```
 1: #include "defs.h"
 2:
 3: #include "PLL.h"
 4: #include "LCDG.h"
 5: #include "Timer.h"
 6: #include "Game.h"
 7: #include "switch.h"
 8: #include "music.h"
 9:
10: void main(void) {
11:   PLL_Init();
12:   LCD_Init();   // TCNT at 1.5 MHz
13:   Timer_Init();
14:   DDRP |= 0xA0; // heartbeats, PP7 every 3000, PP5 at sampling rate
15:   Key_Init();
16:   DAC_Init();
17:   Music_InitOC7();
18:   asm cli
19:   Game_Init();
20:
21:   //enableOC6(&whee, 60000, 25, 5);
22:
23:   for(;;) {
24:   }
25: }
26:
```

```
 1: #ifndef DEFS
 2: #define DEFS
 3:
 4: #include <hidef.h>      /* common defines and macros */
 5: #include "derivative.h"      /* derivative-specific definitions */
 6:
 7: unsigned char reverseByte(unsigned char data);
 8:
 9: #define FIRST
10:
11: #define DEBOUNCE_DELAY 15000
12:
13: #define SET_LCD_DDR1() (DDRT |= 0xC0)
14: #define SET_LCD_DDR2() (DDRP |= 0x0C)
15:
16: #define E PTT_PTT6
17: #define DI PTT_PTT7
18: #define CS2 PTP_PTP3
19: #define CS1 PTP_PTP2
20: #define DATADR DDRB
21: #define SET_DATA(x) (PORTB = reverseByte(x))
22:
23: #define SW_PTP0 Game_DPad(LEFT)
24: #define SW_PTP1 Game_DPad(DOWN)
25: #define SW_PTP2 Game_DPad(UP)
26: #define SW_PTP3 Game_DPad(RIGHT)
27: #define SW_PTP4 Game_B()
28: #define SW_PTP5 Game_A()
29:
30: #define LED_DDR0 DDRA_BIT0
31: #define LED_DDR1 DDRA_BIT1
32: #define LED_DDR2 DDRA_BIT2
33: #define LED_DDR3 DDRS_DDRS2
34: #define LED_DDR4 DDRS_DDRS3
35: #define LED_DDR5 DDRP_DDRP6
36:
37: #define LED0 PORTA_BIT0
38: #define LED1 PORTA_BIT1
39: #define LED2 PORTA_BIT2
40: #define LED3 PTS_PTS2
41: #define LED4 PTS_PTS3
42: #define LED5 PTP_PTP6
43:
44: #define SS_DDR DDRM_DDRM3
45: #define MOSI_DDR DDRM_DDRM4
46: #define SCK_DDR DDRM_DDRM5
47:
48: #define SS PTM_PTM3
49: #define MOSI PTM_PTM4
50: #define SCK PTM_PTM5
51:
52: #define RX_DDR DDRS_DDRS0
53: #define TX_DDR DDRS_DDRS1
54:
55: #define SCI_INTERRUPT 20
56:
57: #endif
```

```
 1: unsigned char reverseByte(unsigned char data) {
 2:   int i;
 3:   unsigned char reversed = 0;
 4:
 5:   for(i=0; i<8; i++) {
 6:     if(data&(1<<i)) {
 7:       reversed |= (1<<(7-i));
 8:     }
 9:   }
10:
11:   return reversed;
12: }
```

```
 1: #ifndef GAME_H
 2: #define GAME_H
 3:
 4: #define HIT           0
 5: #define MISS          1
 6: #define SHIPEND_UP    2
 7: #define SHIPEND_DOWN  3
 8: #define SHIPEND_LEFT  4
 9: #define SHIPEND_RIGHT 5
10: #define SHIP_VERT     6
11: #define SHIP_HORIZ    7
12: #define EMPTY         8
13:
14: #define WELCOME               0
15: #define WAITING_FOR_OPPONENT  1
16: #define PLACING_SHIPS         2
17: #define PLAYER_TURN_WAITING   3
18: #define PLAYER_TURN_DONE      4
19: #define COMPUTER_SCREEN       5
20: #define OPPONENT_TURN_WAITING 6
21: #define OPPONENT_TURN_DONE    7
22: #define WIN                   8
23: #define LOSE                  9
24:
25: #define UP    0
26: #define DOWN  1
27: #define LEFT  2
28: #define RIGHT 3
29:
30: typedef struct {
31:   unsigned int x:4;
32:   unsigned int y:4;
33: } CursorType;
34:
35: void Game_Init(void);
36: void Game_Update(void);
37:
38: void Game_DPad(unsigned char direction);
39: void Game_A(void);
40: void Game_B(void);
41:
42: CursorType Game_GetCursor(void);
43: int Game_GetState(void);
44:
45: #endif
```

```
 1: #include "defs.h"
 2: #include <string.h>
 3: #include "game.h"
 4: #include "LCDG.h"
 5: #include "switch.h"
 6: #include "Timer.h"
 7: #include "Music.h"
 8: #include "xbee.h"
 9:
10: #define VERTICAL 0
11: #define HORIZONTAL 1
12:
13: typedef struct {
14:   unsigned char x;
15:   unsigned char y;
16:   unsigned char orientation;
17:   unsigned char size;
18:   unsigned char hits;
19: } ShipType;
20:
21: typedef struct {
22:   unsigned int type:1;
23:   unsigned int x:4;
24:   unsigned int y:4;
25: } AttackType;
26:
27: CursorType cursor;
28:
29: static int state;
30:
31: static int buttonFlag;
32:
33: static char string[10];
34:
35: static ShipType ships[5] = {
36:   {0, 0, VERTICAL, 2, 0},
37:   {0, 0, VERTICAL, 3, 0},
38:   {0, 0, VERTICAL, 3, 0},
39:   {0, 0, VERTICAL, 4, 0},
40:   {0, 0, VERTICAL, 5, 0}
41: };
42:
43: static ShipType computerShips[5] = {
44:   {0, 0, VERTICAL, 2, 0},
45:   {0, 0, VERTICAL, 3, 0},
46:   {0, 0, VERTICAL, 3, 0},
47:   {0, 0, VERTICAL, 4, 0},
48:   {0, 0, VERTICAL, 5, 0}
49: };
50:
51: static unsigned char field[10][10];
52:
53: static int numShips;
54:
55: static AttackType enemyAttacks[100];
56: static int numEnemyAttacks;
57:
58: static AttackType playerAttacks[100];
59: static int numPlayerAttacks;
60:
61: int findValidPos(ShipType * array, int index);
62:
63: int checkDead(ShipType * array) {
64:   int i;
65:   for(i=0; i<5; i++) {
66:     if(array[i].size != array[i].hits) {
67:       return 0;
68:     }
69:   }
70:   return 1;
71: }
72:
73: void incState(void) {
74:   switch(state) {
75:     case WELCOME:
76:       state = WAITING_FOR_OPPONENT;
77:       break;
78:     case WAITING_FOR_OPPONENT:
```

```
  79:       numShips = 1;
  80:       state = PLACING_SHIPS;
  81:       LCD_Clear(0);
  82:       break;
  83:     case PLACING_SHIPS:
  84:       cursor.x = 0;
  85:       cursor.y = 0;
  86:       state = PLAYER_TURN_WAITING;
  87:       LCD_Clear(0);
  88:       strcpy(string, " ATTACK!!!");
  89:       break;
  90:     case PLAYER_TURN_DONE:
  91:       if(checkDead(computerShips)) {
  92:         state = WIN;
  93:       }
  94:       else {
  95:         state = OPPONENT_TURN_WAITING;
  96:         strcpy(string, " Opponent");
  97:       }
  98:       break;
  99:     case COMPUTER_SCREEN:
 100:       state = PLAYER_TURN_WAITING;
 101:       break;
 102:     case OPPONENT_TURN_WAITING:
 103:       state = OPPONENT_TURN_DONE;
 104:       break;
 105:     case OPPONENT_TURN_DONE:
 106:       if(checkDead(ships)) {
 107:         state = LOSE;
 108:       }
 109:       else {
 110:         state = PLAYER_TURN_WAITING;
 111:         strcpy(string, " ATTACK!!!");
 112:       }
 113:       break;
 114:   }
 115:   Game_Update();
 116: }
 117:
 118: unsigned char random(unsigned char max) {
 119:   unsigned static char seed1 = 0;
 120:   unsigned static char seed2;
 121:   unsigned static short last = 0;
 122:
 123:   unsigned short tcnt = TCNT;
 124:   seed1 = (tcnt&0xFF00) >> 8;
 125:   seed2 = (tcnt&0x00FF);
 126:
 127:   last = ((unsigned short) seed1)*last + seed2;
 128:
 129:   return (unsigned char) (last%max);
 130: }
 131:
 132: int shipInBounds(ShipType * array, int index) {
 133:   ShipType * ship = &array[index];
 134:
 135:   if(ship->x < 0 || ship->x > 9 || ship->y < 0 || ship->y > 9 ||
 136:     (ship->orientation == VERTICAL && ship->x + ship->size > 10) ||
 137:     (ship->orientation == HORIZONTAL && ship->y + ship->size > 10)) {
 138:       return 0;
 139:   }
 140:
 141:   return 1;
 142: }
 143:
 144: int checkHit(ShipType * array, int x, int y) {
 145:   int i, j;
 146:   for(i=0; i<5; i++) {
 147:     for(j=0; j<array[i].size; j++) {
 148:       if(array[i].orientation == HORIZONTAL) {
 149:         if(x == array[i].x && y == array[i].y + j) {
 150:           return i;
 151:         }
 152:       }
 153:       else if(x == array[i].x + j && y == array[i].y) {
 154:         return i;
 155:       }
 156:     }
```

```
157:   }
158:
159:   return -1;
160: }
161:
162: int validShipPos(ShipType * array, int index) {
163:   ShipType ship = array[index];
164:   int i;
165:
166:   for(i=0; i<index; i++) {
167:     if(ship.orientation == HORIZONTAL) {
168:       if(array[i].orientation == HORIZONTAL) {
169:         if(ship.x == array[i].x) {
170:           if((ship.y + ship.size > array[i].y &&
171:               ship.y + ship.size <= array[i].y + array[i].size) ||
