```c
/*
 * helloworld.c: simple test application
 * Currently used to test lab 3 for Space Invaders.
 * Taylor Cowley and Andrew Okazaki
 */

#include <stdio.h>
#include <stdint.h>
#include "platform.h"
#include "xparameters.h"
#include "xaxivdma.h"
#include "xio.h"
#include "time.h"
#include "unistd.h"
#include "bunkers.h"
#include "tank.h"
#include "interface.h"
#include "aliens.h"
#define DEBUG

#define SCREEN_RES_X 640    // Our screen resolution is 640 * 480
#define SCREEN_RES_Y 480    // Our screen resolution is 640 * 480
#define BLACK 0x00000000    // Hex value for black

void print(char *str);



#define FRAME_BUFFER_0_ADDR 0xC1000000  // Starting location in DDR where we will store the images that we display.

int main() {
    init_platform();                    // Necessary for all programs.
    int Status;                         // Keep track of success/failure of system function calls.
    XAxiVdma videoDMAController;
    // There are 3 steps to initializing the vdma driver and IP.
    // Step 1: lookup the memory structure that is used to access the vdma driver.
    XAxiVdma_Config * VideoDMAConfig = XAxiVdma_LookupConfig(XPAR_AXI_VDMA_0_DEVICE_ID);
    // Step 2: Initialize the memory structure and the hardware.
    if(XST_FAILURE == XAxiVdma_CfgInitialize(&videoDMAController, VideoDMAConfig, XPAR_AXI_VDMA_0_BASEADDR)) {
        xil_printf("VideoDMA Did not initialize.\r\n");
    }
    // Step 3: (optional) set the frame store number.
    if(XST_FAILURE ==  XAxiVdma_SetFrmStore(&videoDMAController, 2, XAXIVDMA_READ)) {
        xil_printf("Set Frame Store Failed.");
    }
    // Initialization is complete at this point.

    // Setup the frame counter. We want two read frames. We don't need any write frames but the
    // function generates an error if you set the write frame count to 0. We set it to 2
    // but ignore it because we don't need a write channel at all.
    XAxiVdma_FrameCounter myFrameConfig;
    myFrameConfig.ReadFrameCount = 2;
```

```
53     myFrameConfig.ReadDelayTimerCount = 10;
54     myFrameConfig.WriteFrameCount =2;
55     myFrameConfig.WriteDelayTimerCount = 10;
56     Status = XAxiVdma_SetFrameCounter(&videoDMAController, &myFrameConfig);
57     if (Status != XST_SUCCESS) {
58        xil_printf("Set frame counter failed %d\r\n", Status);
59        if(Status == XST_VDMA_MISMATCH_ERROR)
60            xil_printf("DMA Mismatch Error\r\n");
61     }
62     // Now we tell the driver about the geometry of our frame buffer and a few other
   things.
63     // Our image is 480 x 640.
64     XAxiVdma_DmaSetup myFrameBuffer;
65     myFrameBuffer.VertSizeInput = 480;    // 480 vertical pixels.
66     myFrameBuffer.HoriSizeInput = 640*4;  // 640 horizontal (32-bit pixels).
67     myFrameBuffer.Stride = 640*4;         // Dont' worry about the rest of the values.
68     myFrameBuffer.FrameDelay = 0;
69     myFrameBuffer.EnableCircularBuf=1;
70     myFrameBuffer.EnableSync = 0;
71     myFrameBuffer.PointNum = 0;
72     myFrameBuffer.EnableFrameCounter = 0;
73     myFrameBuffer.FixedFrameStoreAddr = 0;
74     if(XST_FAILURE == XAxiVdma_DmaConfig(&videoDMAController, XAXIVDMA_READ,
   &myFrameBuffer)) {
75         xil_printf("DMA Config Failed\r\n");
76      }
77     // We need to give the frame buffer pointers to the memory that it will use. This
   memory
78     // is where you will write your video data. The vdma IP/driver then streams it to
   the HDMI
79     // IP.
80      myFrameBuffer.FrameStoreStartAddr[0] = FRAME_BUFFER_0_ADDR;
81      myFrameBuffer.FrameStoreStartAddr[1] = FRAME_BUFFER_0_ADDR + 4*640*480;
82
83      if(XST_FAILURE == XAxiVdma_DmaSetBufferAddr(&videoDMAController, XAXIVDMA_READ,
84                            myFrameBuffer.FrameStoreStartAddr)) {
85          xil_printf("DMA Set Address Failed Failed\r\n");
86      }
87      // Print a sanity message if you get this far.
88      xil_printf("Woohoo! I made it through initialization.\n\r");
89      // Now, let's get ready to start displaying some stuff on the screen.
90      // The variables framePointer and framePointer1 are just pointers to the base
   address
91      // of frame 0 and frame 1.
92      uint32_t* framePointer0 = (uint32_t*) FRAME_BUFFER_0_ADDR;
93      // Just paint some large red, green, blue, and white squares in different
94      // positions of the image for each frame in the buffer (framePointer0 and
   framePointer1).
95      int row=0, col=0;
96           for( row=0; row<SCREEN_RES_Y; row++) {
97              for(col=0; col<SCREEN_RES_X; col++) {
98                 framePointer0[row*SCREEN_RES_X + col] = BLACK;
99              }
100          }
101
102     bunkers_init(framePointer0);       // initialize the bunkers
103     tank_init();                       // initialize the tank
104     tank_draw(framePointer0, false);   // draw the tank
```

```
105
106     interface_draw_line(framePointer0);            // draw the line at the bottom
107     interface_draw_tanks(framePointer0);           // draw the tanks at the top
108     aliens_init(framePointer0);            // initialize aliens
109
110
111
112     // This tells the HDMI controller the resolution of your display (there must be a
    better way to do this).
113     XIo_Out32(XPAR_AXI_HDMI_0_BASEADDR, 640*480);
114
115     // Start the DMA for the read channel only.
116     if(XST_FAILURE == XAxiVdma_DmaStart(&videoDMAController, XAXIVDMA_READ)){
117         xil_printf("DMA START FAILED\r\n");
118     }
119     int frameIndex = 0;
120     // We have two frames, let's park on frame 0. Use frameIndex to index them.
121     // Note that you have to start the DMA process before parking on a frame.
122
123     if (XST_FAILURE == XAxiVdma_StartParking(&videoDMAController, frameIndex,
    XAXIVDMA_READ)) {
124         xil_printf("vdma parking failed\n\r");
125     }
126     char input;
127     srand((unsigned)time( NULL ));
128     while(1){
129     input = getchar();
130     switch(input){
131     case '4':
132         tank_move_left(framePointer0);     // move the tank left
133         break;
134     case '6':
135         tank_move_right(framePointer0);        // move the tank right
136         break;
137     case '8':
138         aliens_move(framePointer0);    // move the aliens
139         break;
140     case '2':
141         aliens_kill(framePointer0);    // Kill an alien
142         break;
143     case '5':
144         tank_fire(framePointer0);      // Make the tank fire
145         break;
146     case '3':
147         alien_missle(framePointer0);   // Make the aliens fire
148         break;
149     case'9':
150         tank_update_bullet(framePointer0); // update all bullets
151         aliens_update_bullets(framePointer0);  // update all bullets
152         break;
153     case '7':
154         bunkers_hit_rand_bunker(framePointer0);    //Erode bunker
155         break;
156     }
157     }
158
159
160    cleanup_platform();
```

```
161
162     return 0;
163 }
164
```

```c
/*
 * aliens.h
 *  Taylor Cowley and Andrew Okazaki
 */

#include <stdbool.h>
#include <stdint.h>
#ifndef ALIENS_H_
#define ALIENS_H_


#endif /* ALIENS_H_ */

void aliens_init(uint32_t * framePointer);  // Initializes the aliens
void aliens_move(uint32_t * framePointer);  // Moves the aliens
void aliens_left(uint32_t * framePointer);  // Moves aliens left
void aliens_right(uint32_t * framePointer); // Move aliens right
void aliens_kill(uint32_t * framePointer);  // Kills a random alien
void alien_missle(uint32_t * framePointer); // Shoots an alien bullet
void aliens_update_bullets(uint32_t * framePointer);    // Updates the bullets

```

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

#include <stdio.h>
#include "platform.h"
#include "xparameters.h"
#include "xaxivdma.h"
#include "xio.h"
#include "time.h"
#include "unistd.h"
#include <stdbool.h>
#include <stdint.h>
#define ALIEN_HEIGHT 8      // Aliens are 8 pixels tall
#define ALIEN_COLUMNS 11    // 11 columns of aliens
#define TOP_TOTAL 11        // 11 aliens in top group
#define LOC_ALIEN_ONE 50    // Pixel where the first alien is
#define MIDDLE_TOTAL 22     // There are 22 total middle aliens
#define BOTTOM_TOTAL 22     // There are 22 total bottom aliens
#define ALIEN_NUM_BULLETS 4 // Aliens can have up to 4 bullets at a time
#define ALIEN_NUM_BULLET_TYPES 2// Aliens have 2 types of bullets to choose from
#define BAD_ADDRESS -1      // Nothing exists at screen address -1
#define MOVE_DOWN_PIXELS 15 // When the aliens move down, they do so 15 pixels
#define LEFT_BOUNDRY    11  // Aliens cannot go more left than this
#define RIGHT_BOUNDRY   307 // Aliens cannot go more right than this
#define BULLET_COL_OFFSET 6 // Bullets appear 11 more right than their alien
#define BULLET_ROW_OFFSET 11// Bullets appear more down than their alien
#define SCREEN_LENGTH   320 // Our screen is 320 pixels wide
#define SCREEN_HEIGHT   240 // Our screen is 240 pixels tall
#define SCREEN_RES_X    640 // Our screen RESOLUTION is 640 pixels wide
#define SCREEN_RES_Y    480 // Our screen RESOLUTION is 480 pixels tall
#define WHITE 0xFFFFFFF     // These
#define BLACK 0x0000000     // are colors
#define WORD_WIDTH 12

