```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 "util.h"
#include "bunkers.h"


#define NUM_BUNKERS 4           // We have 4 bunkers
#define NUM_SQUARES 10          // Each bunker has 10 sections
#define NUM_SQUARES_IN_LINE 4   // In a line there are 4 sections
#define BUNKER_ROW 60           // Row the bunkers live on
#define LOC_BUNKER_ONE 60       // Wher ethe first bunker is
#define SQUARE_INCREMENT 6      // Each section is this square
#define LEFT_STRUT_ROW 12       // The extra sections live here
#define LEFT_STRUT_COL 0        // and here
#define RIGHT_STRUT_ROW 12      // and here
#define RIGHT_STRUT_COL 18      // and here
#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 BUNKER_ROW_LOC 175      // Where our bunker livess?
#define BUNKER_DAMAGE_1 1       // how
#define BUNKER_DAMAGE_2 2       // much
#define BUNKER_DAMAGE_3 3       // damage
#define BUNKER_DAMAGE_4 4       // we have
#define WHITE 0xFFFFFFF     // These
#define BLACK 0x0000000     // are colors
#define ZERO_DAMAGE 0           // No damage!
#define BUFFER 1                // One pixel buffer needed sometimes

// -----------------------------------------------
//  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 ) )
```

```
// Shape of the entire bunker.
static const int32_t bunker_24x18[BUNKER_ROWS] = {
        packword24(0,0,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0,0,0),
        packword24(0,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0,0),
        packword24(0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0),
        packword24(1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1),
        packword24(1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1),
        packword24(1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1),
        packword24(1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1),
        packword24(1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1),
        packword24(1,1,1,1,1,1,1,1,1,0,0,0,0,1,1,1,1,1,1,1,1,1,1),
        packword24(1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,1,1,1,1,1,1,1,1),
        packword24(1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1),
        packword24(1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1),
        packword24(1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1),
        packword24(1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1),
        packword24(1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1),
        packword24(1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1),
        packword24(1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1)};

// First time a bunker is hit, the first damage that happens
static const int32_t bunkerDamage0_6x6[SQUARE_INCREMENT] = {
        packword6(0,1,1,0,0,0), packword6(0,0,0,0,0,1), packword6(1,1,0,1,0,0),
        packword6(1,0,0,0,0,0), packword6(0,0,1,1,0,0), packword6(0,0,0,0,1,0)};

// Second time a bunker is hit, this is its damage
static const int32_t bunkerDamage1_6x6[SQUARE_INCREMENT] = {
        packword6(1,1,1,0,1,0), packword6(1,0,1,0,0,1), packword6(1,1,0,1,1,1),
        packword6(1,0,0,0,0,0), packword6(0,1,1,1,0,1), packword6(0,1,1,0,1,0)};

// Third time a bunker is hit, this is its damage
static const int32_t bunkerDamage2_6x6[SQUARE_INCREMENT] = {
        packword6(1,1,1,1,1,1), packword6(1,0,1,1,0,1), packword6(1,1,0,1,1,1),
        packword6(1,1,0,1,1,0), packword6(0,1,1,1,0,1), packword6(1,1,1,1,1,1)};

// Fourth time a bunker is hit, this is its damage
static const int32_t bunkerDamage3_6x6[SQUARE_INCREMENT] = {
        packword6(1,1,1,1,1,1), packword6(1,1,1,1,1,1), packword6(1,1,1,1,1,1),
        packword6(1,1,1,1,1,1), packword6(1,1,1,1,1,1), packword6(1,1,1,1,1,1)};


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


// -------------------------------------
// Internal function declaration
void squares_init();
void bunker_degrade(uint32_t i, uint32_t j);
// end internal function declaration
// -------------------------------------

struct bunker{                  // Our bunker
    uint32_t row;               // has a row
    uint32_t col;               // and a column
    struct squares{             // and 10 sections
        uint32_t row;               // Which have their rows
```

```
        uint32_t col;               // and columns
        uint32_t damage;            // and damage
    }squares[NUM_SQUARES];
}bunker[NUM_BUNKERS];

uint32_t * frame;               // Variable to store the screen frame


// For debugging. Prints out a pixel for each section of bunker
void bunkers_debug_print(){
    int i,j;
    for(i=0;i<NUM_BUNKERS;i++){
        //xil_printf("Bunker %d: %d row, %d col\n\r", i, bunker[i].row, bunker[i].col);
        for(j=0;j<NUM_SQUARES;j++){
            //xil_printf("Bunker %d, square %d: %d row %d col\n\r", i, j,
bunker[i].squares[j].row,bunker[i].squares[j].col);
            util_draw_pixel(frame,SQUARE_INCREMENT+bunker[i].squares[j].row,
SQUARE_INCREMENT+bunker[i].squares[j].col, 0x00000FF);
            util_draw_pixel(frame,bunker[i].squares[j].row,bunker[i].squares[j].col,
0xFFF0000);
        }
    }
}

// Initializes the bunkers
void bunkers_init(uint32_t * framePointer){
