```c
/*
 * main.c: simple test application
 *
 * main file for startup. Inits platform and calls game_controller
 *
 *  Broderick Gardner
 *  Benjamin Gardner
 */

#include <stdio.h>
#include <limits.h>
#include "platform.h"
#include "xparameters.h"
#include "xaxivdma.h"
#include "xio.h"
#include "xgpio.h"
#include "mb_interface.h"   // provides the microblaze interrupt enables, etc.
#include "xintc_l.h"        // Provides handy macros for the interrupt controller.
#include "time.h"
#include "unistd.h"

#include "gpio.h"
#include "timer.h"
#include "render.h"
#include "control.h"
#include "timing.h"

//Defines
//#define MAX_IDLE_COUNT 66651  //For 1,000,000
//#define MAX_IDLE_COUNT 13320  //For 200,000
#define MAX_IDLE_COUNT 31236    //For 500,000
//Variables
volatile s32 idle_count = 0;
volatile s32 avg_util = 0;
volatile s32 max_util = 0;
volatile s32 max_avg_util = 0;

//Prototypes
void interrupt_init();
void interrupt_handler_dispatcher();

#define DEBUG
//#define PRINT_UTIL
void print(char *str);
void init();

int main() {
    init();
    u32 i = 10;
    while (1) {
        idle_count = 0;
        while (timer_flag == 0) {
            idle_count++;
        }
        timer_flag = 0;

#ifdef DEBUG

        //      if (timer_missed) {
        //          print("Timer missed!\n\r");
        //          timer_missed = 0;
        //      }


        idle_count = MAX_IDLE_COUNT - idle_count;
        if (max_util < 0)
            max_util = 0;
        else if (idle_count > max_util) {
            max_util = idle_count;
        }

        if (!(--i)) {
            avg_util = (avg_util - (avg_util >> 2)) + (max_util >> 2);
            xil_printf("utilization: %d, %d\n\r", avg_util, max_util);
            i = 10;
```

```
            max_util = -1;
        }

#endif

        //Tick state machines
        timing_game_tick();
        //      game_controller_run(); //run the game
    }

    cleanup_platform();

    return 0;
}

void init() {

    init_platform(); // Necessary for all programs.

    gpio_init();
    interrupt_init();

    microblaze_enable_interrupts();

}

void interrupt_init() {

    microblaze_register_handler(interrupt_handler_dispatcher, NULL);
    XIntc_EnableIntr(
            XPAR_INTC_0_BASEADDR,
            (XPAR_FIT_TIMER_0_INTERRUPT_MASK
                    | XPAR_PUSH_BUTTONS_5BITS_IP2INTC_IRPT_MASK));
    XIntc_MasterEnable(XPAR_INTC_0_BASEADDR);
}

void interrupt_handler_dispatcher(void* ptr) {
    XIntc_MasterDisable(XPAR_AXI_INTC_0_BASEADDR);

    int intc_status = XIntc_GetIntrStatus(XPAR_INTC_0_BASEADDR);
    // Check the FIT interrupt first.
    if (intc_status & XPAR_FIT_TIMER_0_INTERRUPT_MASK) {
        XIntc_AckIntr(XPAR_INTC_0_BASEADDR, XPAR_FIT_TIMER_0_INTERRUPT_MASK);
        timer_interrupt_handler();
    }
    // Check the push buttons.
    if (intc_status & XPAR_PUSH_BUTTONS_5BITS_IP2INTC_IRPT_MASK) {
        gpio_interrupt_handler();
        XIntc_AckIntr(XPAR_INTC_0_BASEADDR, XPAR_PUSH_BUTTONS_5BITS_IP2INTC_IRPT_MASK);
    }

    XIntc_MasterEnable(XPAR_AXI_INTC_0_BASEADDR);
}

/*
 * bmp.c
 *
 *  Created on: Sep 28, 2017
 *      Author: superman
 */

#include <stdint.h>
#include "bmp.h"

//Macros
//Utilities for packing bits into bitmaps

#define
packword32(b31,b30,b29,b28,b27,b26,b25,b24,b23,b22,b21,b20,b19,b18,b17,b16,b15,b14,b13,b12,b11,b10,b9,b8,b7,b6,b5,b4,b3,b2,b1,b
0) ( \
(b31 << 31) | (b30 << 30) | (b29 << 29) | (b28 << 28) | (b27 << 27) | (b26 << 26) | (b25 << 25) | (b24 << 24) |
      \
 (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 ) |
```

```
 (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 )
        \
 )

#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 )
        \
 )

#define packword16(b15,b14,b13,b12,b11,b10,b9,b8,b7,b6,b5,b4,b3,b2,b1,b0) (\
 ((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 )  \
 )<< 4)

#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 )  \
 ) << 2)    //Padding
//Modified to pad bitmap!