172:               (ship.y >= array[i].y &&
173:               ship.y < array[i].y + array[i].size) ||
174:               (array[i].y + array[i].size > ship.y &&
175:               array[i].y + array[i].size <= ship.y + ship.size) ||
176:               (array[i].y >= ship.y &&
177:               array[i].y < ship.y + ship.size)) {
178:
179:             return 0;
180:           }
181:         }
182:       }
183:       else {
184:         if(ship.x >= array[i].x &&
185:            ship.x < array[i].x + array[i].size &&
186:            array[i].y >= ship.y &&
187:            array[i].y < ship.y + ship.size) {
188:
189:           return 0;
190:         }
191:       }
192:     }
193:     else {
194:       if(array[i].orientation == HORIZONTAL) {
195:         if(ship.y >= array[i].y &&
196:            ship.y < array[i].y + array[i].size &&
197:            array[i].x >= ship.x &&
198:            array[i].x < ship.x + ship.size) {
199:
200:           return 0;
201:         }
202:       }
203:       else {
204:         if(ship.y == array[i].y) {
205:           if((ship.x + ship.size > array[i].x &&
206:               ship.x + ship.size <= array[i].x + array[i].size) ||
207:               (ship.x >= array[i].x &&
208:               ship.x < array[i].x + array[i].size) ||
209:               (array[i].x + array[i].size > ship.x &&
210:               array[i].x + array[i].size <= ship.x + ship.size) ||
211:               (array[i].x >= ship.x &&
212:               array[i].x < ship.x + ship.size)) {
213:
214:             return 0;
215:           }
216:         }
217:       }
218:     }
219:   }
220:
221:   return 1;
222: }
223:
224: void createField(ShipType * shipArray, int shipSize, AttackType * attackArray, int attackSize)          ¬
     {
225:   int i, j;
226:
227:     for(i=0; i<10; i++) {
228:       for(j=0; j<10; j++) {
229:         field[i][j] = EMPTY;
230:       }
231:     }
232:
233:     for(i=0; i<shipSize; i++) {
```

```
234:       ShipType ship = shipArray[i];
235:       if(ship.orientation == HORIZONTAL) {
236:         field[ship.x][ship.y] = SHIPEND_LEFT;
237:         for(j=1; j<ship.size-1; j++) {
238:           field[ship.x][ship.y+j] = SHIP_HORIZ;
239:         }
240:         field[ship.x][ship.y+ship.size-1] = SHIPEND_RIGHT;
241:       }
242:       else {
243:         field[ship.x][ship.y] = SHIPEND_UP;
244:         for(j=1; j<ship.size-1; j++) {
245:           field[ship.x+j][ship.y] = SHIP_VERT;
246:         }
247:         field[ship.x+ship.size-1][ship.y] = SHIPEND_DOWN;
248:       }
249:     }
250:
251:     for(i=0; i<attackSize; i++) {
252:       AttackType attack = attackArray[i];
253:       field[attack.x][attack.y] = attack.type;
254:     }
255: }
256:
257: void enemyInit(void) {
258:   int i;
259:
260:   for (i=0; i<5; i++) {
261:     ShipType * ship = &computerShips[i];
262:     ship->x = random(10);
263:     ship->y = random(10);
264:     ship->orientation = random(2);
265:
266:     findValidPos(computerShips, i);
267:   }
268: }
269:
270: void enemyPickMove(void) {
271:   int i, x, y, moveFlag, hit;
272:
273:   do {
274:     moveFlag = 0;
275:     x = random(10);
276:     y = random(10);
277:
278:     for(i=0; i<numEnemyAttacks; i++) {
279:       if(enemyAttacks[i].x == x && enemyAttacks[i].y == y) {
280:         moveFlag = 1;
281:       }
282:     }
283:   }while(moveFlag);
284:
285:   enemyAttacks[numEnemyAttacks].x = x;
286:   enemyAttacks[numEnemyAttacks].y = y;
287:   hit = checkHit(ships, x, y);
288:   if(hit == -1) {
289:     enemyAttacks[numEnemyAttacks++].type = MISS;
290:     strcpy(string, "    Miss ");
291:   }
292:   else {
293:     enemyAttacks[numEnemyAttacks++].type = HIT;
294:     strcpy(string, "    Hit ");
295:     Music_EnableOC7(EXPLODE);
296:     asm cli
297:   }
298: }
299:
300: void Game_Init(void) {
301:   state = WELCOME;
302:   numShips = 0;