// Packs each horizontal line of the figures into a single 32 bit word.
#define packword12(b11,b10,b9,b8,b7,b6,b5,b4,b3,b2,b1,b0) \
        ((b11 << 11) | (b10 << 10) | (b9  << 9 ) | (b8  << 8 ) | (b7  << 7 ) | (b6  << 6 ) \
                | (b5  << 5 ) | (b4  << 4 ) | (b3  << 3 ) | (b2  << 2 ) | (b1  << 1 ) | (b0  << 0 ) )

// -------------------------------------------------
// The following static const ints define the aliens
// We have 3 types of aliens with 2 poses each
static const int32_t alien_top_in_12x8[ALIEN_HEIGHT] = {
        packword12(0,0,0,0,0,1,1,0,0,0,0,0),
        packword12(0,0,0,0,1,1,1,1,0,0,0,0),
        packword12(0,0,0,1,1,1,1,1,1,0,0,0),
        packword12(0,0,1,1,0,1,1,0,1,1,0,0),
        packword12(0,0,1,1,1,1,1,1,1,1,0,0),
        packword12(0,0,0,1,0,1,1,0,1,0,0,0),
        packword12(0,0,1,0,0,0,0,0,0,1,0,0),
        packword12(0,0,0,1,0,0,0,0,0,1,0,0,0) };
static const int32_t alien_top_out_12x8[ALIEN_HEIGHT] = {
        packword12(0,0,0,0,0,1,1,0,0,0,0,0),
        packword12(0,0,0,0,1,1,1,1,0,0,0,0),
```

```
 57         packword12(0,0,0,1,1,1,1,1,1,0,0,0),
 58         packword12(0,0,1,1,0,1,1,0,1,1,0,0),
 59         packword12(0,0,1,1,1,1,1,1,1,1,0,0),
 60         packword12(0,0,0,0,1,0,0,1,0,0,0,0),
 61         packword12(0,0,0,1,0,1,1,0,1,0,0,0),
 62         packword12(0,0,1,0,1,0,0,1,0,1,0,0) };
 63 static const int32_t alien_middle_in_12x8[ALIEN_HEIGHT] = {
 64         packword12(0,0,0,1,0,0,0,0,0,1,0,0),
 65         packword12(0,0,0,0,1,0,0,0,1,0,0,0),
 66         packword12(0,0,0,1,1,1,1,1,1,1,0,0),
 67         packword12(0,0,1,1,0,1,1,1,0,1,1,0),
 68         packword12(0,1,1,1,1,1,1,1,1,1,1,1),
 69         packword12(0,1,1,1,1,1,1,1,1,1,1,1),
 70         packword12(0,1,0,1,0,0,0,0,0,1,0,1),
 71         packword12(0,0,0,0,1,1,0,1,1,0,0,0) };
 72 static const int32_t alien_middle_out_12x8[] = {
 73         packword12(0,0,0,1,0,0,0,0,0,1,0,0),
 74         packword12(0,1,0,0,1,0,0,0,1,0,0,1),
 75         packword12(0,1,0,1,1,1,1,1,1,1,0,1),
 76         packword12(0,1,1,1,0,1,1,1,0,1,1,1),
 77         packword12(0,1,1,1,1,1,1,1,1,1,1,1),
 78         packword12(0,0,1,1,1,1,1,1,1,1,1,0),
 79         packword12(0,0,0,1,0,0,0,0,0,1,0,0),
 80         packword12(0,0,1,0,0,0,0,0,0,0,1,0) };
 81 static const int32_t alien_bottom_in_12x8[ALIEN_HEIGHT] = {
 82         packword12(0,0,0,0,1,1,1,1,0,0,0,0),
 83         packword12(0,1,1,1,1,1,1,1,1,1,1,0),
 84         packword12(1,1,1,1,1,1,1,1,1,1,1,1),
 85         packword12(1,1,1,0,0,1,1,0,0,1,1,1),
 86         packword12(1,1,1,1,1,1,1,1,1,1,1,1),
 87         packword12(0,0,1,1,1,0,0,1,1,1,0,0),
 88         packword12(0,1,1,0,0,1,1,0,0,1,1,0),
 89         packword12(0,0,1,1,0,0,0,0,1,1,0,0) };
 90 static const int32_t alien_bottom_out_12x8[] = {
 91         packword12(0,0,0,0,1,1,1,1,0,0,0,0),
 92         packword12(0,1,1,1,1,1,1,1,1,1,1,0),
 93         packword12(1,1,1,1,1,1,1,1,1,1,1,1),
 94         packword12(1,1,1,0,0,1,1,0,0,1,1,1),
 95         packword12(1,1,1,1,1,1,1,1,1,1,1,1),
 96         packword12(0,0,0,1,1,0,0,1,1,0,0,0),
 97         packword12(0,0,1,1,0,1,1,0,1,1,0,0),
 98         packword12(1,1,0,0,0,0,0,0,0,0,1,1) };
 99 // End of the const ints that define the alien pixels
100 // ---------------------------------------------------
101
102 // ------------------------------------------------------------
103 // These are our internal methods, used only by ourselves
104 // Draws the aliens on the screen - top, middle, and bottom aliens
105 void build_tops(uint32_t * framePointer, const int32_t alien_top[]);
106 void build_middle(uint32_t * framePointer, const int32_t alien_middle[]);
107 void build_bottom(uint32_t * framePointer, const int32_t alien_bottom[]);
108 // Fire a bullet from either a top, middle, or bottom alien
109 int32_t fire_bottom(uint32_t * framePointer, int32_t r);
110 int32_t fire_middle(uint32_t * framePointer, int32_t r);
111 int32_t fire_top(uint32_t * framePointer, int32_t r);
112 // Checks to see whether our aliens are currently capable of shooting
113 bool can_aliens_shoot();
114 // Draws a bullet on the screen
```

```
115 void draw_bullet(uint32_t * framePointer, int32_t bullet, uint32_t color);
116 // Draws a pixel on the screen.
117 void aliens_draw_pixel(uint32_t *framePointer, uint32_t row, uint32_t col,
118         uint32_t color);
119 // End internal method declarations
120 // -------------------------------------------------------------
121
122 // These structs hold all of our aliens.
123 struct top { // Struct for our top aliens
124     int32_t row;
125     int32_t col;bool alive; // alien has row, column, and alive?
126 } top[TOP_TOTAL];
127
128 struct middleAlien { // Struct for our middle aliens
129     int32_t row;
130     int32_t col;bool alive; // alien has row, column, and alive?
131 } middleAlien[MIDDLE_TOTAL];
132
133 struct bottomAlien { // Struct for our bottom aliens
134     int32_t row;
135     int32_t col;bool alive; // alien has row, column, and alive?
136 } bottomAlien[MIDDLE_TOTAL];
137
138 // aliens can have two types of bullet: cross and lightning
139 // cross 0 and 3 are identical
140 typedef enum {
141     cross0, cross1, cross2, cross3, lightning0, lightning1
142 } bullet_type;
143 struct alien_bullet { // Struct that holds our aliens' bullets
144     int32_t row;
145     int32_t col;bool alive; // Bullets have coordinates and alive?
146     bullet_type bullet_type; // Bullets also have a type.
147 } alien_bullet[ALIEN_NUM_BULLETS];
148
149 int32_t alien_count; // a count of how many aliens are alive
150
151 /*
152  * Draws a pixel on the screen. To compensate for our double-resolution screen,
153  * it must draw 4 real pixels for every in-came pixel.
154  */
155 void aliens_draw_pixel(uint32_t *framePointer, uint32_t row, uint32_t col,
156         uint32_t color) {
157 #define DRAW_PIXEL_ROW_MULTIPLIER 1280  // 640 * 2 for screen doubling
158 #define DRAW_PIXEL_ROW 640              // one row offset
159 #define DRAW_PIXEL_DOUBLE 2             // for doubling
160     // We draw 4 pixels for every 1 small-screen pixel
161     framePointer[row * DRAW_PIXEL_ROW_MULTIPLIER + col * DRAW_PIXEL_DOUBLE]
162             = color;
163     framePointer[row * DRAW_PIXEL_ROW_MULTIPLIER + col * DRAW_PIXEL_DOUBLE + 1]
164             = color;
165     framePointer[row * DRAW_PIXEL_ROW_MULTIPLIER + DRAW_PIXEL_ROW + col
166             * DRAW_PIXEL_DOUBLE] = color;
167     framePointer[row * DRAW_PIXEL_ROW_MULTIPLIER + DRAW_PIXEL_ROW + col
168             * DRAW_PIXEL_DOUBLE + 1] = color;
169 }
170
171 //initialize all of the aliens by setting values contained in struct's and printing
    aliens to the screen
```