#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 )  \
 ) << 2)    //Padding
#define packword6(b5,b4,b3,b2,b1,b0)    (   \
(b5  << 5 ) | (b4  << 4 ) | (b3  << 3 ) | (b2  << 2 ) | (b1  << 1 ) | (b0  << 0 )   \
 )

#define packword4(b3,b2,b1,b0) ( \
(b3 << 3) | (b2 << 2) | (b1 << 1) | (b0 << 0)   )

#define packword3(b2,b1,b0) ( \
(b2 << 2) | (b1 << 1) | (b0 << 0) \
)

const point_t bmp_tank_dim = { { BMP_TANK_W, BMP_TANK_H } };
const point_t bmp_alien_dim = { { BMP_ALIEN_W, BMP_ALIEN_H } };
const point_t bmp_bunker_dim = { { BMP_BUNKER_W, BMP_BUNKER_H } };
const point_t bmp_bunker_block_dim = {
        { BMP_BUNKER_BLOCK_W, BMP_BUNKER_BLOCK_H } };
const point_t bmp_missile_dim = { { BMP_BULLET_W, BMP_BULLET_H } };
const point_t bmp_saucer_dim = { { BMP_SAUCER_W, BMP_SAUCER_H } };

//define packword 27, for the words score and lives.
#define packword27(b26, b25, b24, b23, b22, b21, b20, b19, b18, b17, b16, b15, b14, b13, b12, b11, b10, b9, b8, b7, b6, b5, b4,
b3, b2, b1, b0)\
        ((b26 << 26) |(b25 << 25) |(b24 << 24) |(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 ) )

//Alien explosion
const u32 bmp_alien_explosion_12x10[] = { packword12(0, 0, 0, 0, 0, 0,
        1, 0, 0, 0, 0, 0), packword12(0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0),
        packword12(1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0), packword12(0, 1, 0, 0,
                1, 0, 0, 0, 1, 0, 0, 0), packword12(0, 0, 0, 0, 0, 0, 0, 0, 0,
                0, 1, 1), packword12(1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0),
        packword12(0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0), packword12(0, 0, 1, 0,
                0, 0, 0, 0, 1, 0, 0, 1), packword12(0, 1, 0, 0, 0, 1, 0, 0, 1,
                0, 0, 0), packword12(0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0) };

//Alien top in
const u32 bmp_alien_top_in_12x8[] = { 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, 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, 1, 0, 0, 0) };

//Alien top out
const u32 bmp_alien_top_out_12x8[] = { 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, 0, 1, 0, 0, 1, 0, 0, 0, 0),
	packword12(0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0), packword12(0, 0, 1, 0,
		1, 0, 0, 1, 0, 1, 0, 0) };

//Alien middle in
const u32 bmp_alien_middle_in_12x8[] = { packword12(0, 0, 0, 1, 0, 0,
	0, 0, 0, 1, 0, 0), packword12(0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0),
	packword12(0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0), packword12(0, 0, 1, 1,
		0, 1, 1, 1, 0, 1, 1, 0), packword12(0, 1, 1, 1, 1, 1, 1, 1, 1,
		1, 1, 1), packword12(0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1),
	packword12(0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1), packword12(0, 0, 0, 0,
		1, 1, 0, 1, 1, 0, 0, 0) };

//Alien middle out
const u32 bmp_alien_middle_out_12x8[] = { packword12(0, 0, 0, 1, 0, 0,
	0, 0, 0, 1, 0, 0), packword12(0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1),
	packword12(0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1), packword12(0, 1, 1, 1,
		0, 1, 1, 1, 0, 1, 1, 1), packword12(0, 1, 1, 1, 1, 1, 1, 1, 1,
		1, 1, 1), packword12(0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0),
	packword12(0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0), packword12(0, 0, 1, 0,
		0, 0, 0, 0, 0, 0, 1, 0) };

//Alien bottom in
const u32 bmp_alien_bottom_in_12x8[] = { packword12(0, 0, 0, 0, 1, 1,
	1, 1, 0, 0, 0, 0), packword12(0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0),
	packword12(1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1), packword12(1, 1, 1, 0,
		0, 1, 1, 1, 1, 1, 1, 1), packword12(1, 1, 1, 1, 1, 1, 1, 1, 1,
		1, 1, 1), packword12(0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0),
	packword12(0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0), packword12(0, 0, 1, 1,
		0, 0, 0, 0, 1, 1, 0, 0) };

//Alien bottom out
const u32 bmp_alien_bottom_out_12x8[] = { packword12(0, 0, 0, 0, 1, 1,
	1, 1, 0, 0, 0, 0), packword12(0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0),
	packword12(1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1), packword12(1, 1, 1, 0,
		0, 1, 1, 1, 1, 1, 1, 1), packword12(1, 1, 1, 1, 1, 1, 1, 1, 1,
		1, 1, 1), packword12(0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0),
	packword12(0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 0), packword12(1, 1, 0, 0,
		0, 0, 0, 0, 0, 0, 1, 1) };

//Empty alien
const u32 bmp_alien_empty[8] = { 0 };

//Lookup tables for easy bitmap selection
const u32* bmp_aliens_out[] = { bmp_alien_top_out_12x8,
	bmp_alien_middle_out_12x8, bmp_alien_middle_out_12x8,
	bmp_alien_bottom_out_12x8, bmp_alien_bottom_out_12x8 };
const u32* bmp_aliens_in[] = { bmp_alien_top_in_12x8, bmp_alien_middle_in_12x8,
	bmp_alien_middle_in_12x8, bmp_alien_bottom_in_12x8,
	bmp_alien_bottom_in_12x8 };

const u32** bmp_aliens[] = { bmp_aliens_out, bmp_aliens_in };

//Tank bitmap
const u32 bmp_tank_15x8[] = {
	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) };

const u32 bmp_tank_explode1_15x8[] = {
	packword15(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0),
	packword15(0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0),
	packword15(0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0),
```

```c
        packword15(0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0),
        packword15(0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0),
        packword15(0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0),
        packword15(0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0),
        packword15(0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0) };

const u32 bmp_tank_explode2_15x8[] = {
        packword15(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0),
        packword15(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0),
        packword15(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0),
        packword15(0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0),
        packword15(0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0),
        packword15(0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0),
        packword15(0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0),
        packword15(0, 0, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 0) };

const u32 bmp_tank_empty[] = {
        packword15(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0),
        packword15(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0),
        packword15(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0),
        packword15(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0),
        packword15(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0),
        packword15(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0),
        packword15(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0),
        packword15(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0) };

const u32* bmp_tanks[] = { bmp_tank_empty, bmp_tank_15x8,
        bmp_tank_explode1_15x8, bmp_tank_explode2_15x8 };

// Shape of the entire bunker.