303:   numEnemyAttacks = 0;
304:   numPlayerAttacks = 0;
305:   cursor.x = 0;
306:   cursor.y = 0;
307:   LED_DDR0 = 1;
308:   LED_DDR1 = 1;
309:   LED_DDR2 = 1;
310:   LED_DDR3 = 1;
311:   LED_DDR4 = 1;
```

```
312:   LED_DDR5 = 1;
313:
314:   LED0 = 1;
315:   LED1 = 1;
316:   LED2 = 1;
317:   LED3 = 1;
318:   LED4 = 1;
319:   LED5 = 1;
320:   Game_Update();
321: }
322:
323: void Game_Update(void) {
324:   int frameFlag = 1;
325:   switch(state) {
326:     case WELCOME:
327:       LCD_Clear(0);
328:       LCD_GoTo(4, 1);
329:       LCD_OutString("       Welcome       ");
330:       LCD_GoTo(5, 1);
331:       LCD_OutString("    to Battleship    ");
332:       Timer_Wait10ms(100);
333:       incState();
334:       break;
335:     case WAITING_FOR_OPPONENT:
336:       enemyInit();
337:       incState();
338:       break;
339:     case PLACING_SHIPS:
340:       //LCD_Clear(0);
341:       LCD_GoTo(3,1);
342:       LCD_OutString("   Place  ");
343:       LCD_GoTo(4,1);
344:       LCD_OutString("   your  ");
345:       LCD_GoTo(5,1);
346:       LCD_OutString("   ships  ");
347:       createField(ships, numShips, enemyAttacks, 0);
348:       LCD_DrawGrid(field);
349:       break;
350:     case PLAYER_TURN_WAITING:
351:       //LCD_Clear(0);
352:       LCD_GoTo(4,1);
353:       LCD_OutString(string);
354:       createField(ships, 0, playerAttacks, numPlayerAttacks);
355:       LCD_DrawGrid(field);
356:       break;
357:     case PLAYER_TURN_DONE:
358:       //LCD_Clear(0);
359:       LCD_GoTo(4,1);
360:       LCD_OutString(string);
361:       createField(ships, 0, playerAttacks, numPlayerAttacks);
362:       LCD_DrawGrid(field);
363:       break;
364:     case OPPONENT_TURN_WAITING:
365:       //LCD_Clear(0);
366:       createField(ships, numShips, enemyAttacks, numEnemyAttacks);
367:       LCD_GoTo(4,1);
368:       LCD_OutString(string);
369:       LCD_DrawGrid(field);
370:       Music_EnableOC7(WHISTLE);
371:       asm cli
372:       Timer_Wait10ms(102);
373:       enemyPickMove();
374:       incState();
375:       break;
376:     case OPPONENT_TURN_DONE:
377:       //LCD_Clear(0);
378:       createField(ships, numShips, enemyAttacks, numEnemyAttacks);
379:       LCD_GoTo(4,1);
380:       LCD_OutString(string);
381:       LCD_DrawGrid(field);
382:       Timer_Wait10ms(140);
383:       incState();
384:       break;
385:     case COMPUTER_SCREEN:
386:       //LCD_Clear(0);
387:       createField(computerShips, 5, playerAttacks, numPlayerAttacks);
388:       LCD_DrawGrid(field);
389:       break;
```

```
390:     case WIN:
391:       //LCD_Clear(0);
392:       LCD_GoTo(4, 1);
393:       LCD_OutString("       You Win       ");
394:       break;
395:     case LOSE:
396:       //LCD_Clear(0);
397:       LCD_GoTo(4, 1);
398:       LCD_OutString("       You Lose      ");
399:       break;
400:   }
401: }
402:
403: int findValidPos(ShipType * array, int index) {
404:   if(validShipPos(array, index) && shipInBounds(array, index)) {
405:     return 1;
406:   }
407:   else {
408:     ShipType * ship = &array[index];
409:     unsigned int tempX   = (ship->x + 9)%10;
410:     unsigned int tempY   = (ship->y + 9)%10;
411:     unsigned int tempDir = ship->orientation ^ 1;
412:
413:     for(ship->orientation = tempDir ^ 1; ship->orientation != tempDir; ship->orientation = (++          ¬
    ship->orientation)%2) {
414:       for(ship->x = (tempX+1)%10; ship->x != tempX; ship->x = (++ship->x)%10) {
415:         for(ship->y = (tempY+1)%10; ship->y != tempY; ship->y = (++ship->y)%10) {
416:           if(validShipPos(array, index) && shipInBounds(array, index)) {
417:             return 1;
418:           }
419:         }
420:       }
421:     }
422:
423:     ship->x = (tempX+1)&0x0F;
424:     ship->y = (tempY+1)&0x0F;
425:     ship->orientation = (tempDir+1)&0x01;
426:
427:     return 0;
