```
  1: //*****************LCDG.c*************************************
  2: // implementation of the driver for the AGM1264F MODULE
  3: // Jonathan W. Valvano 11/20/09
  4:
  5: //  This example accompanies the books
  6: //   "Embedded Microcomputer Systems: Real Time Interfacing",
  7: //         Engineering, copyright (c) 2006,
  8: //   "Introduction to Embedded Microcomputer Systems:
  9: //    Motorola 6811 and 6812 Simulation", Thomson, copyright (c) 2002
 10:
 11: // Copyright 2009 by Jonathan W. Valvano, valvano@mail.utexas.edu
 12: //    You may use, edit, run or distribute this file
 13: //    as long as the above copyright notice remains
 14:
 15: // Hardware:
 16: // gnd    =  1- AGM1264F ground
 17: // +5V    =  2- AGM1264F Vcc (with 0.1uF cap to ground)
 18: // pot    =  3- AGM1264F Vo  (center pin of 10k pot)
 19: // PP2    =  4- AGM1264F D/I (0 for command, 1 for data)
 20: // gnd    =  5- AGM1264F R/W (blind cycle synchronization)
 21: // PP3    =  6- AGM1264F E   (1 to latch in data/command)
 22: // PH0    =  7- AGM1264F DB0
 23: // PH1    =  8- AGM1264F DB1
 24: // PH2    =  9- AGM1264F DB2
 25: // PH3    = 10- AGM1264F DB3
 26: // PH4    = 11- AGM1264F DB4
 27: // PH5    = 12- AGM1264F DB5
 28: // PH6    = 13- AGM1264F DB6
 29: // PH7    = 14- AGM1264F DB7
 30: // PP0    = 15- AGM1264F CS1 (controls left half of LCD)
 31: // PP1    = 16- AGM1264F CS2 (controls right half of LCD)
 32: // +5V    = 17- AGM1264F RES (reset)
 33: // pot    = 18- ADM1264F Vee (-10V)
 34: // 10k pot from pin 18 to ground, with center to pin 3
 35: // references   http://www.azdisplays.com/prod/g1264f.php
 36: // sample code  http://www.azdisplays.com/PDF/agm1264f_code.pdf
 37: // data sheet   http://www.azdisplays.com/PDF/agm1264f.pdf
 38:
 39: // BUG NOTICE 11/11/09 -Valvano
 40: // When changing from right to left or from left to right
 41: //   the first write with data=0 goes to two places
 42: // One can reduce the effect of this bug by
 43: // 1) Changing sides less often
 44: // 2) Ignore autoincrement, and set column and page address each time
 45: // 3) Blanking the screen then write 1's to the screen
 46: // GoTo bug fixed on 11/20/09
 47:
 48: //*********************************************************************
 49: #include "defs.h"
 50: #include "LCDG.h"
 51: #include "Timer.h"
 52: #include "game.h"
 53:
 54: // assuming TCNT is 1.5 MHz
 55: #define T1usec 2
 56: #define T4usec 6
 57:
 58: static unsigned short OpenFlag=0;// 5 wide by 7 tall font
 59:
 60: unsigned char Column1; // column position
 61: unsigned char bLeft1;  // to be placed into CS1, in LCD_OutChar
 62: unsigned char bRight1; // to be placed into CS2, in LCD_OutChar
 63: unsigned char Page;
 64: unsigned char bDown;    // true if want font shifted down
 65:
 66: const unsigned char Font[96*5]={   // no numbers with bit7=1
 67:   0,0,0,0,0,           // 32  space
 68:   0,0,95,0,0,          // 33  !
 69:   0,7,0,7,0,           // 34  "
 70:   20,127,20,127,20,    // 35  #
 71:   36,42,127,42,18,     // 36  $
 72:   35,19,8,100,98,      // 37  %
 73:   54,73,85,34,80,      // 38  &
 74:   0,5,3,0,0,           // 39  quote
 75:   0,28,34,65,0,        // 40  (
 76:   0,65,34,28,0,        // 41  )
 77:   20,8,62,8,20,        // 42  *
 78:   8,8,62,8,8,          // 43  plus
```

```
  79:   0,80,48,0,0,          // 44  ,
  80:   8,8,8,8,8,            // 45  minus
  81:   0,112,112,112,0,      // 46  .
