case 4:
          case 7:
            pixels = 0x10;
            break;
          case 2:
          case 5:
            pixels = 0x04;
            break;
        }
        for(k=0; k<8 && (i<7 || k<4); k++) {
          unsigned char boxRow = ((i*8)+k)/6;
          unsigned char boxCol = j/6;

          unsigned char boxX = ((i*8)+k)%6;
          unsigned char boxY = j%6;

          if(pixelOn(field[boxRow][boxCol], boxX, boxY)) {
            pixels |= 1 << k;
          }
        }
      }
      OutByte(i + 0xB8, j + 0x40, pixels);
    }
  }
}

// ********* LCD_OutChar***********
// Output ASCII character on the
//    AGM1264F 128-bit by 64-bit graphics display
// Input: 7-bit ASCII to display
// Output: none
// letter must be between 32 and 127 inclusive
// execute LCD_GoTo to specify cursor location
void LCD_OutChar(unsigned char letter){
unsigned short i,cnt;
  if(OpenFlag == 0) return;
// page 0 is 0xB8, varies from 0xB7 to 0xBF
```

```
  if(letter<32) return;
  if(letter>127) return;
  i = 5*(letter-32); // index into font table
  CS2 = bRight1;   // right enable
  CS1 = bLeft1;    // left enable
  lcdCmd(Page);    // Page address 0 to 7
  lcdCmd(Column1); // Column = 0
  for(cnt=5; cnt>0; cnt--){
    if(bDown){
      lcdData(Font[i]<<1);  // copy one byte, shifted down
    } else{
      lcdData(Font[i]);  // copy one byte
    }
    i++;
    Column1++;
    if(bLeft1&&(Column1==0x80)){
      Column1 = 0x40;
      bLeft1 = 0;
      bRight1 = 1;     // switch to right side
      CS2 = bRight1;   // right enable
      CS1 = bLeft1;    // left enable
      lcdCmd(Page);    // Page address 0 to 7)
      lcdCmd(Column1); // Column = 0
    }
    if(bRight1&&(Column1==0x7F)){
      Column1 = 0x41;
      bLeft1 = 1;
      bRight1 = 0;     // switch to left side
      CS2 = bRight1;   // right enable
      CS1 = bLeft1;    // left enable
      lcdCmd(Page);    // Page address 0 to 7)
      lcdCmd(Column1); // Column = 0
    }
  }
  lcdData(0);  // inter-character space copy one byte
  Column1++;
  if(bLeft1&&(Column1==0x80)){
    Column1 = 0x40;
    bLeft1 = 0;
    bRight1 = 1;     // switch to right side
    CS2 = bRight1;   // right enable
    CS1 = bLeft1;    // left enable
    lcdCmd(Page);    // Page address 0 to 7)
    lcdCmd(Column1); // Column = 0
  }
  if(bRight1&&(Column1==0x7F)){
    Column1 = 0x41;
    bLeft1 = 1;
    bRight1 = 0;     // switch to left side
    CS2 = bRight1;   // right enable
    CS1 = bLeft1;    // left enable
    lcdCmd(Page);    // Page address 0 to 7)
    lcdCmd(Column1); // Column = 0
  }
}



//---------------------LCD_OutString--------------
// Display String
// Input: pointer to NULL-terminationed ASCII string
// Output: none
void LCD_OutString(char *pt){
  if(OpenFlag==0){
    return;  // not open
  }
  while(*pt){
    LCD_OutChar((unsigned char)*pt);
    pt++;
  }
}

//------------------------LCD_GoTo-------------------------
// Move cursor
// Input: line number is 1 to 8, column from 1 to 21
// Output: none
// errors: it will ignore legal addresses
void LCD_GoTo(int line, int column){
```

```
  if(OpenFlag==0){
    return;  // not open
  }
  if((line<1) || (line>8)) return;
  if((column<1) || (column>21)) return;
  if(line<5){
    bDown = 0;              // normal position on lines 1,2,3,4
  } else{
    bDown = 0xFF;           // shifted down on lines 5,6,7,8
  }
  Page = 0xB8+line-1;       // 0xB8 to 0xBF
  if(column<12){
    Column1 = 59+6*column; // 0x41+6*(column-1);
    bLeft1 = 1;
    bRight1 = 0;            // on left side
  } else{
    Column1 = 6*column-5;  // 0x43+6*(column-12);
    bLeft1 = 0;
    bRight1 = 1;            // on right side
  }
}
```