```c
/*
 * tank.c
 * Taylor Cowley and Andrew Okazaki
 */


#include <stdint.h>
#include "platform.h"
#include "xparameters.h"
#include "xaxivdma.h"
#include "xio.h"
#include "time.h"
#include "unistd.h"

#include "tank.h"   // Do we normally have to include our own h function?
#define TANK_HEIGHT     8       // Tank is 8 pixels high
#define TANK_INIT_ROW   210     // Tank starts at row 210
#define TANK_INIT_COL   160     // Tank starts at col 160
#define SHELL_LENGTH 3          // Shell is 3 pixels long
#define SHELL_COL_OFFSET 7      // Shell is 7 pixels offset from the tank

#define GREEN 0x0000FF00        // Hex value for green
#define BLACK 0x00000000        // Hex value for black
#define WHITE 0xFFFFFFFF        // Hex value for white

// Packs each horizontal line of the figures into a single 32 bit word.
#define packword15(b14,b13,b12,b11,b10,b9,b8,b7,b6,b5,b4,b3,b2,b1,b0)  \
((b14 << 14) | (b13 << 13) | (b12 << 12) | (b11 << 11) | (b10 << 10) | \
 (b9  << 9 ) | (b8  << 8 ) | (b7  << 7 ) | (b6  << 6 ) | (b5  << 5 ) | \
 (b4  << 4 ) | (b3  << 3 ) | (b2  << 2 ) | (b1  << 1 ) | (b0  << 0 ) )

static const int tank_15x8[TANK_HEIGHT] = {     // This is how we
    packword15(0,0,0,0,0,0,0,1,0,0,0,0,0,0,0),  // Store the tank
    packword15(0,0,0,0,0,0,1,1,1,0,0,0,0,0,0),  // drawing data
    packword15(0,0,0,0,0,0,1,1,1,0,0,0,0,0,0),
    packword15(0,1,1,1,1,1,1,1,1,1,1,1,1,1,0),
    packword15(1,1,1,1,1,1,1,1,1,1,1,1,1,1,1),
    packword15(1,1,1,1,1,1,1,1,1,1,1,1,1,1,1),
    packword15(1,1,1,1,1,1,1,1,1,1,1,1,1,1,1),
    packword15(1,1,1,1,1,1,1,1,1,1,1,1,1,1,1)
};

#define WORD_WIDTH 15

struct tank{        // The struct for our tank
    int row;        // Tank's row
    int col;        // Tank's column
}tank;

struct tank_shell{  // The struct that stores the tank's bullet data
    int row;        // Shell's row
    int col;        // Shell's column
    bool alive;     // Whether it is alive
}tank_shell;


// -----------------------------------------
// Our declaration of functions to be used
```

```
 59 void tank_draw_pixel(uint32_t *framePointer,uint32_t row,uint32_t col,uint32_t color);
 60 // Ending declaration of internal functions
 61 // ----------------------------------------
 62
 63 // This is 100% copied from aliens.c. Eventually it needs to move to its own global
    file
 64 void tank_draw_pixel(uint32_t *framePointer,uint32_t row,uint32_t col,uint32_t color){
 65     #define DRAW_PIXEL_ROW_MULTIPLIER 1280  // 640 * 2 for screen doubling
 66     #define DRAW_PIXEL_ROW 640              // one row offset
 67     #define DRAW_PIXEL_DOUBLE 2             // for doubling
 68
 69     // We draw 4 pixels for every 1 small-screen pixel
 70     framePointer[row*DRAW_PIXEL_ROW_MULTIPLIER + col*DRAW_PIXEL_DOUBLE] = color;
 71     framePointer[row*DRAW_PIXEL_ROW_MULTIPLIER + col*DRAW_PIXEL_DOUBLE+1] = color;
 72     framePointer[row*DRAW_PIXEL_ROW_MULTIPLIER+DRAW_PIXEL_ROW+ col*DRAW_PIXEL_DOUBLE]
    = color;
 73     framePointer[row*DRAW_PIXEL_ROW_MULTIPLIER+DRAW_PIXEL_ROW+ col*DRAW_PIXEL_DOUBLE +
    1] = color;
 74
 75 }
 76
 77 // This initializes our tank at its proper location
 78 void tank_init(){
 79     tank.row = 210;     // Tank starts at this row
 80     tank.col = 160;     // and column
 81 }
 82
 83 // This draws (or erases, via the erase bool) an entire tank.