```c
void aliens_init(uint32_t * framePointer) {
#define ALIEN_TOP_ROW_INIT 30               // Where
#define ALIEN_MIDDLE_ROW_INIT 45            // the
#define ALIEN_MIDDLE2_ROW_INIT 60           // aliens
#define ALIEN_BOTTOM_ROW_INIT 75            // are
#define ALIEN_BOTTOM2_ROW_INIT 90           // initialized to
#define ALIEN_SPACING 15                    // Spacing between aliens
    //local variables, loc is the starting location of alien one on the screen
    int32_t i, loc = LOC_ALIEN_ONE;

    //loops through one row of aliens
    for (i = 0; i < ALIEN_COLUMNS; i++) {
        top[i].row = ALIEN_TOP_ROW_INIT; //set the row of alien tops to 30
        top[i].col = loc;//sets the column of alien tops
        top[i].alive = true;//sets the alien is alive flag

        middleAlien[i].row = ALIEN_MIDDLE_ROW_INIT; //middle aliens
        middleAlien[i].col = loc;//sets column of first row of middle aliens
        middleAlien[i].alive = true;//sets first row of middle aliens to alive
        middleAlien[i + ALIEN_COLUMNS].row = ALIEN_MIDDLE2_ROW_INIT;//sets middle
        middleAlien[i + ALIEN_COLUMNS].col = loc;//sets column second row middle
        middleAlien[i + ALIEN_COLUMNS].alive = true;//sets second row middle alive

        bottomAlien[i].row = ALIEN_BOTTOM_ROW_INIT;//sets bottom aliens
        bottomAlien[i].col = loc;//sets column of first row of bottom aliens
        bottomAlien[i].alive = true;//sets first row of bottom aliens to alive
        bottomAlien[i + ALIEN_COLUMNS].row = ALIEN_BOTTOM2_ROW_INIT;//bottom
        bottomAlien[i + ALIEN_COLUMNS].col = loc;//sets column second row bottom
        bottomAlien[i + ALIEN_COLUMNS].alive = true;//sets second row bottom alive
        loc += ALIEN_SPACING; //controls the column spacing in-between alien
    }

    //now that structs are built draw top, middle, and bottom aliens to screen
    build_tops(framePointer, alien_top_in_12x8); // Top
    build_middle(framePointer, alien_middle_in_12x8); // Middle
    build_bottom(framePointer, alien_bottom_in_12x8); // Bottom
}

// Draws the top aliens on the screen
void build_tops(uint32_t * framePointer, const int32_t alien_top[]) {
    int32_t row, col, i; // initialize variables
    for (i = 0; i < TOP_TOTAL; i++) { //loop through top column of aliens
        for (row = 0; row < ALIEN_HEIGHT; row++) { //loop top aliens' pixels row
            int32_t currentRow = row + top[i].row;// current pixel row of alien
            for (col = 0; col < WORD_WIDTH; col++) { //loop alien's pixel col
                int32_t currentCol = col + top[i].col; //current col of alien
                if ((alien_top[row] & (1 << (WORD_WIDTH - col - 1)))
                        && top[i].alive) {
                    // If our alien is alive and has a pixel there, draw it
                    aliens_draw_pixel(framePointer, currentRow, currentCol,
                            WHITE);
                } else { // If not, erase it.
                    aliens_draw_pixel(framePointer, currentRow, currentCol,
                            BLACK);
                }
            }
        }
    }
```

```
230 }
231
232 // Draws the middle aliens to the screen
233 void build_middle(uint32_t * framePointer, const int32_t alien_middle[]) {
234     int32_t row, col, i; // declare our variables
235     for (i = 0; i < MIDDLE_TOTAL; i++) { // Looping through all the middle aliens
236         for (row = 0; row < ALIEN_HEIGHT; row++) { // Pixel y
237             int32_t currentRow = row + middleAlien[i].row;//current pixel row
238             for (col = 0; col < WORD_WIDTH; col++) {// Pixel x
239                 int32_t currentCol = col + middleAlien[i].col;// current col alien
240                 if ((alien_middle[row] & (1 << (WORD_WIDTH - col - 1)))
241                         && middleAlien[i].alive) {
242                     // If our alien is alive and has a pixel there, draw it
243                     aliens_draw_pixel(framePointer, currentRow, currentCol,
244                             WHITE);
245                 } else { // Otherwise, erase it.
246                     aliens_draw_pixel(framePointer, currentRow, currentCol,
247                             BLACK);
248                 }
249             }
250         }
251     }
252 }
253
254 // Draws the bottom aliens to the screen
255 void build_bottom(uint32_t * framePointer, const int32_t alien_bottom[]) {
256     int32_t row, col, i; // Declare vars
257     for (i = 0; i < BOTTOM_TOTAL; i++) { // Looping through all the bottom aliens
258         for (row = 0; row < ALIEN_HEIGHT; row++) { // looping through y pixels
259             int32_t currentRow = row + bottomAlien[i].row; // current row
260             for (col = 0; col < WORD_WIDTH; col++) { // looping through x pixels
261                 int32_t currentCol = col + bottomAlien[i].col; // current col
262                 if ((alien_bottom[row] & (1 << (WORD_WIDTH - col - 1)))
263                         && bottomAlien[i].alive) {
264                     // If our alien is alive and has a pixel here, draw it
265                     aliens_draw_pixel(framePointer, currentRow, currentCol,
266                             WHITE);
267                 } else { // otherwise, erase it.
268                     aliens_draw_pixel(framePointer, currentRow, currentCol,
269                             BLACK);
270                 }
271             }
272         }
273     }
274 }
275
276 // Does the needful to move the aliens left
277 void aliens_left(uint32_t * framePointer) {
278     int32_t i, row; // Declare loop vars
279     for (i = 0; i < MIDDLE_TOTAL; i++) { // Move every single alien LEFT
280         if (i < TOP_TOTAL) {
281             top[i].col--;
282         } // Move the top aliens LEFT
283         middleAlien[i].col--; // Move the middle aliens LEFT
284         bottomAlien[i].col--; // Move the bottom aliens LEFT
285     }
286     if (alien_count == 0) { // If aliens are out, make them in
287         alien_count = 1;
```

```
288         build_tops(framePointer, alien_top_in_12x8); // Draw top aliens
289         build_middle(framePointer, alien_middle_in_12x8); // Draw mid aliens
290         build_bottom(framePointer, alien_bottom_in_12x8); // Draw bot aliens
291     } else { // And vice versa
292         alien_count = 0;
293         build_tops(framePointer, alien_top_out_12x8); // Draw top aliens
294         build_middle(framePointer, alien_middle_out_12x8); // Draw mid aliens
295         build_bottom(framePointer, alien_bottom_out_12x8); // Draw bot aliens
296     }
297 
298     for (row = 0; row < ALIEN_HEIGHT; row++) { // For all the alien Y pixels
299         for (i = 0; i < MIDDLE_TOTAL; i++) { // For every alien
300             // Erase them for the middle and bottom aliens - top is skinnier
301             aliens_draw_pixel(framePointer, row + bottomAlien[i].row,
302                     WORD_WIDTH + bottomAlien[i].col, BLACK);
303             aliens_draw_pixel(framePointer, row + middleAlien[i].row,
304                     WORD_WIDTH + middleAlien[i].col, BLACK);
305         }
306 
307     }
308 }
309 
310 // Does the needful to move the aliens right
311 void aliens_right(uint32_t * framePointer) {
312     int32_t i, row; // Declare loop vars
313     for (i = 0; i < MIDDLE_TOTAL; i++) { // Move every single alien RIGHT
314         if (i < 11) {
315             top[i].col += 1;
316         } // Move top aliens RIGHT
317         middleAlien[i].col += 1; // Move middle aliens RIGHT
318         bottomAlien[i].col += 1; // Move bottom aliens RIGHT
319     }
320 
321     if (alien_count == 0) { // If aliens are out, make them in
322         alien_count = 1;
323         build_tops(framePointer, alien_top_in_12x8); // Draw top aliens
324         build_middle(framePointer, alien_middle_in_12x8); // Draw mid aliens
325         build_bottom(framePointer, alien_bottom_in_12x8); // Draw bot aliens
326     } else { // And vice versa
327         alien_count = 0;
328         build_tops(framePointer, alien_top_out_12x8); // Draw top aliens
329         build_middle(framePointer, alien_middle_out_12x8); // Draw mid aliens
330         build_bottom(framePointer, alien_bottom_out_12x8); // Draw bot aliens
331     }
332 
333     for (row = 0; row < ALIEN_HEIGHT; row++) { // For all the alien Y pixels
334         for (i = 0; i < MIDDLE_TOTAL; i++) { // For every alien
335             // Erase that column of pixels for mid and bottom. Top not necessary
336             aliens_draw_pixel(framePointer, row + bottomAlien[i].row,
337                     bottomAlien[i].col - 1, BLACK); // Notice it's col-1 bottom
338             aliens_draw_pixel(framePointer, row + middleAlien[i].row,
339                     middleAlien[i].col, BLACK);
340         }
341     }
342 }
343 
344 // Does the needful when aliens hit the left rail
345 void hit_left_rail(uint32_t * framePointer) {
```