const u32
        bmp_bunker_24x18[] =
                {
                        packword24(0, 0, 0, 1, 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, 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, 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, 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, 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, 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, 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, 1),
                        packword24(1, 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, 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, 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, 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, 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, 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, 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, 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, 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, 1) };

const u32 bmp_bunker0_6x6[] = { packword6(0,0,0,1,1,1), packword6(0,0,1,1,1,1),
        packword6(0,1,1,1,1,1), packword6(1,1,1,1,1,1), packword6(1,1,1,1,1,1),
        packword6(1,1,1,1,1,1) };
const u32 bmp_bunker1_6x6[] = { 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) };
const u32 bmp_bunker2_6x6[] = { packword6(1,1,1,0,0,0), packword6(1,1,1,1,0,0),
        packword6(1,1,1,1,1,0), packword6(1,1,1,1,1,1), packword6(1,1,1,1,1,1),
        packword6(1,1,1,1,1,1) };
const u32 bmp_bunker3_6x6[] = { packword6(1,1,1,1,1,1), packword6(1,1,1,1,0,0),
        packword6(1,1,1,0,0,0), packword6(1,1,0,0,0,0), packword6(1,0,0,0,0,0),
        packword6(0,0,0,0,0,0) };
const u32 bmp_bunker4_6x6[] = { packword6(0,0,0,0,0,0), packword6(0,0,0,0,0,0),
        packword6(0,0,0,0,0,0), packword6(0,0,0,0,0,0), packword6(0,0,0,0,0,0),
        packword6(0,0,0,0,0,0) };
const u32 bmp_bunker5_6x6[] = { packword6(1,1,1,1,1,1), packword6(0,0,1,1,1,1),
        packword6(0,0,0,1,1,1), packword6(0,0,0,0,1,1), packword6(0,0,0,0,0,1),
        packword6(0,0,0,0,0,0) };

const u32* bmp_bunker_blocks[] = { bmp_bunker2_6x6, bmp_bunker1_6x6,
        bmp_bunker1_6x6, bmp_bunker0_6x6, bmp_bunker1_6x6, bmp_bunker5_6x6,
        bmp_bunker3_6x6, bmp_bunker1_6x6, bmp_bunker1_6x6, bmp_bunker4_6x6,
        bmp_bunker4_6x6, bmp_bunker1_6x6 };
```

```
// These are the blocks that comprise the bunker and each time a bullet
// strikes one of these blocks, you erode the block as you sequence through
// these patterns.
const u32 bmp_bunkerDamage0_6x6[] = { packword6(0,0,0,0,0,0),
        packword6(0,0,0,0,0,0), packword6(0,0,0,0,0,0), packword6(0,0,0,0,0,0),
        packword6(0,0,0,0,0,0), packword6(0,0,0,0,0,0) };
const u32 bmp_bunkerDamage1_6x6[] = { 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) };
const u32 bmp_bunkerDamage2_6x6[] = { 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) };
const u32 bmp_bunkerDamage3_6x6[] = { 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) };
const u32 bmp_bunkerDamage4_6x6[] = { 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) };
const u32* bmp_bunker_damages[] = { bmp_bunkerDamage0_6x6,
        bmp_bunkerDamage1_6x6, bmp_bunkerDamage2_6x6, bmp_bunkerDamage3_6x6,
        bmp_bunkerDamage4_6x6 };

//Missile bitmaps
const u32
        bmp_alien_missile_cross2_3x5[] =
                { packword3(0, 1, 0), packword3(0, 1,
                        0), packword3(0, 1, 0), packword3(1, 1, 1),
                        packword3(0, 1, 0) };
const u32
        bmp_alien_missile_cross1_3x5[] =
                { packword3(0, 1, 0), packword3(1, 1,
                        1), packword3(0, 1, 0), packword3(0, 1, 0),
                        packword3(0, 1, 0) };
const u32
        bmp_alien_missile_diagonal1_3x5[] = { packword3(1, 0, 0),
                packword3(0, 1,
                        0), packword3(0, 0, 1), packword3(0, 1, 0),
                packword3(1, 0, 0) };
const u32
        bmp_alien_missile_diagonal2_3x5[] = { packword3(0, 0, 1),
                packword3(0, 1,
                        0), packword3(1, 0, 0), packword3(0, 1, 0),
                packword3(0, 0, 1) };
const u32* bmp_alien_missiles_cross[] = { bmp_alien_missile_cross2_3x5,
        bmp_alien_missile_cross1_3x5 };
const u32* bmp_alien_missiles_diagonal[] = { bmp_alien_missile_diagonal1_3x5,
        bmp_alien_missile_diagonal2_3x5 };
const u32** bmp_alien_missiles[] = { bmp_alien_missiles_cross,
        bmp_alien_missiles_diagonal };