  82:   32,16,8,4,2,          // 47  /
  83:   62,81,73,69,62,       // 48  0
  84:   0,66,127,64,0,        // 49  1
  85:   66,97,81,73,70,       // 50  2
  86:   33,65,69,75,49,       // 51  3
  87:   24,20,18,127,16,      // 52  4
  88:   39,69,69,69,57,       // 53  5
  89:   60,74,73,73,48,       // 54  6
  90:   3,1,113,9,7,          // 55  7
  91:   54,73,73,73,54,       // 56  8
  92:   6,73,73,41,30,        // 57  9
  93:   0,54,54,0,0,          // 58  :
  94:   0,86,54,0,0,          // 59  ;
  95:   8,20,34,65,0,         // 60  <
  96:   20,20,20,20,20,       // 61  equals
  97:   65,34,20,8,0,         // 62  >
  98:   2,1,81,9,6,           // 63  ?
  99:   50,73,121,65,62,      // 64  @
 100:   126,17,17,17,126,     // 65  A
 101:   127,73,73,73,54,      // 66  B
 102:   62,65,65,65,34,       // 67  C
 103:   127,65,65,65,62,      // 68  D
 104:   127,73,73,73,65,      // 69  E
 105:   127,9,9,9,1,          // 70  F
 106:   62,65,73,73,122,      // 71  G
 107:   127,8,8,8,127,        // 72  H
 108:   65,65,127,65,65,      // 73  I
 109:   32,64,65,63,1,        // 74  J
 110:   127,8,20,34,65,       // 75  K
 111:   127,64,64,64,64,      // 76  L
 112:   127,2,12,2,127,       // 77  M
 113:   127,6,24,96,127,      // 78  N
 114:   62,65,65,65,62,       // 79  O
 115:   127,9,9,9,6,          // 80  P
 116:   62,65,81,33,94,       // 81  Q
 117:   127,9,25,41,70,       // 82  R
 118:   70,73,73,73,49,       // 83  S
 119:   1,1,127,1,1,          // 84  T
 120:   63,64,64,64,63,       // 85  U
 121:   31,32,64,32,31,       // 86  V
 122:   63,64,56,64,63,       // 87  W
 123:   99,20,8,20,99,        // 88  X
 124:   7,8,112,8,7,          // 89  Y
 125:   97,81,73,69,67,       // 90  Z
 126:   0,127,65,65,0,        // 91  [
 127:   2,4,8,16,32,          // 92  back slash
 128:   0,65,65,127,0,        // 93  ]
 129:   4,2,1,2,4,            // 94  ^
 130:   64,64,64,64,64,       // 95  _
 131:   0,1,2,4,0,            // 96  quote
 132:   32,84,84,84,120,      // 97  a
 133:   127,72,68,68,56,      // 98  b
 134:   56,68,68,68,32,       // 99  c
 135:   56,68,68,72,127,      // 100  d
 136:   56,84,84,84,24,       // 101  e
 137:   8,126,9,1,2,          // 102  f
 138:   8,84,84,84,60,        // 103  g
 139:   127,8,4,4,120,        // 104  h
 140:   0,72,125,64,0,        // 105  i
 141:   32,64,68,61,0,        // 106  j
 142:   127,16,40,68,0,       // 107  k
 143:   0,65,127,64,0,        // 108  l
 144:   124,4,24,4,120,       // 109  m
 145:   124,8,4,4,120,        // 110  n
 146:   56,68,68,68,56,       // 111  o
 147:   124,20,20,20,8,       // 112  p
 148:   12,18,18,20,126,      // 113  q
 149:   124,8,4,4,8,          // 114  r
 150:   72,84,84,84,36,       // 115  s
 151:   4,63,68,64,32,        // 116  t
 152:   60,64,64,32,124,      // 117  u
 153:   28,32,64,32,28,       // 118  v
 154:   60,64,48,64,60,       // 119  w
 155:   68,40,16,40,68,       // 120  x
 156:   12,80,80,80,60,       // 121  y
```

```
157:   68,100,84,76,68,     // 122  z
158:   0,65,54,8,0,         // 123  }
159:   0,0,127,0,0,         // 124  |
160:   0,8,54,65,0,         // 125  {
161:   8,4,8,16,8,          // 126  ~
162:   31,36,124,36,31      // 127  UT sign
163: };
164:
165:
166: // ********* lcdCmd***********
167: // Output command to AGM1264F 128-bit by 64-bit graphics display