```
346     // First we erase the entire top row of alien pixels for moving down.
347     int32_t col, row, i; // declare loop vars
348     for (row = 0; row < ALIEN_HEIGHT; row++) { // Go through alien pixels Y
349         for (col = 0; col < WORD_WIDTH; col++) { // Go through alien pixels X
350             if (((alien_top_out_12x8[row] | alien_top_in_12x8[row]) & (1
351                     << (WORD_WIDTH - col - 1)))) {// if pixel exists here
352                 for (i = 0; i < TOP_TOTAL; i++) { // ERASE IT!
353                     aliens_draw_pixel(framePointer, row + top[i].row,
354                             col + top[i].col, BLACK);
355                 }
356             }
357         }
358     }
359     for (i = 0; i < MIDDLE_TOTAL; i++) { // For all the aliens, move them down
360         if (i < TOP_TOTAL) {
361             top[i].row += MOVE_DOWN_PIXELS;
362         } // Move top aliens down
363         middleAlien[i].row += MOVE_DOWN_PIXELS; // Move mid aliens down
364         bottomAlien[i].row += MOVE_DOWN_PIXELS; // Move bot aliens down
365     }
366     for (row = 0; row < ALIEN_HEIGHT; row++) { // Now to erase pixels on left side
367         for (i = 0; i < MIDDLE_TOTAL; i++) { // For all the middle aliens
368             aliens_draw_pixel(framePointer, row + middleAlien[i].row,
369                     middleAlien[i].col, BLACK);// Erase the pixels on the left
370         }
371     }
372 }
373
374 // Does the needful when aliens hit the right rail
375 void hit_right_rail(uint32_t * framePointer) {
376     // First we erase the entire top row of alien pixels for moving down
377     int32_t col, row, i; // Declare loop vars
378     for (row = 0; row < ALIEN_HEIGHT; row++) { // Go through alien pixels Y
379         for (col = 0; col < WORD_WIDTH; col++) { // Go through alien pixels X
380             if (((alien_top_out_12x8[row] | alien_top_in_12x8[row]) & (1
381                     << (WORD_WIDTH - col - 1)))) {// if pixel exists here
382                 for (i = 0; i < TOP_TOTAL; i++) { // Erase it!
383                     aliens_draw_pixel(framePointer, row + top[i].row,
384                             col + top[i].col, BLACK);
385                 }
386             }
387         }
388     }
389     for (i = 0; i < MIDDLE_TOTAL; i++) { // For all the aliens, move them down
390         if (i < TOP_TOTAL) {
391             top[i].row += MOVE_DOWN_PIXELS;
392         }// Move top aliens down
393         middleAlien[i].row += MOVE_DOWN_PIXELS; // Move mid aliens down
394         bottomAlien[i].row += MOVE_DOWN_PIXELS; // Move bot aliens down
395     }
396     for (row = 0; row < ALIEN_HEIGHT; row++) { // Now to erase pixels on the right
    side
397         for (i = 0; i < TOP_TOTAL; i++) { // Erase the pixels on the right
398             aliens_draw_pixel(framePointer, row + top[i].row,
399                     WORD_WIDTH - 1 + top[i].col, BLACK);
400         }
401     }
402 }
```

```
403
404 // moves the aliens and detects wall boundries and direction changes too!
405 void aliens_move(uint32_t * framePointer) {
406     static int32_t flag;
407     int32_t i, j;
408     for (i = 0; i < ALIEN_COLUMNS; i++) { // Go through every alien column
409         // And see if any alien in that column is alive and has hit left
410         if (top[i].alive || middleAlien[i].alive || middleAlien[i
411                 + ALIEN_COLUMNS].alive || bottomAlien[i].alive || bottomAlien[i
412                 + ALIEN_COLUMNS].alive) {
413             if (top[i].col == LEFT_BOUNDRY) { // If an alien has hit side
414                 flag = 1; // Set the flag that we've hit the side
415                 hit_left_rail(framePointer); // Call hit_rail.
416             }
417         }
418     }
419     for (j = ALIEN_COLUMNS - 1; j >= 0; j--) { // Now to check to see
420         if (top[j].alive || middleAlien[j].alive || middleAlien[j
421                 + ALIEN_COLUMNS].alive || bottomAlien[j].alive || bottomAlien[j
422                 + ALIEN_COLUMNS].alive) {
423             if (top[j].col == RIGHT_BOUNDRY) {// if an alien has hit right.
424                 flag = 0; // false
425                 hit_right_rail(framePointer); // we have hit the right rail
426             }
427         }
428     }
429     if (flag == 1) { // if we are moving right
430         aliens_right(framePointer); // go right
431     } else { // we are actually going left
432         aliens_left(framePointer); // so go left
433     }
434 }
435
436 // Kills a random alien
437 // Currently has a bug that if the last alien dies, infinite loop
438 void aliens_kill(uint32_t * framePointer) {
439     int32_t r = rand() % 55; // Get a random number
440
441     if (r < TOP_TOTAL) { // If we have killed a top
442         if (!top[r].alive) { // Already dead!
443             aliens_kill(framePointer); // Try again
444         } else {
445             top[r].alive = false; // kill the alien
446             build_tops(framePointer, alien_top_in_12x8); // redraw aliens
447         }
448     } else if (r < (TOP_TOTAL + MIDDLE_TOTAL)) { // if we have killed a mid
449         if (!middleAlien[r - TOP_TOTAL].alive) { // Already dead!
450             aliens_kill(framePointer); // try again
451         } else {
452             middleAlien[r - TOP_TOTAL].alive = false; // kill alien
453             build_middle(framePointer, alien_middle_in_12x8);// redraw aliens
454         }
455     } else { // we have killed a bot
456         if (!bottomAlien[r - (TOP_TOTAL + MIDDLE_TOTAL)].alive) { // Already dead!
457             aliens_kill(framePointer); // Try again
458         } else {
459             bottomAlien[r - (TOP_TOTAL + MIDDLE_TOTAL)].alive = false;// Kill alien
460             build_bottom(framePointer, alien_bottom_in_12x8);// redraw aliens
```

```
461         }
462     }
463 }
464 
465 // Returns true if aliens can shoot- that is, if there exists a top alive alien
466 bool can_aliens_shoot() {
467     int32_t i; // Declare loop variable
468     for (i = 0; i < TOP_TOTAL; i++) { // Look at all the top aliense
469         if (top[i].alive) { // If there exists a single alive top alien
470             return true; // We have an alive alien!
471         }
472     }
473     return false; // All the top aliens are dead; we cannot shoot
474 }
475 
476 // Fires a bullet from a random alien
477 void alien_missle(uint32_t * framePointer) {
478     if (!can_aliens_shoot()) { // The aliens can't even shoot! Don't even try.
479         return;
480     }
481 
482     int32_t r = rand() % ALIEN_COLUMNS; // Get a random column
483     int32_t bullet_address = BAD_ADDRESS; // Initialize the address
484     do { // Keep trying to shoot
485         bullet_address = fire_bottom(framePointer, r);
486     } while (bullet_address == BAD_ADDRESS); // until we get a good address
487 
488     // We have a bullet address! now to make it alive and draw it.
489     int32_t i;
490     for (i = 0; i < ALIEN_NUM_BULLETS; i++) {
491         if (alien_bullet[i].alive) { // If we already have a living bullet
492             continue; // Go on to the next one
493         } else { // We have a dead bullet spot- let's alive a bullet here!
494             alien_bullet[i].alive = true;
495             // Randomly choose a bullet type
496             alien_bullet[i].bullet_type
497                     = rand() % ALIEN_NUM_BULLET_TYPES ? cross0 : lightning0;
498             // TODO: This math can be simplified
499             alien_bullet[i].col = bullet_address % SCREEN_RES_X;// Set address
500             alien_bullet[i].row = bullet_address / SCREEN_RES_X;// of bullet
501             draw_bullet(framePointer, i, WHITE); // And draw it!
502             return;
503         }
504     }
505 }
506 
507 // Draws the selected bullet to the screen
508 void draw_bullet(uint32_t * framePointer, int32_t bullet, uint32_t color) {
509 #define PIXEL_LINE_1 1      // These
510 #define PIXEL_LINE_2 2      // defines
511 #define PIXEL_LINE_3 3      // only
512 #define PIXEL_LINE_4 4      // have
513 #define PIXEL_LEFT -1       // meaning
514 #define PIXEL_RIGHT 1       // in this function, so I put them here
515     uint32_t row = alien_bullet[bullet].row; // Current row
516     uint32_t col = alien_bullet[bullet].col; // and column where to draw
517     switch (alien_bullet[bullet].bullet_type) {
518     case cross0: // Cross0 and cross 3 are identically drawn
```