const u32 bmp_bullet_straight_3x5[] = { packword3(0, 1, 0), packword3(0, 1,
        0), packword3(0, 1, 0), packword3(0, 1, 0), packword3(0, 1, 0),
        packword3(0, 1, 0) };

const u32 bmp_empty_projectile[] = { packword3(0, 0, 0), packword3(0, 0, 0),
        packword3(0, 0, 0), packword3(0, 0, 0), packword3(0, 0, 0) };

const u32 number_one_4x7[] = { packword4(0,0,1,0), packword4(0,0,1,1),
        packword4(0,0,1,0), packword4(0,0,1,0), packword4(0,0,1,0),
        packword4(0,0,1,0), packword4(0,1,1,1) };
const u32 number_two_4x7[] = { packword4(0,1,1,0), packword4(1,0,0,1),
        packword4(1,0,0,0), packword4(1,1,1,0), packword4(0,0,0,1),
        packword4(0,0,0,1), packword4(1,1,1,1) };
const u32 number_three_4x7[] = { packword4(0,1,1,0), packword4(1,0,0,1),
        packword4(1,0,0,0), packword4(1,1,1,0), packword4(1,0,0,0),
        packword4(1,0,0,1), packword4(0,1,1,0) };
const u32 number_four_4x7[] = { packword4(1,0,0,1), packword4(1,0,0,1),
        packword4(1,0,0,1), packword4(1,1,1,1), packword4(1,0,0,0),
        packword4(1,0,0,0), packword4(1,0,0,0) };
const u32 number_five_4x7[] = { packword4(1,1,1,1), packword4(0,0,0,1),
```

```
        packword4(0,0,0,1), packword4(0,1,1,1), packword4(1,0,0,0),
        packword4(1,0,0,1), packword4(0,1,1,0) };
const u32 number_six_4x7[] = { packword4(0,1,1,0), packword4(0,0,0,1),
        packword4(0,0,0,1), packword4(0,1,1,1), packword4(1,0,0,1),
        packword4(1,0,0,1), packword4(0,1,1,0) };
const u32 number_seven_4x7[] = { packword4(1,1,1,1), packword4(1,0,0,0),
        packword4(1,0,0,0), packword4(1,0,0,0), packword4(1,0,0,0),
        packword4(1,0,0,0), packword4(1,0,0,0) };
const u32 number_eight_4x7[] = { packword4(0,1,1,0), packword4(1,0,0,1),
        packword4(1,0,0,1), packword4(0,1,1,0), packword4(1,0,0,1),
        packword4(1,0,0,1), packword4(0,1,1,0) };
const u32 number_nine_4x7[] = { packword4(0,1,1,0), packword4(1,0,0,1),
        packword4(1,0,0,1), packword4(1,1,1,1), packword4(1,0,0,0),
        packword4(1,0,0,0), packword4(0,1,1,0) };

const u32 number_zero_4x7[] = { packword4(0,1,1,0), packword4(1,0,0,1),
        packword4(1,0,0,1), packword4(1,0,0,1), packword4(1,0,0,1),
        packword4(1,0,0,1), packword4(0,1,1,0) };

#define NUMBERS 10
const u32* bmp_numbers[NUMBERS] = { number_zero_4x7, number_one_4x7,
        number_two_4x7, number_three_4x7, number_four_4x7, number_five_4x7,
        number_six_4x7, number_seven_4x7, number_eight_4x7, number_nine_4x7 };
// This is for the drawing of the alien saucer sprite.
const u32 bmp_saucer_16x7[] = { packword16(0,0,0,0,0,1,1,1,1,1,1,0,0,0,0,0),
        packword16(0,0,0,1,1,1,1,1,1,1,1,1,1,0,0,0),
        packword16(0,0,1,1,1,1,1,1,1,1,1,1,1,1,0,0),
        packword16(0,1,1,0,1,1,0,1,1,0,1,1,0,1,1,0),
        packword16(1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1),
        packword16(0,0,1,1,1,0,0,1,1,0,0,1,1,1,0,0),
        packword16(0,0,0,1,0,0,0,0,0,0,0,0,1,0,0,0) };

//This is for drawing the words
const u32 bmp_word_lives_27x8[] = {
        packword27(0,1,1,1,0,0,1,1,1,1,0,1,0,0,0,0,0,1,0,1,0,0,0,0,1,1,0,0),
        packword27(0,0,0,0,1,0,0,0,0,1,0,1,0,0,0,0,0,1,0,1,0,0,0,0,1,1,0,0),
        packword27(0,0,0,0,1,0,0,0,0,1,0,1,0,0,0,0,0,1,0,1,0,0,0,0,1,1,0,0),
        packword27(0,0,1,1,0,0,0,1,1,1,0,1,0,0,0,0,0,1,0,1,0,0,0,0,1,1,0,0),
        packword27(0,1,0,0,0,0,0,0,0,1,0,0,1,0,0,0,1,0,0,1,0,0,0,0,1,1,0,0),
        packword27(0,1,0,0,0,0,0,0,0,1,0,0,0,1,0,1,0,0,0,1,0,0,0,0,1,1,0,0),
        packword27(0,0,1,1,1,0,1,1,1,1,0,0,0,0,1,0,0,0,0,1,0,1,1,1,1,1,0,0),
        packword27(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0) };

const u32 bmp_word_score_27x8[] = {
        packword27(1,1,1,1,0,0,1,1,1,0,1,1,1,1,0,1,1,1,0,0,1,1,1,0,0,0,0,0),
        packword27(0,0,0,1,0,1,0,0,1,0,1,0,0,1,0,0,0,0,0,1,0,0,0,0,1,0,0,0),
        packword27(0,0,0,1,0,1,0,0,1,0,1,0,0,1,0,0,0,0,0,1,0,0,0,0,1,0,0,0),
        packword27(0,1,1,1,0,0,1,1,1,0,1,0,0,1,0,0,0,0,0,1,0,0,1,1,0,0,0,0),
        packword27(0,0,0,1,0,1,0,0,1,0,1,0,0,1,0,0,0,0,0,1,0,1,0,0,0,0,0,0),
        packword27(0,0,0,1,0,1,0,0,1,0,1,0,0,1,0,0,0,0,0,1,0,1,0,0,0,0,0,0),
        packword27(1,1,1,1,0,1,0,0,1,0,1,1,1,1,0,1,1,1,0,0,0,1,1,1,0,0,0,0),
        packword27(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0) };

const u32 bmp_word_game_27x8[] = {
        packword27(1,1,1,1,0,1,0,0,0,0,0,1,0,0,1,1,1,0,0,0,1,1,1,1,0,0,0,0),
        packword27(0,0,0,1,0,1,1,0,0,0,1,1,0,1,0,0,0,1,0,0,0,0,0,0,1,0,0,0),
        packword27(0,0,0,1,0,1,0,1,0,1,0,1,0,1,0,0,0,1,0,0,0,0,0,0,1,0,0,0),