168: // Inputs: 8-bit instruction
169: // Outputs: none
170: void lcdCmd(unsigned char instruction){
171:   // R/W=0, write mode  default, R/W=0 always
172:   // normally D/I will be left at D/I=1 for data
173:   DI = 0;        // D/I=0, COMMAND WRITE
174:   Timer_Wait(T1usec);
175:   E = 1;         // E pulse width > 450ns
176:   SET_DATA(instruction);
177:   Timer_Wait(T1usec);
178:   E = 0;         // falling edge latch, setup time 200ns
179:   DI = 1;        // D/I=1 default state is data
180:   Timer_Wait(T4usec);
181: }
182:
183: // ********* lcdData***********
184: // Output data to AGM1264F 128-bit by 64-bit graphics display
185: // Inputs: 8-bit data
186: // Outputs: none
187: void lcdData(unsigned char data){
188:   // R/W=0, write mode  default, R/W=0 always
189:   // normally D/I will be left at D/I=1 for data
190:   E = 1;         // E pulse width > 450ns
191:   SET_DATA(data);
192:   Timer_Wait(T1usec);
193:   E = 0;         // falling edge latch, setup time 200ns
194:   Timer_Wait(T4usec);
195: }
196:
197: // ********* LCD_Init***********
198: // Initialize AGM1264F 128-bit by 64-bit graphics display
199: // activates TCNT at 1.5 MHz, assumes PLL active
200: // Input: none
201: // Output: none
202: // does not clear the display
203: void LCD_Init(void){
204:   Timer_Init();   // TCNT at 1.5 MHz
205:   DATADR = 0xFF;    // PH7-PH0 outputs to DB7-DB0, PT3=E
206:   SET_LCD_DDR1();   // PP3-PP0 outputs to E,DI,CS1,CS2
207:   SET_LCD_DDR2();   // PP3-PP0 outputs to E,DI,CS1,CS2
208:   CS2 = 1;         // talk to both LCD controllers
209:   CS1 = 1;
210:   DI = 1;          // default mode is data
211:   E = 0;           // inactive
212:   Timer_Wait1ms(100); // let it warm up
213:   lcdCmd(0x3F);    // display=ON
214:   lcdCmd(0xB8);    // Page address (0 to 7) is 0
215:   lcdCmd(0x40);    // Column address (0 to 63) is 0
216:   lcdCmd(0xC0);    // Y=0 is at top
217:   OpenFlag = 1;    // device openopen
218:   Column1 = 0x41;  // column position
219:   bLeft1 = 1;
220:   bRight1 = 0;
221:   Page = 0xB8;
222:   bDown = 0;    // true if want font shifted down
223:
224: }
225:
226:
227: // ********* LCD_Clear***********
228: // Clear the entire 1024 byte (8192 bit) image on the
229: //    AGM1264F 128-bit by 64-bit graphics display
230: // Input: value to write into all bytes of display RAM
231: // Output: none
232: // e.g., LCD_Clear(0);  // makes all pixels off
233: void LCD_Clear(unsigned char data){
234: unsigned char page;
```

```
235:   int i;
236:   if(OpenFlag == 0) return;
237:   for(page = 0xB8; page< 0xB8+8; page++){ // pages 0 to 7
238:     CS2 = 1;          // right enable
239:     CS1 = 0;
240:     lcdCmd(page);    // Page address (0 to 7)
241:     lcdCmd(0x40);    // Column = 0
242:     for(i=64; i>0; i--){
243:       lcdData(data);   // copy one byte to right side
244:     }
245:     CS2 = 0;
246:     CS1 = 1;         // left enable
247:     lcdCmd(page);    // Page address (0 to 7)
248:     lcdCmd(0x40);    // Column = 0
249:     for(i=64; i>0; i--){
250:       lcdData(data);   // copy one byte to left side
251:     }
252:   }
253: }
254:
255: // page   is 0xB8 to 0xBF for pages 0 to 7
256: // column is 0x40 to 0x7F for columns 0 to 63
257: void OutByte(unsigned char page, unsigned char column,unsigned char data){