```
519     case cross3: // The only difference is in the state machine where they go
520         // 5 pixels down in a line
521         aliens_draw_pixel(framePointer, row, col, color);
522         aliens_draw_pixel(framePointer, row + PIXEL_LINE_1, col, color);
523         aliens_draw_pixel(framePointer, row + PIXEL_LINE_2, col, color);
524         aliens_draw_pixel(framePointer, row + PIXEL_LINE_3, col, color);
525         aliens_draw_pixel(framePointer, row + PIXEL_LINE_4, col, color);
526 
527         // Crossbar on the cross - right in the middle
528         aliens_draw_pixel(framePointer, row + PIXEL_LINE_2, col + PIXEL_RIGHT,
529                 color);
530         aliens_draw_pixel(framePointer, row + PIXEL_LINE_2, col + PIXEL_LEFT,
531                 color);
532         break;
533     case cross1:
534         // 5 pixels down in a line
535         aliens_draw_pixel(framePointer, row, col, color);
536         aliens_draw_pixel(framePointer, row + PIXEL_LINE_1, col, color);
537         aliens_draw_pixel(framePointer, row + PIXEL_LINE_2, col, color);
538         aliens_draw_pixel(framePointer, row + PIXEL_LINE_3, col, color);
539         aliens_draw_pixel(framePointer, row + PIXEL_LINE_4, col, color);
540 
541         // Crossbar on the cross- on the lower one
542         aliens_draw_pixel(framePointer, row + PIXEL_LINE_3, col + PIXEL_RIGHT,
543                 color);
544         aliens_draw_pixel(framePointer, row + PIXEL_LINE_3, col + PIXEL_LEFT,
545                 color);
546         break;
547     case cross2:
548         // 5 pixels down in a line
549         aliens_draw_pixel(framePointer, row, col, color);
550         aliens_draw_pixel(framePointer, row + PIXEL_LINE_1, col, color);
551         aliens_draw_pixel(framePointer, row + PIXEL_LINE_2, col, color);
552         aliens_draw_pixel(framePointer, row + PIXEL_LINE_3, col, color);
553         aliens_draw_pixel(framePointer, row + PIXEL_LINE_4, col, color);
554 
555         // Crossbar on the cross- on the upper one
556         aliens_draw_pixel(framePointer, row + PIXEL_LINE_1, col + PIXEL_RIGHT,
557                 color);
558         aliens_draw_pixel(framePointer, row + PIXEL_LINE_1, col + PIXEL_LEFT,
559                 color);
560         break;
561     case lightning0:
562         // 5 pixels down - starting left then right, then going back left
563         aliens_draw_pixel(framePointer, row, col + PIXEL_LEFT, color);
564         aliens_draw_pixel(framePointer, row + PIXEL_LINE_1, col, color);
565         aliens_draw_pixel(framePointer, row + PIXEL_LINE_2, col + PIXEL_RIGHT,
566                 color);
567         aliens_draw_pixel(framePointer, row + PIXEL_LINE_3, col, color);
568         aliens_draw_pixel(framePointer, row + PIXEL_LINE_4, col + PIXEL_LEFT,
569                 color);
570         break;
571     case lightning1:
572         // 5 pixels down - starting right then left, then back right
573         aliens_draw_pixel(framePointer, row, col + PIXEL_RIGHT, color);
574         aliens_draw_pixel(framePointer, row + PIXEL_LINE_1, col, color);
575         aliens_draw_pixel(framePointer, row + PIXEL_LINE_2, col + PIXEL_LEFT,
576                 color);
```

```
577         aliens_draw_pixel(framePointer, row + PIXEL_LINE_3, col, color);
578         aliens_draw_pixel(framePointer, row + PIXEL_LINE_4, col + PIXEL_RIGHT,
579                 color);
580         break;
581     }
582
583 }
584
585 // This sees if our bottom alien at index r is alive to shoot
586 int32_t fire_bottom(uint32_t * framePointer, int32_t r) {
587     if (!bottomAlien[r + ALIEN_COLUMNS].alive) { // If the very bottom alien is dead
588         if (!bottomAlien[r].alive) {// AND the second row alien is also dead
589             return fire_middle(framePointer, r); // Try to make a higher alien shoot
    it
590         } else { // the bottom alien is dead, but the second-row one is alive
591             // This is the starting coordinate of the bullet.
592             return (bottomAlien[r].row + BULLET_COL_OFFSET + 1) * SCREEN_RES_X
593                     + (BULLET_COL_OFFSET + bottomAlien[r].col);
594         }
595     } else { // The very bottom alien is alive and needs to shoot
596         // Time to return the starting position of the bullet!
597         return (bottomAlien[r + ALIEN_COLUMNS].row + BULLET_COL_OFFSET + 1)
598                 * SCREEN_RES_X + (BULLET_COL_OFFSET + bottomAlien[r
599                 + ALIEN_COLUMNS].col);
600     }
601 }
602
603 // This sees if either middle alien at index r is alive to shoot
604 int32_t fire_middle(uint32_t * framePointer, int32_t r) {
605     if (!middleAlien[r + ALIEN_COLUMNS].alive) { // If the very bottom (middle) alien
    is dead
606         if (!middleAlien[r].alive) {// AND the second row (middle) alien is dead
607             return fire_top(framePointer, r); // Top row alien has to fire
608         } else { // the bottom alien is dead, but the second-row one is alive
609             // This is the starting coordinate of the bullet
610             return (middleAlien[r].row + BULLET_COL_OFFSET) * SCREEN_RES_X
611                     + (BULLET_COL_OFFSET + middleAlien[r].col);
612         }
613     } else { // The bottom alien is alive and needs to fire
614         // This is the starting coordinate of the bullet
615         return (middleAlien[r + ALIEN_COLUMNS].row + BULLET_COL_OFFSET)
616                 * SCREEN_RES_X + (BULLET_COL_OFFSET + middleAlien[r
617                 + ALIEN_COLUMNS].col);
618     }
619 }
620
621 // This sees to see if our top alien at index r is alive to shoot
622 int32_t fire_top(uint32_t * framePointer, int32_t r) {
623     if (!top[r].alive) { // Our top alien is dead.
624         return BAD_ADDRESS; // We failed to fire a missle! return -1
625     } else { // Our alien is alive!
626         return (top[r].row + BULLET_COL_OFFSET) * SCREEN_RES_X
627                 + (BULLET_COL_OFFSET + top[r].col); // Return good address
628     }
629 }
630
631 // Updates alien bullets. erases previous one, increments type, and redraws.
632 void aliens_update_bullets(uint32_t * framePointer) {
```

```
633     int32_t i; // Declare loop var
634     for (i = 0; i < ALIEN_NUM_BULLETS; i++) { // Cycle through all bullets
635         if (alien_bullet[i].row > SCREEN_HEIGHT) { // If bullet off screen
636             alien_bullet[i].alive = false; // kill it
637         } else if (alien_bullet[i].alive) { // If bullet is alive
638             draw_bullet(framePointer, i, BLACK); // erase to prep redraw
639 
640             switch (alien_bullet[i].bullet_type) { // Increment bullet type
641             case cross0: // mid, going down
642                 alien_bullet[i].bullet_type = cross1; // bar go down
643                 break;
644             case cross1: // down
645                 alien_bullet[i].bullet_type = cross3; // bar go mid
646                 break;
647             case cross2: // up
648                 alien_bullet[i].bullet_type = cross0; // bar go down
649                 break;
650             case cross3: // mid, going up
651                 alien_bullet[i].bullet_type = cross2; // bar go up
652                 break;
653             case lightning0:// left lightning
654                 alien_bullet[i].bullet_type = lightning1; // go right
655                 break;
656             case lightning1:// right lightning
657                 alien_bullet[i].bullet_type = lightning0; // go left
658                 break;
659             }
660             alien_bullet[i].row++; // Move bullet down
661             draw_bullet(framePointer, i, WHITE); // redraw bullet
662         }
663     }
664 }
665 
```

```c
/*
 * bunkers.h
 * Taylor Cowley and Andrew Okazaki
 */

#ifndef BUNKERS_H_
#define BUNKERS_H_

#include <stdint.h>


// Initializes the bunkers - draws them to the screen
void bunkers_init(uint32_t * framePointer);

// Draws the bunkers to the screen
void bunkers_build(uint32_t * framePointer);

// Hits a random bunker in a random place
void bunkers_hit_rand_bunker(uint32_t * framePointer);

#endif /* BUNKERS_H_ */

```

```c
/*
 * bunkers.c
 * Taylor Cowley and Andrew Okazaki
 */
#include <stdio.h>
#include <stdint.h>
#include <stdbool.h>
#include "platform.h"
#include "xparameters.h"
#include "xaxivdma.h"
#include "xio.h"
#include "time.h"
#include "unistd.h"

#include "bunkers.h"

#define BUNKER_HEIGHT 18        // Bunkers are 18 pixels high
#define BUNKER_DAMAGE_HEIGHT 6  // Each bunnker square is size 6
#define BUNKER_ROW 175          // All bunkers are at row 175
#define BUNKER_SIZE 10          // All bunkers have 10 sections
#define BUNKER_0 0              // Gotta have
#define BUNKER_1 1              // constants to
#define BUNKER_2 2              // represent
#define BUNKER_3 3              // each bunker
#define BUNKER_DAMAGE_0 0       // Gotta have
#define BUNKER_DAMAGE_1 1       // different
#define BUNKER_DAMAGE_2 2       // damage
#define BUNKER_DAMAGE_3 3       // values
#define BUNKER_DEAD     4       // Damage bunker has when it is dead
#define BUNKER_ROWS     18      // How many rows each bunker has
#define BUNKER_COLS     24      // How many columns each bunker has