        packword27(0,1,1,1,0,1,0,0,1,0,0,1,0,1,1,1,1,1,0,1,1,1,1,0,1,0,0,0),
        packword27(0,0,0,1,0,1,0,0,0,0,0,1,0,1,0,0,0,1,0,1,0,0,0,0,1,0,0,0),
        packword27(0,0,0,1,0,1,0,0,0,0,0,1,0,1,0,0,0,1,0,1,0,0,0,0,1,0,0,0),
        packword27(1,1,1,1,0,1,0,0,0,0,0,1,0,1,0,0,0,1,0,0,1,1,1,1,0,0,0,0),
        packword27(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0) };

const u32 bmp_word_over_27x8[] = {
        packword27(0,1,1,1,0,1,1,1,1,0,1,0,0,0,0,0,1,0,1,1,1,1,1,0,0,0,0,0),
        packword27(1,0,0,1,0,0,0,0,1,0,1,0,0,0,0,0,1,0,1,0,0,0,1,0,0,0,0,0),
        packword27(1,0,0,1,0,0,0,0,1,0,1,0,0,0,0,0,1,0,1,0,0,0,1,0,0,0,0,0),
        packword27(0,1,1,1,0,0,1,1,1,0,1,0,0,0,0,0,1,0,1,0,0,0,1,0,0,0,0,0),
        packword27(1,0,0,1,0,0,0,0,1,0,0,1,0,0,0,1,0,0,1,0,0,0,1,0,0,0,0,0),
        packword27(1,0,0,1,0,0,0,0,1,0,0,0,1,0,1,0,0,0,1,0,0,0,1,0,0,0,0,0),
        packword27(1,0,0,1,0,1,1,1,1,0,0,0,0,1,0,0,0,0,1,1,1,1,1,0,0,0,0,0),
        packword27(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0) };

/*
 * control.c
```

```
 *
 *  Created on: Oct 4, 2017
 *      Author: superman
 */

#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include "control.h"
#include "render.h"
#include "table.h"
#include "gpio.h"
#include "timing.h"
#include "input.h"

#define check(byte,bit) (byte & (table_bit[bit])) //checks to see if there is overlap in the alien bit table
#define set(byte,bit) (byte |= (table_bit[bit])) //sets an alien in the bit table
#define clear(byte,bit) (byte &= ~(table_bit[bit])) //macro for eliminating an alien from the bit table
#define BULLET_POS -1 //initial bullet position
#define MOVE_SPRITE 2 //increment at which sprites move
#define TANK_MOVE_DIST 1
#define MISSILE_MOVE_DIST 4
#define BULLET_MOVE_DIST 2
#define OFFSET_MAX 10 //max offset for deleting columns
#define BIT_SHIFT 1

#define ALIEN_MID (BMP_ALIEN_W/2)       //midpoint of an alien
#define ALIEN_ROW_ALIVE 0x7ff
#define TANK_BULL_Y 7       //bullet y offset for tank
#define TANK_BULL_X (19/2)-1 //bullet x offset for tank
#define LIFE_COUNT 3 //initial tank lives
//
//Local typedefs


//Prototypes
void init_bunker_states(void); //function that sets the bunker states to max health
void update_tank_lives(void);

static bool collision_detect(point_t pos1, point_t dim1, point_t pos2,
        point_t dim2);
static bool collision_detect_bmp(const u32* bmp1, point_t pos1, point_t dim1,
        const u32* bmp2, point_t pos2, point_t dim2, const u32* bmp_mask);
static bool detect_bunker_collision(u16* bunker_num, u16* block_num,
        point_t projectile_pos, const u32* bmp);
static bool detect_alien_collision(u16* alien_row, u16* alien_col, point_t pos,
        point_t dim, const u32* bmp);
static bool detect_tank_collision(point_t pos, point_t dim, const u32* bmp);
static bool detect_saucer_collision(point_t pos, point_t dim, const u32* bmp);
void resurrect_tank();

//Const

//Variables
saucer_t saucer;
bunker_t bunkers[GAME_BUNKER_COUNT]; //array of 4 bunker states
alien_missiles_t alien_missiles[GAME_MISSILE_COUNT]; //array of 4 alien missiles
//initialize the tank with a position and a bullet
tank_t tank;
//initialize the alien block with a position and flags
alien_block_t alien_block;
u32 game_score = 0;

//starts up the game and initializes the key components
void control_init(void) {

    //initialize the bunker states to full health
    init_bunker_states();

    s16 i; //initialize the alien life/death array
    for (i = 0; i < GAME_ALIEN_ROWS; i++) {
        alien_block.alien_status[i] = ALIEN_ROW_ALIVE; //set all aliens to alive
    }
    for (i = 0; i < GAME_MISSILE_COUNT; i++) {
```

```
        alien_missiles[i].active = 0;
    }
    alien_block.legs = OUT;
    alien_block.loffset = 0;
    alien_block.roffset = 0;
    alien_block.row_col.x = GAME_INIT_ROW_COL;
    alien_block.row_col.y = GAME_INIT_ROW_COL;
    alien_block.pos.x = GAME_ALIEN_STARTX;
    alien_block.pos.y = GAME_ALIEN_STARTY;
    alien_block.alive = 1;

    saucer.pos.x = GAME_SAUCER_STARTX;
    saucer.pos.y = GAME_SAUCER_Y;
    saucer.active = 0;
    saucer.alive = 0;

    tank.changed = 1;
    tank.lives = GAME_LIFE_COUNT;
    tank.pos.x = GAME_TANK_STARTX;
    tank.pos.y = GAME_TANK_STARTY;
    tank.state = ALIVE;
    tank.missile.active = 0;
    game_score = 0;

    //render(&tank, &alien_block, &alien_missiles, bunkers, &saucer, game_score); //render the sprites
}

#define ALIEN_DEATH_COUNT 20
//function that blocks on the user input and goes to correct function handler
void control_run(void) {