258:   lcdCmd(page);    // Page address (0 to 7)
259:   lcdCmd(column);  // Column = 0 to 63
260:   lcdData(data);   // data
261: }
262:
263: int pixelOn(int type, int x, int y) {
264:   switch(type) {
265:     case SHIPEND_UP:
266:       if((x == 2 && y == 3) ||
267:          (x == 3 && y == 2) ||
268:          (x == 3 && y == 3) ||
269:          (x == 3 && y == 4) ||
270:          (x == 4 && y == 2) ||
271:          (x == 4 && y == 3) ||
272:          (x == 4 && y == 4) ||
273:          (x == 5 && y == 2) ||
274:          (x == 5 && y == 3) ||
275:          (x == 5 && y == 4)) {
276:
277:         return 1;
278:       }
279:       break;
280:     case SHIPEND_DOWN:
281:       if((x == 1 && y == 2) ||
282:          (x == 1 && y == 3) ||
283:          (x == 1 && y == 4) ||
284:          (x == 2 && y == 2) ||
285:          (x == 2 && y == 3) ||
286:          (x == 2 && y == 4) ||
287:          (x == 3 && y == 2) ||
288:          (x == 3 && y == 3) ||
289:          (x == 3 && y == 4) ||
290:          (x == 4 && y == 3)) {
291:
292:         return 1;
293:       }
294:       break;
295:     case SHIPEND_LEFT:
296:       if((x == 2 && y == 3) ||
297:          (x == 2 && y == 4) ||
298:          (x == 2 && y == 5) ||
299:          (x == 3 && y == 2) ||
300:          (x == 3 && y == 3) ||
301:          (x == 3 && y == 4) ||
302:          (x == 3 && y == 5) ||
303:          (x == 4 && y == 3) ||
304:          (x == 4 && y == 4) ||
305:          (x == 4 && y == 5)) {
306:
307:         return 1;
308:       }
309:       break;
310:     case SHIPEND_RIGHT:
311:       if((x == 2 && y == 1) ||
312:          (x == 2 && y == 2) ||
```

```
313:          (x == 2 && y == 3) ||
314:          (x == 3 && y == 1) ||
315:          (x == 3 && y == 2) ||
316:          (x == 3 && y == 3) ||
317:          (x == 3 && y == 4) ||
318:          (x == 4 && y == 1) ||
319:          (x == 4 && y == 2) ||
320:          (x == 4 && y == 3)) {
321:
322:        return 1;
323:      }
324:      break;
325:    case SHIP_VERT:
326:      if((x == 1 && y == 2) ||
327:         (x == 1 && y == 3) ||
328:         (x == 1 && y == 4) ||
329:         (x == 2 && y == 2) ||
330:         (x == 2 && y == 3) ||
331:         (x == 2 && y == 4) ||
332:         (x == 3 && y == 2) ||
333:         (x == 3 && y == 3) ||
334:         (x == 3 && y == 4) ||
335:         (x == 4 && y == 2) ||
336:         (x == 4 && y == 3) ||
337:         (x == 4 && y == 4) ||
338:         (x == 5 && y == 2) ||
339:         (x == 5 && y == 3) ||
340:         (x == 5 && y == 4)) {
341:
342:        return 1;
343:      }
344:      break;
345:    case SHIP_HORIZ:
346:      if((x == 2 && y == 1) ||
347:         (x == 2 && y == 2) ||
348:         (x == 2 && y == 3) ||
349:         (x == 2 && y == 4) ||
350:         (x == 2 && y == 5) ||
351:         (x == 3 && y == 1) ||
352:         (x == 3 && y == 2) ||
353:         (x == 3 && y == 3) ||
354:         (x == 3 && y == 4) ||
355:         (x == 3 && y == 5) ||
356:         (x == 4 && y == 1) ||
357:         (x == 4 && y == 2) ||
358:         (x == 4 && y == 3) ||
359:         (x == 4 && y == 4) ||
360:         (x == 4 && y == 5)) {
361:
362:        return 1;
363:      }
364:      break;
365:    case HIT:
366:      if((x == 1 && y == 3) ||
367:         (x == 2 && y == 2) ||
368:         (x == 2 && y == 3) ||