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

#define DAMAGE_WORD_WIDTH 6
#define WORD_WIDTH 24
#define NUM_OF_BUNKERS 4
#define LOC_BUNKER_ONE 60   // Divided this by 2 because screen is half

// ------------------------------------------
//  hardcoded static const stuff

// Necessary for storing bunker damage data
#define packword6(b5,b4,b3,b2,b1,b0) \
        ((b5  << 5 ) | (b4  << 4 ) | (b3  << 3 ) | (b2  << 2 ) | (b1  << 1 ) | (b0  << 0 ) )

// Necessary for storing the bunker data
#define packword24(b23,b22,b21,b20,b19,b18,b17,b16,b15,b14,b13,b12,b11,b10,b9,b8,b7,b6,b5,b4,b3,b2,b1,b0) \
        ((b23 << 23) | (b22 << 22) | (b21 << 21) | (b20 << 20) | (b19 << 19) | (b18 << 18) | (b17 << 17) | (b16 << 16) | \
                (b15 << 15) | (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 ) )
```

```
53 // Shape of the entire bunker.
54 static const int32_t bunker_24x18[BUNKER_HEIGHT] = {
55         packword24(0,0,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0,0,0),
56         packword24(0,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0,0),
57         packword24(0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0),
58         packword24(1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1),
59         packword24(1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1),
60         packword24(1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1),
61         packword24(1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1),
62         packword24(1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1),
63         packword24(1,1,1,1,1,1,1,1,1,1,0,0,0,0,1,1,1,1,1,1,1,1,1),
64         packword24(1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,1,1,1,1,1,1,1,1),
65         packword24(1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1),
66         packword24(1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1),
67         packword24(1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1),
68         packword24(1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1),
69         packword24(1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1),
70         packword24(1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1),
71         packword24(1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1),
72         packword24(1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1)};
73
74 // First time a bunker is hit, the first damage that happens
75 static const int32_t bunkerDamage0_6x6[BUNKER_DAMAGE_HEIGHT] = {
76         packword6(0,1,1,0,0,0), packword6(0,0,0,0,0,1), packword6(1,1,0,1,0,0),
77         packword6(1,0,0,0,0,0), packword6(0,0,1,1,0,0), packword6(0,0,0,0,1,0)};
78
79 // Second time a bunker is hit, this is its damage
80 static const int32_t bunkerDamage1_6x6[BUNKER_DAMAGE_HEIGHT] = {
81         packword6(1,1,1,0,1,0), packword6(1,0,1,0,0,1), packword6(1,1,0,1,1,1),
82         packword6(1,0,0,0,0,0), packword6(0,1,1,1,0,1), packword6(0,1,1,0,1,0)};
83
84 // Third time a bunker is hit, this is its damage
85 static const int32_t bunkerDamage2_6x6[BUNKER_DAMAGE_HEIGHT] = {
86         packword6(1,1,1,1,1,1), packword6(1,0,1,1,0,1), packword6(1,1,0,1,1,1),
87         packword6(1,1,0,1,1,0), packword6(0,1,1,1,0,1), packword6(1,1,1,1,1,1)};
88
89 // Fourth time a bunker is hit, this is its damage
90 static const int32_t bunkerDamage3_6x6[BUNKER_DAMAGE_HEIGHT] = {
91         packword6(1,1,1,1,1,1), packword6(1,1,1,1,1,1), packword6(1,1,1,1,1,1),
92         packword6(1,1,1,1,1,1), packword6(1,1,1,1,1,1), packword6(1,1,1,1,1,1)};
93
94 //  End hardcoded static const stuff
95 // ------------------------------------------------------
96
97 struct bunker{      // Holds the data for each bunker
98     int32_t row;        // Where it is
99     int32_t col;        // on the screen
100    int32_t damage;     // What damage level the bunker is at
101    int32_t pixel[];    // A bunker is made out of squares- whether it's alive/dead
102 }bunker[3];
103
104
105 // These arrays show how decayed each part of the bunker is.
106 int32_t bunker_zero[BUNKER_SIZE] = { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
107 int32_t bunker_one[BUNKER_SIZE] = { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
108 int32_t bunker_two[BUNKER_SIZE] = { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
109 int32_t bunker_three[BUNKER_SIZE] = { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
110
```

```
111 // -----------------------------------------------------
112 // Declaration for internal functions
113 void bunkers_draw_pixel(uint32_t *framePointer,uint32_t row,uint32_t col,uint32_t
    color);
114 void bunker0(int32_t r, uint32_t * framePointer);
115 void bunker1(int32_t r, uint32_t * framePointer);
116 void bunker2(int32_t r, uint32_t * framePointer);
117 void bunker3(int32_t r, uint32_t * framePointer);
118 void degrigation_patern(int32_t row, int32_t col, int32_t bunker_number, int32_t
    damage, uint32_t * framePointer);
119 void bunker_hit(uint32_t * framePointer, int32_t location, int32_t bunker_num);
120 // End internal function declaration
121 // -----------------------------------------------------
122
123 /*
124  * Draws a pixel on the screen. To compensate for our double-resolution screen,
125  * it must draw 4 real pixels for every in-came pixel.
126  */
127 void bunkers_draw_pixel(uint32_t *framePointer,uint32_t row,uint32_t col,uint32_t
    color){
128 #define DRAW_PIXEL_ROW_MULTIPLIER 1280  // 640 * 2 for screen doubling
129 #define DRAW_PIXEL_ROW 640              // one row offset
130 #define DRAW_PIXEL_DOUBLE 2             // for doubling
131
132     // We draw 4 pixels for every 1 small-screen pixel
133     framePointer[row*DRAW_PIXEL_ROW_MULTIPLIER + col*DRAW_PIXEL_DOUBLE] = color;
134     framePointer[row*DRAW_PIXEL_ROW_MULTIPLIER + col*DRAW_PIXEL_DOUBLE+1] = color;
135     framePointer[row*DRAW_PIXEL_ROW_MULTIPLIER+DRAW_PIXEL_ROW+ col*DRAW_PIXEL_DOUBLE]
    = color;
136     framePointer[row*DRAW_PIXEL_ROW_MULTIPLIER+DRAW_PIXEL_ROW+ col*DRAW_PIXEL_DOUBLE +
    1] = color;
137 }
138
139 // Initializes the bunkers
140 void bunkers_init(uint32_t * framePointer){
141     int32_t i, loc = LOC_BUNKER_ONE;    //
142     for(i = 0; i < NUM_OF_BUNKERS ; i++){
143         bunker[i].row = BUNKER_ROW; // Divided by 2 because screen is half
144         bunker[i].col = loc;        // which column it is at
145         bunker[i].damage = 0;       // Start undamaged
146         loc += LOC_BUNKER_ONE;      // Add by the offset
147     }
148     bunkers_build(framePointer);    // Draw the bunkers on the screen
149 }
150
151
152 void bunkers_build(uint32_t * framePointer){
153     int32_t row, col, b;                            // Declare loop vars
154     for(row=0;row<BUNKER_ROWS;row++){                   // Go through rows
155         for(col=0;col<BUNKER_COLS;col++){               // Go through cols
156             if ((bunker_24x18[row] & (1<<(WORD_WIDTH-col-1)))) {// if pixel
157                 for(b = 0; b <NUM_OF_BUNKERS; b++){// draw that pixel every time
158                     bunkers_draw_pixel(framePointer,row+bunker[b].row,col+bunker[b].co
    l,GREEN);
159                 }
160             }
161         }
162     }
```

```
163 }
164
165 // This randomly selects a bunker and randomly destroys part of it
166 void bunkers_hit_rand_bunker(uint32_t * framePointer){
167     int32_t r = rand()%NUM_OF_BUNKERS;  // Randomly pick a bunker
168
169     switch (r){
170     case BUNKER_0:              // Depending on the bunker number, destroy one in
    that.
171         bunker0(rand()%BUNKER_SIZE, framePointer);
172         break;
173     case BUNKER_1:              // bunker 1
174         bunker1(rand()%BUNKER_SIZE, framePointer);
175         break;
176     case BUNKER_2:              // bunker 2
177         bunker2(rand()%BUNKER_SIZE, framePointer);
178         break;
179     case BUNKER_3:              // bunker 3
180         bunker3(rand()%BUNKER_SIZE, framePointer);
181         break;
182     }
183 }
184
185
186 #define HIT_ROW_LOC_0 0     // For
187 #define HIT_ROW_LOC_1 0     // every
188 #define HIT_ROW_LOC_2 0     // location
189 #define HIT_ROW_LOC_3 0     // it has
190 #define HIT_ROW_LOC_4 6     // a specific
191 #define HIT_ROW_LOC_5 6     // row
192 #define HIT_ROW_LOC_6 6     // associated
193 #define HIT_ROW_LOC_7 6     // with
194 #define HIT_ROW_LOC_8 12    // each
195 #define HIT_ROW_LOC_9 12    // location
196
197 #define HIT_COL_LOC_0 0     // for
198 #define HIT_COL_LOC_1 6     // every
199 #define HIT_COL_LOC_2 12    // location
200 #define HIT_COL_LOC_3 18    // it has
201 #define HIT_COL_LOC_4 0     // a specific
202 #define HIT_COL_LOC_5 6     // column
203 #define HIT_COL_LOC_6 12    // associated
204 #define HIT_COL_LOC_7 18    // with
205 #define HIT_COL_LOC_8 0     // each
206 #define HIT_COL_LOC_9 18    // location
207
208 #define HIT_0 0             // There
209 #define HIT_1 1             // isn't
210 #define HIT_2 2             // an easy
211 #define HIT_3 3             // way to
212 #define HIT_4 4             // loop through
213 #define HIT_5 5             // all these
214 #define HIT_6 6             // yet,
215 #define HIT_7 7             // so we have
216 #define HIT_8 8             // every location
217 #define HIT_9 9             // hard-coded in.
218
219 // Put a hit on bunker 0 at a certain location
```