    static u16 alien_death_count = ALIEN_DEATH_COUNT;
    if (alien_block.row_col.x != GAME_INIT_ROW_COL && alien_block.row_col.y
            != GAME_INIT_ROW_COL && !(--alien_death_count)) {
        control_kill_alien(alien_block.row_col.y, alien_block.row_col.x);
        alien_block.row_col.x = GAME_INIT_ROW_COL;
        alien_block.row_col.y = GAME_INIT_ROW_COL;
        //alien_block.changed = 1;
        alien_death_count = ALIEN_DEATH_COUNT;
    }
    input_tank_controls();
    render(&tank, &alien_block, &alien_missiles, bunkers, &saucer, game_score); //render after button press
}

#define EMPTY_BLOCK1 9
#define EMPTY_BLOCK2 10

//function that sets the bunker states to max health
void init_bunker_states(void) {
    u8 i; //iterates through all the bunkers
    u8 j;
    for (i = 0; i < GAME_BUNKER_COUNT; i++) {
        bunkers[i].alive = 1; //set state to max
        bunkers[i].changed = 1;
        for (j = 0; j < GAME_BUNKER_BLOCK_COUNT; j++) {
            if (j == EMPTY_BLOCK1 || j == EMPTY_BLOCK2)
                continue;
            bunkers[i].block[j].block_health = GAME_BUNKER_MAX;
            bunkers[i].block[j].changed = 1;
        }
    }
}

//function that moves the tank
void control_tank_move(direction_t dir) {
    if (tank.state != ALIVE)
        return;
    //if the tank is moving left
    if (dir == LEFT) {
        //check to see if it has hit the edge
        if (tank.pos.x >= TANK_MOVE_DIST) {
            //update the tank position to the left
            tank.pos.x -= TANK_MOVE_DIST;
            tank.changed = 1;
        }
    } else if (dir == RIGHT) { //check to see if the tank is moving right
```

```
        if (tank.pos.x <= GAME_W - BMP_TANK_W - TANK_MOVE_DIST) { //check to see if tank is moving off edge
            tank.pos.x += TANK_MOVE_DIST; // move to the right
            tank.changed = 1;
        }
    }
}

void reset_tank() {
    if (tank.lives) {
        tank.pos.x = GAME_TANK_STARTX;
        tank.pos.y = GAME_TANK_STARTY;
        tank.state = ALIVE;
        tank.missile.active = 0;
    } else {
        tank.state = EMPTY;
        timing_set_gameover();
    }

}

#define EXPLODE1_TIME 5
#define EXPLODE2_TIME 5
#define NEW_LIFE_WAIT 2
void control_tank_explode() {
    static u16 counter = 0;
    static u16 new_life_timer = 0;
    if (!counter) {
        if (tank.state == EXPLODE1) {
            counter = EXPLODE1_TIME;
        } else if (tank.state == EXPLODE2) {
            counter = EXPLODE2_TIME;
        } else {
            counter = 0;
        }
        return;
    }

    if (!(--counter)) {
        if (tank.state == EXPLODE1) {
            tank.state = EXPLODE2;
            tank.changed = 1;
            if (new_life_timer == 0) {
                new_life_timer = NEW_LIFE_WAIT;
            } else if (!(--new_life_timer)) {
                counter = 0;
                tank.changed = 1;
                reset_tank();
            }
        } else if (tank.state == EXPLODE2) {
            tank.state = EXPLODE1;
            tank.changed = 1;
        }
    }
}

#define RIGHT_WALL (GAME_W - GAME_ALIEN_COLS*GAME_ALIEN_SEPX)
//function that updates the alien block position
void control_update_alien_position(void) {
    static direction_t dir = GAME_ALIEN_DIR;

    if (dir == LEFT) { //if aliens are moving left
        if (alien_block.pos.x + alien_block.loffset >= MOVE_SPRITE) { //and they haven't run off the screen
            alien_block.pos.x -= MOVE_SPRITE; //move the block by 2 pixels
        } else { //otherwise
            alien_block.pos.y += MOVE_SPRITE; //move the block down by 2 pixels
            dir = RIGHT; //and change the direction to right
        }

    } else if (dir == RIGHT) { //if aliens are moving right
        if (alien_block.pos.x - alien_block.roffset < RIGHT_WALL) { //and they haven't run into the wall
            alien_block.pos.x += MOVE_SPRITE; //move the block by 2 pixels
        } else { //otherwise
            alien_block.pos.y += MOVE_SPRITE; //move the block down by 2 pixels
            dir = LEFT; // and change the direction to the left
        }
```

```
    }
    if (alien_block.legs == OUT) //alternate between the aliens with legs out
        alien_block.legs = IN; //and legs in
    else
        //if the legs are already in then shift back to out
        alien_block.legs = OUT;
    alien_block.changed = 1;
}

//function that queries the user for an alien to kill, then wipes it out
void control_kill_alien(u16 alien_row, u16 alien_col) {
    u8 i = 0; //keep track of length of input