369:         (x == 2 && y == 4) ||
370:         (x == 3 && y == 1) ||
371:         (x == 3 && y == 2) ||
372:         (x == 3 && y == 4) ||
373:         (x == 3 && y == 5) ||
374:         (x == 4 && y == 2) ||
375:         (x == 4 && y == 3) ||
376:         (x == 4 && y == 4) ||
377:         (x == 5 && y == 3)) {
378:
379:        return 1;
380:      }
381:      break;
382:    case MISS:
383:      if((x == 1 && y == 1) ||
384:         (x == 1 && y == 5) ||
385:         (x == 2 && y == 2) ||
386:         (x == 2 && y == 4) ||
387:         (x == 3 && y == 3) ||
388:         (x == 4 && y == 2) ||
389:         (x == 4 && y == 4) ||
390:         (x == 5 && y == 1) ||
```

```
391:          (x == 5 && y == 5)) {
392:
393:         return 1;
394:       }
395:       break;
396:   }
397:   return 0;
398: }
399:
400: void LCD_DrawGrid(unsigned char field[10][10]) {
401:   int i, j, k;
402:
403:   //PTP |= 0x80;
404:
405:   CS1 = 0;
406:   CS2 = 1;
407:
408:   for(i=0; i<8; i++) {
409:     for(j=0; j<61; j++) {
410:       unsigned char pixels = 0;
411:       if(!(j%6)) {
412:         if(i<7) {
413:           pixels = 0xFF;
414:         }
415:         else {
416:           pixels = 0x1F;
417:         }
418:       }
419:       else {
420:         switch(i) {
421:           case 0:
422:           case 3:
423:           case 6:
424:             pixels = 0x41;
425:             break;
426:           case 1:
427:           case 4:
428:           case 7:
429:             pixels = 0x10;
430:             break;
431:           case 2:
432:           case 5:
433:             pixels = 0x04;
434:             break;
435:         }
436:         for(k=0; k<8 && (i<7 || k<4); k++) {
437:           unsigned char boxRow = ((i*8)+k)/6;
438:           unsigned char boxCol = j/6;
439:
440:           unsigned char boxX = ((i*8)+k)%6;
441:           unsigned char boxY = j%6;
442:
443:           CursorType curs = Game_GetCursor();
444:
445:           if(pixelOn(field[boxRow][boxCol], boxX, boxY)) {
446:             pixels |= 1 << k;
447:           }
448:           if(Game_GetState() == PLAYER_TURN_WAITING) {
449:             if(curs.x == boxRow && curs.y == boxCol) {
450:               if((boxX == 1 && boxY == 1) ||
451:                  (boxX == 1 && boxY == 2) ||
452:                  (boxX == 1 && boxY == 4) ||
453:                  (boxX == 1 && boxY == 5) ||
454:                  (boxX == 2 && boxY == 1) ||
455:                  (boxX == 2 && boxY == 5) ||
456:                  (boxX == 4 && boxY == 1) ||
457:                  (boxX == 4 && boxY == 5) ||
458:                  (boxX == 5 && boxY == 1) ||
459:                  (boxX == 5 && boxY == 2) ||
460:                  (boxX == 5 && boxY == 4) ||
461:                  (boxX == 5 && boxY == 5)) {
462:
463:                 pixels |= 1 << k;
464:               }
465:             }
466:           }
467:         }
468:       }
```

```
469:       OutByte(i + 0xB8, j + 0x40, pixels);
470:     }
471:   }
472: }
473:
474: // ********* LCD_OutChar***********
475: // Output ASCII character on the
476: //    AGM1264F 128-bit by 64-bit graphics display
477: // Input: 7-bit ASCII to display
478: // Output: none
479: // letter must be between 32 and 127 inclusive
480: // execute LCD_GoTo to specify cursor location
481: void LCD_OutChar(unsigned char letter){