428:   }
429: }
430:
431: void flag(void) {
432:   buttonFlag = 0;
433: }
434:
435: void Game_DPad(unsigned char direction) {
436:   unsigned int tempX, tempY;
437:   if(!buttonFlag) {
438:     switch(state) {
439:       case PLACING_SHIPS:
440:         tempX = ships[numShips-1].x;
441:         tempY = ships[numShips-1].y;
442:
443:         do {
444:           switch(direction) {
445:             case UP:
446:               ships[numShips-1].x--;
447:               break;
448:             case DOWN:
449:               ships[numShips-1].x++;
450:               break;
451:             case LEFT:
452:               ships[numShips-1].y--;
453:               break;
454:             case RIGHT:
455:               ships[numShips-1].y++;
456:               break;
457:           }
458:         }while(!validShipPos(ships, numShips-1) && shipInBounds(ships, numShips-1));
459:
460:         if(validShipPos(ships, numShips-1) && shipInBounds(ships, numShips-1)) {
461:           Game_Update();
462:         }
463:         else {
464:           ships[numShips-1].x = tempX&0x0F;
465:           ships[numShips-1].y = tempY&0x0F;
466:         }
```

```
467:         break;
468:       case PLAYER_TURN_WAITING:
469:         switch(direction) {
470:           case UP:
471:             cursor.x = (cursor.x+9)%10;
472:             break;
473:           case DOWN:
474:             cursor.x = (cursor.x+1)%10;
475:             break;
476:           case LEFT:
477:             cursor.y = (cursor.y+9)%10;
478:             break;
479:           case RIGHT:
480:             cursor.y = (cursor.y+1)%10;
481:             break;
482:         }
483:         Game_Update();
484:         break;
485:       case COMPUTER_SCREEN:
486:         incState();
487:         break;
488:     }
489:
490:     buttonFlag = 1;
491:     enableOC6(&flag, DEBOUNCE_DELAY, 8, 1);
492:   }
493: }
494:
495: void LEDflash(void) {
496:   int i;
497:   for(i=0; i<10; i++) {
498:     LED0 ^= 1;
499:     LED1 ^= 1;
500:     LED2 ^= 1;
501:     LED3 ^= 1;
502:     LED4 ^= 1;
503:     LED5 ^= 1;
504:     Timer_Wait1ms(100);
505:   }
506: }
507:
508: void Game_A(void) {
509:   int i, attackFlag;
510:   if(!buttonFlag) {
511:     switch(state) {
512:       case PLACING_SHIPS:
513:         if(findValidPos(ships, numShips)) {
514:           numShips++;
515:         }
516:
517:         if(numShips == 6) {
518:           numShips--;
519:           incState();
520:         }
521:         else {
522:           Game_Update();
523:         }
524:         break;
525:       case PLAYER_TURN_WAITING:
526:         attackFlag = 0;
527:         for(i=0; i<numPlayerAttacks; i++) {
528:           if(playerAttacks[i].x == cursor.x && playerAttacks[i].y == cursor.y) {
529:             attackFlag = 1;
530:           }
531:         }
532:         if(!attackFlag) {
533:           int hit = checkHit(computerShips, cursor.x, cursor.y);
534:           playerAttacks[numPlayerAttacks].x = cursor.x;
535:           playerAttacks[numPlayerAttacks].y = cursor.y;
536:           if(hit == -1) {
537:             playerAttacks[numPlayerAttacks++].type = MISS;
538:             state = PLAYER_TURN_DONE;
539:             Music_EnableOC7(WHISTLE);
540:             asm cli
541:             Timer_Wait10ms(102);
542:             strcpy(string, "   MISS!   ");
543:             Game_Update();
544:             Timer_Wait10ms(100);
```

```
545:           }
546:           else {
547:             computerShips[hit].hits++;
548:             playerAttacks[numPlayerAttacks++].type = HIT;
549:             state = PLAYER_TURN_DONE;
550:             Music_EnableOC7(WHISTLE);
551:             asm cli
552:             Timer_Wait10ms(102);
553:             strcpy(string, "   HIT!   ");
554:             Game_Update();
555:             Music_EnableOC7(EXPLODE);
556:             asm cli