```
220 void bunker0(int32_t r, uint32_t * framePointer){
221     if(bunker_zero[r]== BUNKER_DEAD){   // If our bunker is already dead here
222         bunkers_hit_rand_bunker(framePointer);      // call rand kill bunker again
223         return;
224     }
225     switch(r){                          // Based on where the hit is
226     case HIT_0:     // Hit in location 0. Row 0 and column 0
227         degrigation_patern(HIT_ROW_LOC_0, HIT_COL_LOC_0, BUNKER_0, bunker_zero[r],
    framePointer);
228         break;
229     case HIT_1:     // Hit in location 1. Row 1 and column 1
230         degrigation_patern(HIT_ROW_LOC_1, HIT_COL_LOC_1, BUNKER_0, bunker_zero[r],
    framePointer);
231         break;
232     case HIT_2:     // Hit in location 2. Row 2 and column 2
233         degrigation_patern(HIT_ROW_LOC_2, HIT_COL_LOC_2, BUNKER_0, bunker_zero[r],
    framePointer);
234         break;
235     case HIT_3:     // Hit in location 3. Row 3 and column 3
236         degrigation_patern(HIT_ROW_LOC_3, HIT_COL_LOC_3, BUNKER_0, bunker_zero[r],
    framePointer);
237         break;
238     case HIT_4:     // Hit in location 4. Row 4 and column 4
239         degrigation_patern(HIT_ROW_LOC_4, HIT_COL_LOC_4, BUNKER_0, bunker_zero[r],
    framePointer);
240         break;
241     case HIT_5:     // Hit in location 5. Row 5 and column 5
242         degrigation_patern(HIT_ROW_LOC_5, HIT_COL_LOC_5, BUNKER_0, bunker_zero[r],
    framePointer);
243         break;
244     case HIT_6:     // Hit in location 6. Row 6 and column 6
245         degrigation_patern(HIT_ROW_LOC_6, HIT_COL_LOC_6, BUNKER_0, bunker_zero[r],
    framePointer);
246         break;
247     case HIT_7:     // Hit in location 7. Row 7 and column 7
248         degrigation_patern(HIT_ROW_LOC_7, HIT_COL_LOC_7, BUNKER_0, bunker_zero[r],
    framePointer);
249         break;
250     case HIT_8:     // Hit in location 8. Row 8 and column 8
251         degrigation_patern(HIT_ROW_LOC_8, HIT_COL_LOC_8, BUNKER_0, bunker_zero[r],
    framePointer);
252         break;
253     case HIT_9:     // Hit in location 9. Row 9 and column 9
254         degrigation_patern(HIT_ROW_LOC_9, HIT_COL_LOC_9, BUNKER_0, bunker_zero[r],
    framePointer);
255         break;
256     }
257     bunker_zero[r]++;
258 }
259
260 // Put a hit on bunker 1 at a certain location
261 void bunker1(int32_t r, uint32_t * framePointer){
262     if(bunker_one[r]== BUNKER_DEAD){    // If our bunker is already dead here
263         bunkers_hit_rand_bunker(framePointer);      // call rand kill bunker again
264         return;
265     }
266     switch(r){                          // Based on where the hit is
267     case HIT_0:     // Hit in location 0. Row 0 and column 0
```

```
268         degrigation_patern(HIT_ROW_LOC_0, HIT_COL_LOC_0, BUNKER_1, bunker_one[r],
    framePointer);
269         break;
270     case HIT_1:     // Hit in location 1. Row 1 and column 1
271         degrigation_patern(HIT_ROW_LOC_1, HIT_COL_LOC_1, BUNKER_1, bunker_one[r],
    framePointer);
272         break;
273     case HIT_2:     // Hit in location 2. Row 2 and column 2
274         degrigation_patern(HIT_ROW_LOC_2, HIT_COL_LOC_2, BUNKER_1, bunker_one[r],
    framePointer);
275         break;
276     case HIT_3:     // Hit in location 3. Row 3 and column 3
277         degrigation_patern(HIT_ROW_LOC_3, HIT_COL_LOC_3, BUNKER_1, bunker_one[r],
    framePointer);
278         break;
279     case HIT_4:     // Hit in location 4. Row 4 and column 4
280         degrigation_patern(HIT_ROW_LOC_4, HIT_COL_LOC_4, BUNKER_1, bunker_one[r],
    framePointer);
281         break;
282     case HIT_5:     // Hit in location 5. Row 5 and column 5
283         degrigation_patern(HIT_ROW_LOC_5, HIT_COL_LOC_5, BUNKER_1, bunker_one[r],
    framePointer);
284         break;
285     case HIT_6:     // Hit in location 6. Row 6 and column 6
286         degrigation_patern(HIT_ROW_LOC_6, HIT_COL_LOC_6, BUNKER_1, bunker_one[r],
    framePointer);
287         break;
288     case HIT_7:     // Hit in location 7. Row 7 and column 7
289         degrigation_patern(HIT_ROW_LOC_7, HIT_COL_LOC_7, BUNKER_1, bunker_one[r],
    framePointer);
290         break;
291     case HIT_8:     // Hit in location 8. Row 8 and column 8
292         degrigation_patern(HIT_ROW_LOC_8, HIT_COL_LOC_8, BUNKER_1, bunker_one[r],
    framePointer);
293         break;
294     case HIT_9:     // Hit in location 9. Row 9 and column 9
295         degrigation_patern(HIT_ROW_LOC_9, HIT_COL_LOC_9, BUNKER_1, bunker_one[r],
    framePointer);
296         break;
297     }
298     bunker_one[r]++;
299 }
300
301 // Put a hit on bunker 2 at a certain location
302 void bunker2(int32_t r, uint32_t * framePointer){
303     if(bunker_two[r]== BUNKER_DEAD){    // If our bunker is already dead here
304         bunkers_hit_rand_bunker(framePointer);      // call rand kill bunker again
305         return;
306     }
307     switch(r){                          // Based on where the hit is
308     case HIT_0:     // Hit in location 0. Row 0 and column 0
309         degrigation_patern(HIT_ROW_LOC_0, HIT_COL_LOC_0, BUNKER_2, bunker_two[r],
    framePointer);
310         break;
311     case HIT_1:     // Hit in location 1. Row 1 and column 1
312         degrigation_patern(HIT_ROW_LOC_1, HIT_COL_LOC_1, BUNKER_2, bunker_two[r],
    framePointer);
313         break;
```

```
314     case HIT_2:     // Hit in location 2. Row 2 and column 2
315         degrigation_patern(HIT_ROW_LOC_2, HIT_COL_LOC_2, BUNKER_2, bunker_two[r],
    framePointer);
316         break;
317     case HIT_3:     // Hit in location 3. Row 3 and column 3
318         degrigation_patern(HIT_ROW_LOC_3, HIT_COL_LOC_3, BUNKER_2, bunker_two[r],
    framePointer);
319         break;
320     case HIT_4:     // Hit in location 4. Row 4 and column 4
321         degrigation_patern(HIT_ROW_LOC_4, HIT_COL_LOC_4, BUNKER_2, bunker_two[r],
    framePointer);
322         break;
323     case HIT_5:     // Hit in location 5. Row 5 and column 5
324         degrigation_patern(HIT_ROW_LOC_5, HIT_COL_LOC_5, BUNKER_2, bunker_two[r],
    framePointer);
325         break;
326     case HIT_6:     // Hit in location 6. Row 6 and column 6
327         degrigation_patern(HIT_ROW_LOC_6, HIT_COL_LOC_6, BUNKER_2, bunker_two[r],
    framePointer);
328         break;
329     case HIT_7:     // Hit in location 7. Row 7 and column 7
330         degrigation_patern(HIT_ROW_LOC_7, HIT_COL_LOC_7, BUNKER_2, bunker_two[r],
    framePointer);
331         break;
332     case HIT_8:     // Hit in location 8. Row 8 and column 8
333         degrigation_patern(HIT_ROW_LOC_8, HIT_COL_LOC_8, BUNKER_2, bunker_two[r],
    framePointer);
334         break;
335     case HIT_9:     // Hit in location 9. Row 9 and column 9
336         degrigation_patern(HIT_ROW_LOC_9, HIT_COL_LOC_9, BUNKER_2, bunker_two[r],
    framePointer);
337         break;
338     }
339     bunker_two[r]++;
340 }
341
342 // Put a hit on bunker 3 at a certain location
343 void bunker3(int32_t r, uint32_t * framePointer){
344     if(bunker_three[r]== BUNKER_DEAD){  // If our bunker is already dead here
345         bunkers_hit_rand_bunker(framePointer);      // call rand kill bunker again
346         return;
347     }
348     switch(r){                          // Based on where the hit is
349     case HIT_0:     // Hit in location 0. Row 0 and column 0
350         degrigation_patern(HIT_ROW_LOC_0, HIT_COL_LOC_0, BUNKER_3, bunker_three[r],
    framePointer);
351         break;
352     case HIT_1:     // Hit in location 1. Row 1 and column 1
353         degrigation_patern(HIT_ROW_LOC_1, HIT_COL_LOC_1, BUNKER_3, bunker_three[r],
    framePointer);
354         break;
355     case HIT_2:     // Hit in location 2. Row 2 and column 2
356         degrigation_patern(HIT_ROW_LOC_2, HIT_COL_LOC_2, BUNKER_3, bunker_three[r],
    framePointer);
357         break;
358     case HIT_3:     // Hit in location 3. Row 3 and column 3
359         degrigation_patern(HIT_ROW_LOC_3, HIT_COL_LOC_3, BUNKER_3, bunker_three[r],
    framePointer);
```