 84 void tank_draw(uint32_t * framePointer, bool erase){
 85     int color = erase ? BLACK : GREEN ;     // green or black depending on erase
 86     int row, col;                           // init loop vars
 87     for(row=0;row<TANK_HEIGHT;row++){       // Go through tank x pixels
 88         for(col=0;col<WORD_WIDTH;col++){    // and tank y pixels
 89             if ((tank_15x8[row] & (1<<(WORD_WIDTH-col-1)))) {   // If a pixel
 90                 //  Draw the pixel
 91                 tank_draw_pixel(framePointer, row+tank.row,col+tank.col,color);
 92             }
 93         }
 94     }
 95 }
 96
 97 // moves our tank left by a certain number of pixels
 98 void tank_move_left(uint32_t * framePointer){
 99 #define L_0_GREEN   7   // When moving left,
100 #define L_2_GREEN   6   // where to
101 #define L_3_GREEN   1   // draw green
102 #define L_7_GREEN   0   // pixels based on row
103
104 #define L_0_BLACK   8   // When moving left,
105 #define L_2_BLACK   9   // where to
106 #define L_3_BLACK   14  // erase pixels
107 #define L_7_BLACK   15  // based on row
108     tank.col --;        // Move our tank left by a pixel
109     int row;            // Declare loop var
110     for(row = 0; row < TANK_HEIGHT; row++){
111         switch (row){   // Depending on the row
112         case 0:         // Draw/erase proper pixels
113             tank_draw_pixel(framePointer,row+tank.row,L_0_GREEN+tank.col,GREEN);
```

```
114             tank_draw_pixel(framePointer,row+tank.row,L_0_BLACK+tank.col,BLACK);
115             break;
116         case 1: // Cases 1 and 2 are identical
117         case 2:         // Keep drawing/erasing pixels
118             tank_draw_pixel(framePointer,row+tank.row,L_2_GREEN+tank.col,GREEN);
119             tank_draw_pixel(framePointer,row+tank.row,L_2_BLACK+tank.col,BLACK);
120             break;
121         case 3:         // Keep drawing/erasing pixels
122             tank_draw_pixel(framePointer,row+tank.row,L_3_GREEN+tank.col,GREEN);
123             tank_draw_pixel(framePointer,row+tank.row,L_3_BLACK+tank.col,BLACK);
124             break;
125         case 4: // Cases 4, 5, 6, and 7 are all identical.
126         case 5:
127         case 6:
128         case 7:         // Keep drawing/erasing pixels
129             tank_draw_pixel(framePointer,row+tank.row,L_7_GREEN+tank.col,GREEN);
130             tank_draw_pixel(framePointer,row+tank.row,L_7_BLACK+tank.col,BLACK);
131             break;
132         }
133     }
134 }
135
136 //moves our tank right by a certain number of pixels
137 void tank_move_right(uint32_t * framePointer){
138 #define R_0_GREEN 7     // When moving
139 #define R_1_GREEN 8     // right,
140 #define R_2_GREEN 8     // which pixels
141 #define R_3_GREEN 13    // are
142 #define R_4_GREEN 14    // to
143 #define R_5_GREEN 14    // be drawn
144 #define R_6_GREEN 14    // green
145 #define R_7_GREEN 14    // based on the row
146
147 #define R_0_BLACK 6     // When moving
148 #define R_1_BLACK 5     // right,
149 #define R_2_BLACK 5     // which pixels
150 #define R_3_BLACK 0     // are
151 #define R_4_BLACK -1    // to
152 #define R_5_BLACK -1    // be ERASED
153 #define R_6_BLACK -1    // with black
154 #define R_7_BLACK -1    // based on the row
155
156         tank.col ++;    // Move our tank right by a single pixel
157         int r = 0;      // Start our count pointer
158         // Draw and erase the proper pixels for row 0
159         tank_draw_pixel(framePointer, r+tank.row, R_0_GREEN+tank.col, GREEN);