557:             LEDflash();
558:           }
559:           incState();
560:         }
561:         break;
562:       case COMPUTER_SCREEN:
563:         incState();
564:         break;
565:     }
566:
567:     buttonFlag = 1;
568:     enableOC6(&flag, DEBOUNCE_DELAY, 8, 1);
569:   }
570: }
571:
572: void Game_B(void) {
573:   if(!buttonFlag) {
574:     switch(state) {
575:       case PLACING_SHIPS:
576:         ships[numShips-1].orientation ^= 1;
577:         if(validShipPos(ships, numShips-1) && shipInBounds(ships, numShips-1)) {
578:           Game_Update();
579:         }
580:         else {
581:           ships[numShips-1].orientation ^= 1;
582:         }
583:         break;
584:       case PLAYER_TURN_WAITING:
585:         state = COMPUTER_SCREEN;
586:         Game_Update();
587:         break;
588:       case COMPUTER_SCREEN:
589:         incState();
590:         break;
591:
592:     }
593:
594:     buttonFlag = 1;
595:     enableOC6(&flag, DEBOUNCE_DELAY, 8, 1);
596:   }
597: }
598:
599: CursorType Game_GetCursor(void) {
600:   return cursor;
601: }
602:
603: int Game_GetState(void) {
604:   return state;
605: }
```

```
1: #ifndef SWITCH_H
2: #define SWITCH_H
3:
4: void Key_Init(void);
5:
6: void enableOC6(void (*function) (void), unsigned short delay, unsigned short delayCount, unsigne ¬
   d short count);
7: void disableOC6(void);
8: #endif
```

```
  1: #include "defs.h"
  2: #include "game.h"
  3:
  4: // UP     PT5
  5: // DOWN   PT4
  6: // LEFT   PT3
  7: // RIGHT  PT2
  8: // A      PT1
  9: // B      PT0
 10:
 11: static void (*OC6Func) (void);
 12: unsigned static char  OC6Enabled;
 13: unsigned static short OC6Delay;
 14: unsigned static short OC6DelayCount1;
 15: unsigned static short OC6DelayCount2;
 16: unsigned static short OC6Count;
 17:
 18: void Key_Init(void){
 19: asm sei           // make atomic
 20:   DDRT &= ~0x3F;    // PT7,PT6 all rows are output
 21:   PERT  =  0x3F;    // internal pullup on PT3,PT2
 22:   TCTL3 =  0x05;
 23:   TCTL4 =  0x55;   // falling edges IC3,IC2
 24:   TIOS  =  0xC0;
 25:   TIE   =  0x3F;     // Arm only IC3,IC2
 26: asm cli
 27: }
 28:
 29: void enableOC6(void (*function) (void), unsigned short delay, unsigned short delayCount, unsign          ¬
     ed short count) {
 30:   TIE |= 0x40;
 31:   OC6Enabled = 1;
 32:   OC6Func = function;
 33:   OC6Delay = delay;
 34:   OC6DelayCount1 = delayCount;
 35:   OC6DelayCount2 = delayCount;
 36:   OC6Count = count;
 37:   TFLG1 = 0x40;
 38:   TC6 = TCNT + OC6Delay;
 39: }
 40:
 41: void disableOC6(void) {
 42:   TIE &= ~0x40;
 43:   TFLG1 = 0x40;
 44: }
 45:
 46: void interrupt 8 IC0Han(void) {
 47:   TFLG1 = 0x01;
 48:   SW_PTP0;
 49: }
 50:
 51: void interrupt 9 IC1Han(void) {
 52:   TFLG1 = 0x02;
 53:   SW_PTP1;
 54: }
 55:
 56: void interrupt 10 IC2Han(void) {
 57:   TFLG1 = 0x04;
 58:   SW_PTP2;
 59: }
 60:
 61: void interrupt 11 IC3Han(void) {
 62:   TFLG1 = 0x08;
 63:   SW_PTP3;
 64: }
 65:
 66: void interrupt 12 IC4Han(void) {
 67:   TFLG1 = 0x10;
 68:   SW_PTP4;
 69: }
 70:
 71: void interrupt 13 IC5Han(void){
 72:   TFLG1 = 0x20;
 73:   SW_PTP5;
 74: }
 75:
 76: void interrupt 14 OC6Han(void){
 77:   TFLG1 = 0x40;
```

```
78:   if(!OC6DelayCount2) {
79:     OC6DelayCount2 = OC6DelayCount1;
80:     (*OC6Func)();
81:     OC6Count--;
82:     if(!OC6Count) {
83:       disableOC6();
84:     }
85:   }
86:   else {
87:     OC6DelayCount2--;
88:   }
89:
90:   TC6 = TCNT + OC6Delay;
91: }
```

```
1: #define EXPLODE 10672
2: #define WHISTLE 8151
3:
4: void DAC_Init(void);
5: void Music_InitOC7(void);
6: void Music_EnableOC7(int sound);
```

```
#include "defs.h"
#include "music.h"

unsigned const char explode[EXPLODE];

unsigned const char whistle[WHISTLE];

int soundEffect;

// 9S12DP512 SPI1 interface to Max539
// PS6 (out) SCLK synchronous clock
// PS5 (out) MOSI serial data output