```
360         break;
361     case HIT_4:     // Hit in location 4. Row 4 and column 4
362         degrigation_patern(HIT_ROW_LOC_4, HIT_COL_LOC_4, BUNKER_3, bunker_three[r],
    framePointer);
363         break;
364     case HIT_5:     // Hit in location 5. Row 5 and column 5
365         degrigation_patern(HIT_ROW_LOC_5, HIT_COL_LOC_5, BUNKER_3, bunker_three[r],
    framePointer);
366         break;
367     case HIT_6:     // Hit in location 6. Row 6 and column 6
368         degrigation_patern(HIT_ROW_LOC_6, HIT_COL_LOC_6, BUNKER_3, bunker_three[r],
    framePointer);
369         break;
370     case HIT_7:     // Hit in location 7. Row 7 and column 7
371         degrigation_patern(HIT_ROW_LOC_7, HIT_COL_LOC_7, BUNKER_3, bunker_three[r],
    framePointer);
372         break;
373     case HIT_8:     // Hit in location 8. Row 8 and column 8
374         degrigation_patern(HIT_ROW_LOC_8, HIT_COL_LOC_8, BUNKER_3, bunker_three[r],
    framePointer);
375         break;
376     case HIT_9:     // Hit in location 9. Row 9 and column 9
377         degrigation_patern(HIT_ROW_LOC_9, HIT_COL_LOC_9, BUNKER_3, bunker_three[r],
    framePointer);
378         break;
379     }
380     bunker_three[r]++;
381 }
382
383 // This goes through all the bunkers and destroys them according to our pattern
384 void degrigation_patern(int32_t row, int32_t col, int32_t bunker_num, int32_t damage,
    uint32_t * framePointer){
385     int32_t r,c;
386     for(r=0;r<BUNKER_DAMAGE_HEIGHT;r++){        // Go through rows
387         for(c=0;c<DAMAGE_WORD_WIDTH;c++){       // and columns
388             if (damage == BUNKER_DAMAGE_0       // 0 damage level
389                     && (bunkerDamage0_6x6[r] & (1<<(DAMAGE_WORD_WIDTH-c-1)))){
390                 // If we need to erase a pixel here, do so.
391                 bunkers_draw_pixel(framePointer,r+row+bunker[bunker_num].row
392                         ,c+col+bunker[bunker_num].col, BLACK);
393
394             }else if(damage == BUNKER_DAMAGE_1  // 1 damage level
395                     && (bunkerDamage1_6x6[r] & (1<<(DAMAGE_WORD_WIDTH-c-1)))){
396                 // If we need to erase a pixel here, do so.
397                 bunkers_draw_pixel(framePointer,r+row+bunker[bunker_num].row
398                         ,c+col+bunker[bunker_num].col, BLACK);
399
400             }else if(damage == BUNKER_DAMAGE_2  // 2 damage level
401                     && (bunkerDamage2_6x6[r] & (1<<(DAMAGE_WORD_WIDTH-c-1)))){
402                 // If we need to erase a pixel here, do so.
403                 bunkers_draw_pixel(framePointer,r+row+bunker[bunker_num].row
404                         ,c+col+bunker[bunker_num].col, BLACK);
405
406             }else if(damage == BUNKER_DAMAGE_3  // 3 damage level
407                     && (bunkerDamage3_6x6[r] & (1<<(DAMAGE_WORD_WIDTH-c-1)))){
408                 // If we need to erase a pixel here, do so.
409                 bunkers_draw_pixel(framePointer,r+row+bunker[bunker_num].row
410                         ,c+col+bunker[bunker_num].col, BLACK);
```

```
411             }
412         }
413     }
414 }
415
416
417
```

```c
/*
 * interface.h
 * Taylor Cowley and Andrew Okazaki
 */

#ifndef INTERFACE_H_
#define INTERFACE_H_


#endif /* INTERFACE_H_ */

// Draws the line at the bottom of the screen
void interface_draw_line(uint32_t * framePointer);

// Draws the "extra life" tanks
void interface_draw_tanks(uint32_t * framePointer);

```

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

#include <stdio.h>
#include <stdint.h>
#include "platform.h"
#include "xparameters.h"
#include "xaxivdma.h"
#include "xio.h"
#include "time.h"
#include "unistd.h"
#define TANK_HEIGHT 8
#define GREEN 0x0000FF00
#define GAME_X 320          // How wide our game screen is
#define LINE_Y 225          // Where the line at the bottom goes

#define EXTRA_TANK_0 250    // X coordinate of extra tanks
#define EXTRA_TANK_1 270    // X coordinate of extra tanks
#define EXTRA_TANK_2 290    // X coordinate of extra tanks
#define EXTRA_TANK_Y_OFFSET 5   // How far down the extra tanks are

// 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 ) )

 // This seems like a *very bad* way to store the tank data, but this is what
 // we are doing for the moment.
static const int tank_15x8[TANK_HEIGHT] =
{
packword15(0,0,0,0,0,0,0,1,0,0,0,0,0,0,0),
packword15(0,0,0,0,0,0,1,1,1,0,0,0,0,0,0),
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

// ----------------------------------------
// Our declaration of functions to be used
void interface_draw_pixel(uint32_t * framePointer,uint32_t row,uint32_t col,uint32_t color);
// Ending declaration of internal functions
// ----------------------------------------

// This is 100% copied from aliens.c. Eventually it needs to move to its own global file
void interface_draw_pixel(uint32_t * framePointer,uint32_t row,uint32_t col,uint32_t color){
    #define DRAW_PIXEL_ROW_MULTIPLIER 1280  // 640 * 2 for screen doubling
    #define DRAW_PIXEL_ROW 640              // one row offset
```

```
56    #define DRAW_PIXEL_DOUBLE 2             // for doubling
57
58    // We draw 4 pixels for every 1 small-screen pixel
59    framePointer[row*DRAW_PIXEL_ROW_MULTIPLIER + col*DRAW_PIXEL_DOUBLE] = color;
60    framePointer[row*DRAW_PIXEL_ROW_MULTIPLIER + col*DRAW_PIXEL_DOUBLE+1] = color;
61    framePointer[row*DRAW_PIXEL_ROW_MULTIPLIER+DRAW_PIXEL_ROW+ col*DRAW_PIXEL_DOUBLE] =
  color;
62    framePointer[row*DRAW_PIXEL_ROW_MULTIPLIER+DRAW_PIXEL_ROW+ col*DRAW_PIXEL_DOUBLE +
  1] = color;
63
64}
65
66// This draws the green line at the bottom of the screen
67void interface_draw_line(uint32_t * framePointer){
68    int row, col;                           // Initialize
69    row = LINE_Y;                           // variables
70    for(col=0;col<GAME_X;col++){            // Go along the screen and draw
71        interface_draw_pixel(framePointer, row, col, GREEN);
72    }
73}
74
75// This draws the extra tanks to the screen
76void interface_draw_tanks(uint32_t * framePointer){
77     int row, col;                          // Init loop vars
78     for(row=0;row<TANK_HEIGHT;row++){      // Go through width
79         for(col=0;col<WORD_WIDTH;col++){   // and height
80             if((tank_15x8[row] & (1<<(WORD_WIDTH-col-1)))) {   // and draw 3 tanks
81                 interface_draw_pixel(framePointer, row+EXTRA_TANK_Y_OFFSET,
  col+EXTRA_TANK_0, GREEN);
82                 interface_draw_pixel(framePointer, row+EXTRA_TANK_Y_OFFSET,
  col+EXTRA_TANK_1, GREEN);
83                 interface_draw_pixel(framePointer, row+EXTRA_TANK_Y_OFFSET,
  col+EXTRA_TANK_2, GREEN);
84             }
85         }
86     }
87}
88
```

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

#ifndef TANK_H_
#define TANK_H_

#include <stdint.h>
#include <stdbool.h>

void tank_init();
// moves our tank left by a certain number of pixels
void tank_move_left(uint32_t * framePointer);
// moves our tank right by a certain number of pixels
void tank_move_right(uint32_t * framePointer);

// This simply draws the tank on the screen, where it is at now.
void tank_draw(uint32_t * framePointer, bool erase);

// Alives a shell and draws it to the screen
void tank_fire(uint32_t * framePointer);

// Moves the shell up on the screen
void tank_update_bullet(uint32_t * framePointer);

#endif /* TANK_H_ */

```

```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