160         tank_draw_pixel(framePointer, r+tank.row, R_0_BLACK+tank.col, BLACK);
161         r++;            // increment row counter
162         // Draw and erase the proper pixels for row 1
163         tank_draw_pixel(framePointer, r+tank.row, R_1_GREEN+tank.col, GREEN);
164         tank_draw_pixel(framePointer, r+tank.row, R_1_BLACK+tank.col, BLACK);
165         r++;            // increment row counter
166         // Draw and erase the proper pixels for row 2
167         tank_draw_pixel(framePointer, r+tank.row, R_2_GREEN+tank.col, GREEN);
168         tank_draw_pixel(framePointer, r+tank.row, R_2_BLACK+tank.col, BLACK);
169         r++;            // increment row counter
170         // Draw and erase the proper pixels for row 3
171         tank_draw_pixel(framePointer, r+tank.row, R_3_GREEN+tank.col, GREEN);
```

```
172         tank_draw_pixel(framePointer, r+tank.row, R_3_BLACK+tank.col, BLACK);
173         r++;            // increment row counter
174         // Draw and erase the proper pixels for row 4
175         tank_draw_pixel(framePointer, r+tank.row, R_4_GREEN+tank.col, GREEN);
176         tank_draw_pixel(framePointer, r+tank.row, R_4_BLACK+tank.col, BLACK);
177         r++;            // increment row counter
178         // Draw and erase the proper pixels for row 5
179         tank_draw_pixel(framePointer, r+tank.row, R_5_GREEN+tank.col, GREEN);
180         tank_draw_pixel(framePointer, r+tank.row, R_5_BLACK+tank.col, BLACK);
181         r++;            // increment row counter
182         // Draw and erase the proper pixels for row 6
183         tank_draw_pixel(framePointer, r+tank.row, R_6_GREEN+tank.col, GREEN);
184         tank_draw_pixel(framePointer, r+tank.row, R_6_BLACK+tank.col, BLACK);
185         r++;            // increment row counter
186         // Draw and erase the proper pixels for row 07
187         tank_draw_pixel(framePointer, r+tank.row, R_7_GREEN+tank.col, GREEN);
188         tank_draw_pixel(framePointer, r+tank.row, R_7_BLACK+tank.col, BLACK);
189 }
190
191 // This creates a shell and initially draws it to the screen
192 void tank_fire(uint32_t * framePointer){
193     if(!tank_shell.alive){          // Only go on if our shell is dead
194         tank_shell.col = tank.col;  // give it
195         tank_shell.row = tank.row;  // a location
196         tank_shell.alive = true;    // make it alive!
197
198         // Tank bullet is 3 pixels long.
199         int row;
200         // So go through all 3 pixels and draw them to the screen!
201         for(row = tank_shell.row-1;row>tank_shell.row-SHELL_LENGTH;row--){
202             tank_draw_pixel(framePointer,row,SHELL_COL_OFFSET+tank_shell.col,WHITE);
203         }
204     }
205 }
206
207 // This moves the shell up the screen
208 void tank_update_bullet(uint32_t * framePointer){
209     if(tank_shell.row<0){           // If shell is off the screen
210         tank_shell.alive = false;   // Kill it
211     }
212     else if(tank_shell.alive){      // Don't do anything if it's dead
213         tank_shell.row -= 1;            // move it up
214         // Erase the lowest pixel, and draw one higher up.
215         tank_draw_pixel(framePointer,tank_shell.row-SHELL_LENGTH,SHELL_COL_OFFSET+tank_
    shell.col, WHITE);
216         tank_draw_pixel(framePointer,tank_shell.row,SHELL_COL_OFFSET+tank_shell.col,
    BLACK);
217     }
218 }
219
220
221
222
223
224
225
226
227
```

228
229
230
231
232
233
234
235
236
237