    //remove the alien from the alien bit table
    clear(alien_block.alien_status[alien_row],alien_col);

    //takes all the row status bits and ORs them together to find offset
    u16 mask = 0;
    for (i = 0; i < GAME_ALIEN_ROWS; i++) {
        mask |= alien_block.alien_status[i]; //OR together the statuses
    }

    alien_block.changed = 1;

    if (mask == 0) {
        alien_block.alive = 0;
        alien_block.loffset = -1;
        alien_block.roffset = -1;
        timing_set_win();
        tank.lives++;
        return;
    }

    u16 maskp = mask; //init the mask
    u8 n = OFFSET_MAX; //init the max offset

    while (maskp >>= BIT_SHIFT) { //finding the right offset
        n--;
    }

    i = OFFSET_MAX; //init the max offset
    maskp = mask;
    while ((maskp = ((maskp << BIT_SHIFT) & ALIEN_ROW_ALIVE))) {//find the left offset
        i--;
    }

    alien_block.loffset = i * GAME_ALIEN_SEPX; //calculate the left offset
    alien_block.roffset = n * GAME_ALIEN_SEPX; //calculate the right offset

    alien_block.changed = 1;
}
//function that fires the tank bullet
void control_tank_fire(void) {
    if (tank.state == EMPTY)
        timing_restart_game();
    if (tank.state != ALIVE)
        return;
    static u8 fired = 0;
    //check to see if the tank bullet is already active
    if (fired == 0) {
        //place the tank bullet at the correct x position
        tank.missile.pos.x = tank.pos.x + TANK_BULL_X;
        //place the bullet at the top of the tank
        tank.missile.pos.y = tank.pos.y - TANK_BULL_Y;
        //set the bullet to active
        tank.missile.active = 1;
        //kill_saucer();
        //update_tank_lives();
        fired = 1;
    } else {
        if (tank.missile.active == 0)
            fired = 0;
    }

}
```

```
//function that fires the alien missiles
void control_alien_fire_missile(void) {
    u8 i;
    //used to identify the column of the shooter
    u8 shooter_col;
    //iterate over the missiles to check if they are active
    if (!alien_block.alive)
        return;
    for (i = 0; i < CONTROL_MISSILES; i++) {
        //if the next alien missile is not active
        if (!alien_missiles[i].active) {
            //activate it
            alien_missiles[i].active = 1;
            //stop checking the missiles
            break;
        }
    }
    //if all missiles were checked and all were active
    if (i == CONTROL_MISSILES) {
        //exit the function
        return;
    }
    //flag to check if a missile was fired
    u8 fired = 0;
    //track the shooter's row
    s16 shooter_row;
    //until a missile has been fired
    while (!fired) {
        shooter_row = GAME_ALIEN_ROWS - 1;
        //pick a random column to shoot
        shooter_col = rand() % GAME_ALIEN_COLS;
        //check to see if the shooter is in the bottom row
        while ((shooter_row >= 0)
                && !check(alien_block.alien_status[shooter_row],shooter_col)) {
            shooter_row--;
        }

        //if the shooter was in the bottom row
        if (shooter_row >= 0) {
            //show that a missile was fired
            fired = 1;
        }
    }
    //update the alien missile x position
    alien_missiles[i].pos.x = alien_block.pos.x + shooter_col * BMP_ALIEN_W
            + ALIEN_MID;
    //update the alien missile y position
    alien_missiles[i].pos.y = alien_block.pos.y + (shooter_row + 1)
            * GAME_ALIEN_SEPY - ALIEN_MID;

    //about once every 4 shots shoot a strong alien missile
    if (rand() % CONTROL_MISSILES == 0)
        //set the alien missile type to strong
        alien_missiles[i].type = STRONG;
    else
        //else set the alien missile type to normal
        alien_missiles[i].type = NORMAL;
    alien_missiles[i].guise = GUISE_0;
}

#define RES_SCALE 2
#define OFFLIMITS 10
#define MIN_SCORE 50
#define INCREMENT 9
#define POINTS_ROW0 40
#define POINTS_ROW1 20
#define POINTS_ROW2 20
#define POINTS_ROW3 10
#define POINTS_ROW4 10
//function that updates the position of all bullets, both alien and tank
void control_update_bullet(void) {
    static const u16 point_values[GAME_ALIEN_ROWS] = { POINTS_ROW0,
            POINTS_ROW1, POINTS_ROW2, POINTS_ROW3, POINTS_ROW4 };
    //if there is already an active tank bullet
    if (tank.missile.active) {
```

```
        //update its y position by 2 pixels
        tank.missile.pos.y = tank.missile.pos.y - BULLET_MOVE_DIST;
        //if the tank bullet leaves the screen
        if (tank.missile.pos.y < OFFLIMITS) {
            //allow for another active bullet