482: unsigned short i,cnt;
483:   if(OpenFlag == 0) return;
484: // page 0 is 0xB8, varies from 0xB7 to 0xBF
485:   if(letter<32) return;
486:   if(letter>127) return;
487:   i = 5*(letter-32); // index into font table
488:   CS2 = bRight1;   // right enable
489:   CS1 = bLeft1;    // left enable
490:   lcdCmd(Page);    // Page address 0 to 7
491:   lcdCmd(Column1); // Column = 0
492:   for(cnt=5; cnt>0; cnt--){
493:     if(bDown){
494:       lcdData(Font[i]<<1);  // copy one byte, shifted down
495:     } else{
496:       lcdData(Font[i]);  // copy one byte
497:     }
498:     i++;
499:     Column1++;
500:     if(bLeft1&&(Column1==0x80)){
501:       Column1 = 0x40;
502:       bLeft1 = 0;
503:       bRight1 = 1;     // switch to right side
504:       CS2 = bRight1;   // right enable
505:       CS1 = bLeft1;    // left enable
506:       lcdCmd(Page);    // Page address 0 to 7)
507:       lcdCmd(Column1); // Column = 0
508:     }
509:     if(bRight1&&(Column1==0x7F)){
510:       Column1 = 0x41;
511:       bLeft1 = 1;
512:       bRight1 = 0;     // switch to left side
513:       CS2 = bRight1;   // right enable
514:       CS1 = bLeft1;    // left enable
515:       lcdCmd(Page);    // Page address 0 to 7)
516:       lcdCmd(Column1); // Column = 0
517:     }
518:   }
519:   lcdData(0);  // inter-character space copy one byte
520:   Column1++;
521:   if(bLeft1&&(Column1==0x80)){
522:     Column1 = 0x40;
523:     bLeft1 = 0;
524:     bRight1 = 1;     // switch to right side
525:     CS2 = bRight1;   // right enable
526:     CS1 = bLeft1;    // left enable
527:     lcdCmd(Page);    // Page address 0 to 7)
528:     lcdCmd(Column1); // Column = 0
529:   }
530:   if(bRight1&&(Column1==0x7F)){
531:     Column1 = 0x41;
532:     bLeft1 = 1;
533:     bRight1 = 0;     // switch to left side
534:     CS2 = bRight1;   // right enable
535:     CS1 = bLeft1;    // left enable
536:     lcdCmd(Page);    // Page address 0 to 7)
537:     lcdCmd(Column1); // Column = 0
538:   }
539: }
540:
541:
542:
543: //---------------------LCD_OutString-------------
544: // Display String
545: // Input: pointer to NULL-terminationed ASCII string
546: // Output: none
```

```
547: void LCD_OutString(char *pt){
548:   if(OpenFlag==0){
549:     return;  // not open
550:   }
551:   while(*pt){
552:     LCD_OutChar((unsigned char)*pt);
553:     pt++;
554:   }
555: }
556:
557: //-----------------------LCD_GoTo-----------------------
558: // Move cursor
559: // Input: line number is 1 to 8, column from 1 to 21
560: // Output: none
561: // errors: it will ignore legal addresses
562: void LCD_GoTo(int line, int column){
563:   if(OpenFlag==0){
564:     return;  // not open
565:   }
566:   if((line<1) || (line>8)) return;
567:   if((column<1) || (column>21)) return;
568:   if(line<5){
569:     bDown = 0;             // normal position on lines 1,2,3,4
570:   } else{
571:     bDown = 0xFF;          // shifted down on lines 5,6,7,8
572:   }
573:   Page = 0xB8+line-1;      // 0xB8 to 0xBF
574:   if(column<12){
575:     Column1 = 59+6*column; // 0x41+6*(column-1);
576:     bLeft1 = 1;
577:     bRight1 = 0;           // on left side
578:   } else{
579:     Column1 = 6*column-5;  // 0x43+6*(column-12);
580:     bLeft1 = 0;
581:     bRight1 = 1;           // on right side
582:   }
583: }
```