// PS7 (out) CS used to latch data into Max539
// PS4 (in) is associated with SPI1, but not used

//---------------------DAC_Init---------------------
// initializes DAC
// Input: none
// Output: none
void DAC_Init(void) {
  SS_DDR = 1; // 1) make PS5, PS6, PS7 outputs, PS4 input
  MOSI_DDR = 1;
  SCK_DDR = 1;  // DDRS

  SPICR1 = 0x58; // 2) enable SPI, no interrupts, master, CPOL=1, CPHA=0
                 // SPI0CR1 = 0101 1000
  SPICR2 = 0x00; // 3) set up PS7 as a regular output
                 // SSOE=0, MODFEN=0 SPI0CR1, SPI0CR2
  SPIBR = 0x00;  // 4) set the baud rate, SPI0BR
  SS = 1;   // 5) make PS7=CS high
}

//---------------------transmitByte---------------------
// outputs byte to DAC
// Input: none
// Output: none
void transmitByte(unsigned char data) {
  unsigned char dummy;
  while(!(SPISR&0x20)) {}  // 1) wait for SPTEF to be 1, SPI0SR
  SPIDR = data;            // 2) write 8-bit data to SPI0DR
  while(!(SPISR&0x80)) {}  // 3) wait for SPIF to be 1, SPI0SR
  dummy = SPIDR;            // 4) clear the SPIF flag by reading the data
                            // dummy = SPI0DR;
}

//---------------------DAC_Out---------------------
// outputs 12 bits to DAC
// Input: none
// Output: none
void DAC_Out(unsigned char data) {
  SS = 0;                     // 1) set PS7=CS low
  //transmitByte((data&0x3F00) >> 8); // 2) transmit most significant 8-bit data to the DAC
  transmitByte(0);
  transmitByte(data);        // 3) transmit least significant 8-bit data to the DAC
  SS = 1;                     // 4) set PS7=CS high
}

//---------------------Music_InitOC0---------------------
// arm output compare 0 for melody
// also enables timer to 43 ns period
// Input: none
// Output: none
void Music_InitOC7(void) {
  TIOS |= 0x80;   // activate TC0 as output compare
}

void Music_EnableOC7(int sound) {
  while(TIE&0x80);
  soundEffect = sound;
  TIE |= 0x80;
  TC7 = TCNT+50;// first interrupt right away
}

// OC handler for melody
interrupt 15 void TC7Handler() {
  unsigned static long i = 0;
  TFLG1 = 0x80;
  if(i >= soundEffect) {
```

```
      i = 0;
      TIE &= ~0x80;
    }
    else {
      if(soundEffect == EXPLODE) {
        DAC_Out(explode[i]);
      }
      else {
        DAC_Out(whistle[i]);
      }
      i++;
    }

    TC7 += 187;
  }
```

```
 1: #ifndef TIMER_H
 2: #define TIMER_H
 3:
 4: // File **********Timer.h***********
 5: // Timer wait routine, 9S12DP512
 6: // assumes PLL is active and E clock is 24 MHz
 7: // TCNT will become 1.5MHz
 8: // Jonathan W. Valvano 1/27/09
 9:
10: //  This example accompanies the books
11: //   "Embedded Microcomputer Systems: Real Time Interfacing",
12: //         Thomson Engineering, copyright (c) 2006,
13: //   "Introduction to Embedded Microcomputer Systems:
14: //    Motorola 6811 and 6812 Simulation", Thomson, copyright (c) 2002
15:
16: // Copyright 2007 by Jonathan W. Valvano, valvano@mail.utexas.edu
17: //    You may use, edit, run or distribute this file
18: //    as long as the above copyright notice remains
19:
20:
21:
22: //---------------------Timer_Init---------------------
23: // activate TCNT at 1.5 MHz
24: // inputs: none
25: // outputs: none
26: void Timer_Init(void);
27:
28:
29: //---------Timer_Wait------------------
30: // fixed time delay
31: // inputs: time to wait in 667ns cycles
32: // outputs: none
33: void Timer_Wait(unsigned short delay);
34:
35: //---------Timer_Wait1ms------------------
36: // fixed time delay
37: // inputs: time to wait in ms
38: // outputs: none
39: // 1500 cycles equals 1ms
40: void Timer_Wait1ms(unsigned short delay);
41:
42: //---------Timer_Wait10ms------------------
43: // fixed time delay
44: // inputs: time to wait in 10ms
45: // outputs: none
46: // 15000 cycles equals 10ms
47: void Timer_Wait10ms(unsigned short delay);
48:
49: #endif
```

```