    int32_t i, loc = LOC_BUNKER_ONE;    //
    for(i = 0; i < NUM_BUNKERS ; i++){
        bunker[i].row = BUNKER_ROW_LOC; // Divided by 2 because screen is half
        bunker[i].col = loc;        // which column it is at
        loc += LOC_BUNKER_ONE;      // Add by the offset
    }
    bunkers_build(framePointer);    // Draw the bunkers on the screen

    squares_init(); // init the bunker squares
}

// Initializes the bunker sections
void squares_init(){
    uint32_t i, j, row_count, col_count;            // Var init
    row_count = 0;
    col_count = 0;
    for(i = 0; i < NUM_BUNKERS; i++){               // Go through all bunkers
        for(j = 0; j < NUM_SQUARES-2; j++){         // And all squares
            if(j == NUM_SQUARES_IN_LINE){
                row_count += SQUARE_INCREMENT;
                col_count = 0;
            }
            //// And give them addresses and damage
            bunker[i].squares[j].row = bunker[i].row + row_count;
            bunker[i].squares[j].col = bunker[i].col + col_count;
            bunker[i].squares[j].damage = ZERO_DAMAGE;
            col_count += SQUARE_INCREMENT;
        }
        // Now to initialize the last two sections
        bunker[i].squares[j].row = bunker[i].row + LEFT_STRUT_ROW;
        bunker[i].squares[j].col = bunker[i].col + LEFT_STRUT_COL;
        bunker[i].squares[j].damage = ZERO_DAMAGE;
```

```
            j++;
            bunker[i].squares[j].row = bunker[i].row + RIGHT_STRUT_ROW;
            bunker[i].squares[j].col = bunker[i].col + RIGHT_STRUT_COL;
            bunker[i].squares[j].damage = ZERO_DAMAGE;
            row_count = 0;
            col_count = 0;
        }
    }

    // Draws the bunkers
    void bunkers_build(uint32_t * framePointer){
        frame = framePointer;
        int32_t row, col, b;                                // Declare loop vars
        for(row=0;row<BUNKER_ROWS;row++){                       // Go through rows
            for(col=0;col<BUNKER_COLS;col++){                   // Go through cols
                if ((bunker_24x18[row] & (1<<(BUNKER_COLS-col-1)))) {// if pixel
                    for(b = 0; b <NUM_BUNKERS; b++){// draw that pixel every time
                        util_draw_pixel(framePointer,row+bunker[b].row,col+bunker[b].col,GREEN
    );
                    }
                }
            }
        }
    }



    // Is our bunker hit by something?
    bool bunkers_detect_collision(uint32_t row, uint32_t col, bool forceDestroy){
        uint32_t i, j;
        for(i = 0; i < NUM_BUNKERS; i++){
            for(j=0; j < NUM_SQUARES; j++){
                if(bunker[i].squares[j].damage < BUNKER_DAMAGE_4 && bunker[i].squares[j].row +
                        SQUARE_INCREMENT >= row&&  bunker[i].squares[j].row <= row){
                    // If we have been hit
                    if((col <=  bunker[i].squares[j].col + SQUARE_INCREMENT+BUFFER)
                            && (col >= bunker[i].squares[j].col-BUFFER)){
                        // and we have been hit
                        if(forceDestroy){       // an alien crashed into us
                            bunker_degrade(i,j);// completely
                            bunker_degrade(i,j);// destroy
                            bunker_degrade(i,j);// totally
                            bunker_degrade(i,j);//
                        }else {                 // Just a bullet
                            bunker_degrade(i,j);// only one destroy
                        }
                        return true;            // We have been hit!
                    }
                }
            }
        }
        return false;                           // Noone got hit, sorry
    }

    void bunker_degrade(uint32_t i, uint32_t j){
        bunker[i].squares[j].damage++;
        int32_t r,c;
        for(r=0;r<SQUARE_INCREMENT;r++){        // Go through rows
```

```
        for(c=0;c<SQUARE_INCREMENT;c++){        // and columns
            if (bunker[i].squares[j].damage == BUNKER_DAMAGE_1 && (bunkerDamage0_6x6[r] &
(1<<(SQUARE_INCREMENT-c-1)))){
                // If we need to erase a pixel here, do so.
                util_draw_pixel(frame,r+bunker[i].squares[j].row,c+bunker[i].squares[j].c
ol, BLACK);
            }else if(bunker[i].squares[j].damage == BUNKER_DAMAGE_2 &&
(bunkerDamage1_6x6[r] & (1<<(SQUARE_INCREMENT-c-1)))){
                // If we need to erase a pixel here, do so.
                util_draw_pixel(frame,r+bunker[i].squares[j].row,c+bunker[i].squares[j].c
ol, BLACK);

            }else if(bunker[i].squares[j].damage == BUNKER_DAMAGE_3 // 2 damage level
                    && (bunkerDamage2_6x6[r] & (1<<(SQUARE_INCREMENT-c-1)))){
                // If we need to erase a pixel here, do so.
                util_draw_pixel(frame,r+bunker[i].squares[j].row,c+bunker[i].squares[j].c
ol, BLACK);

            }else if(bunker[i].squares[j].damage == BUNKER_DAMAGE_4 // 3 damage level
                    && (bunkerDamage3_6x6[r] & (1<<(SQUARE_INCREMENT-c-1)))){
                // If we need to erase a pixel here, do so.
                util_draw_pixel(frame,r+bunker[i].squares[j].row,c+bunker[i].squares[j].c
ol, BLACK);
            }
        }
    }
}
```