// File **********Timer.C***********
// Timer wait routines, 9S12DP512
// assumes PLL is active and E clock is 24 MHz
// TCNT will become 1.5MHz
// Jonathan W. Valvano 1/27/09

//  This example accompanies the books
//   "Embedded Microcomputer Systems: Real Time Interfacing",
//         Thomson 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

#include "defs.h"



//--------------------Timer_Init--------------------
// activate TCNT at 1.5 MHz, assumes 24 MHz E clock
// inputs: none
// outputs: none
void Timer_Init(void){
  asm sei         // make ritual atomic
  TSCR1 = 0x80;   // Enable TCNT, 24 MHz E clock
  TSCR2 = 0x04;   // divide by 16 TCNT prescale, TOI disarm
  PACTL = 0;      // timer prescale used for TCNT
/* Bottom three bits of TSCR2 (PR2,PR1,PR0) determine TCNT period
    divide  FastMode(24MHz)    Slow Mode (4MHz)
000   1     42ns  TOF  2.73ms  250ns TOF 16.384ms
001   2     84ns  TOF  5.46ms  500ns TOF 32.768ms
010   4    167ns  TOF  10.9ms    1us TOF 65.536ms
011   8    333ns  TOF  21.8ms   2us TOF 131.072ms
100  16    667ns  TOF  43.7ms   4us TOF 262.144ns
101  32   1.33us  TOF  87.4ms     8us TOF 524.288ms
110  64   2.67us  TOF 174.8ms   16us TOF 1.048576s
111 128   5.33us  TOF 349.5ms   32us TOF 2.097152s */
}


//---------Timer_Wait------------------
// fixed time delay
// inputs: time to wait in 667ns cycles
// outputs: none
void Timer_Wait(unsigned short delay){
unsigned short startTime;
  startTime = TCNT;
  while((TCNT-startTime) <= delay){}
}

//---------Timer_Wait1ms------------------
// fixed time delay
// inputs: time to wait in ms
// outputs: none
// 1500 cycles equals 1ms
void Timer_Wait1ms(unsigned short delay){
  for(;delay>0;delay--){
    Timer_Wait(1500);
  }
}

//---------Timer_Wait10ms------------------
// fixed time delay
// inputs: time to wait in 10ms
// outputs: none
// 15000 cycles equals 10ms
void Timer_Wait10ms(unsigned short delay){
  for(;delay>0;delay--){
    Timer_Wait(15000);
  }
}
```

```
#ifndef LCDG_H
#define LCDG_H
//*****************LCDG.h*************************************
// implementation of the driver for the AGM1264F MODULE
// Jonathan W. Valvano 11/21/09

//  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

//*********************************************************************

// to use it as an 8-line by 21-character display
// initialize it once using
//      LCD_Init
// clear screen with
//      LCD_Clear
// set cursor position using
//      LCD_GoTo
// place ASCII on the screen using
//      LCD_OutChar
//      LCD_OutString
//      LCD_OutDec
//      LCD_OutFix1
//      LCD_OutFix2
//      LCD_OutFix3

// ********* 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);


// ********* 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
```

```
 79: // Output: none
 80: // e.g., LCD_Clear(0);  // makes all pixels off
 81: void LCD_Clear(unsigned char data);
 82:
 83: //-----------------------LCD_GoTo-----------------------
 84: // Move cursor
 85: // Input: line number is 1 to 8, column from 1 to 21
 86: // Output: none
 87: // errors: it will ignore legal addresses
 88: void LCD_GoTo(int line, int column);
 89:
 90:
 91: // ********* LCD_OutChar***********
 92: // Output ASCII character on the
 93: //    AGM1264F 128-bit by 64-bit graphics display
 94: // Input: 7-bit ASCII to display
 95: // Output: none
 96: // letter must be between 32 and 127 inclusive
 97: // execute LCD_GoTo to specify cursor location
 98: void LCD_OutChar(unsigned char letter);
 99:
100:
101: //---------------------LCD_OutString--------------
102: // Display String
103: // Input: pointer to NULL-terminationed ASCII string
104: // Output: none
105: void LCD_OutString(char *pt);
106:
107: void LCD_DrawGrid(unsigned char field[10][10]);
108:
109: #endif
```

```
  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: }
```