            tank.missile.active = 0;
            return;
        }
        u16 bunker_num;
        u16 block_num;
        if (detect_bunker_collision(&bunker_num, &block_num, tank.missile.pos,
                bmp_bullet_straight_3x5)) {

            tank.missile.active = 0;
            bunkers[bunker_num].block[block_num].block_health--;
            bunkers[bunker_num].block[block_num].changed = 1;
            bunkers[bunker_num].changed = 1;
        }

        u16 alien_row;
        u16 alien_col;
        if (detect_alien_collision(&alien_row, &alien_col, tank.missile.pos,
                bmp_missile_dim, bmp_bullet_straight_3x5)) {
            alien_block.row_col.x = alien_col;
            alien_block.row_col.y = alien_row;
            tank.missile.active = 0;
            game_score += point_values[alien_row];
        }

        if (detect_saucer_collision(tank.missile.pos, bmp_missile_dim,
                bmp_bullet_straight_3x5)) {
            tank.missile.active = 0;

            game_score += saucer.points;
            saucer.alive = 0;
        }

    }
}

void control_update_missiles(void) {
    //iterate through all the alien missiles
    u8 i;
    for (i = 0; i < CONTROL_MISSILES; i++) {
        //check whether the alien missile is active
        if (alien_missiles[i].active) {
            //move the alien missile position up by 2 pixels
            alien_missiles[i].pos.y = alien_missiles[i].pos.y
                    + MISSILE_MOVE_DIST;
            //check the guise of the alien missile
            if (alien_missiles[i].guise == GUISE_1)
                //alternate the guise every movement
                alien_missiles[i].guise = GUISE_0;
            else
                //change the guise
                alien_missiles[i].guise = GUISE_1;
            //if the alien missile exits the screen, allow it to be fired again
            if (alien_missiles[i].pos.y >= 220) {
                //allow for another alien missile
                alien_missiles[i].active = 0;
            }
            u16 bunker_num;
            u16 block_num;
            if (detect_bunker_collision(
                    &bunker_num,
                    &block_num,
                    alien_missiles[i].pos,
                    bmp_alien_missiles[alien_missiles[i].type][alien_missiles[i].guise])) {

                alien_missiles[i].active = 0;
                bunkers[bunker_num].block[block_num].block_health--;
                bunkers[bunker_num].block[block_num].changed = 1;
                bunkers[bunker_num].changed = 1;
            } else if (detect_tank_collision(
                    alien_missiles[i].pos,
```

```
                    bmp_missile_dim,
                    bmp_alien_missiles[alien_missiles[i].type][alien_missiles[i].guise])) {
                alien_missiles[i].active = 0;
                update_tank_lives();

            }
        }
    }
}

#define SAUCER_WIDTH 20
#define NEG_SAUCER_WIDTH -21 //saucer can go off screen
#define SAUCER_PAUSE 100

//
void control_saucer_move(void) {
    static direction_t saucer_dir = LEFT; //start the saucer going left
    static u16 saucer_pause = 0;

    //  xil_printf("saucer %d,%d active: %d alive: %d\n\r", saucer.pos.x,
    //          saucer.pos.y, saucer.active, saucer.alive);

    if (saucer_pause) {
        --saucer_pause;
        saucer.alive = 1;
        return;
    }
    if (saucer.alive == 0) {
        saucer_pause = SAUCER_PAUSE;
        saucer.points = 0;
        saucer_dir = LEFT;
        saucer.pos.x = GAME_SAUCER_STARTX;
        saucer.pos.y = GAME_SAUCER_Y;
        saucer.points = ((rand() % INCREMENT) * MIN_SCORE + MIN_SCORE);
        return;
    }

    if (saucer_dir == LEFT) { //if the saucer is moving left
        if (saucer.pos.x > NEG_SAUCER_WIDTH * 2) //move as long as not off screen
            saucer.pos.x -= MOVE_SPRITE; //decrement x position
        else {
            saucer_dir = RIGHT; //change saucer direction
        }
    } else if (saucer_dir == RIGHT) { //if the saucer is moving right
        if (saucer.pos.x < GAME_W + SAUCER_WIDTH * 2) //move until off the screen
            saucer.pos.x += MOVE_SPRITE; //increment position
        else {
            saucer_dir = LEFT; //change saucer direction
        }
    }
}

void update_tank_lives(void) {

    if (tank.state == ALIVE) {
        tank.state = EXPLODE1;
        tank.changed = 1;
        tank.lives--;
    }

}

static bool detect_saucer_collision(point_t pos, point_t dim, const u32* bmp) {
    if (collision_detect_bmp(bmp, pos, dim, bmp_saucer_16x7, saucer.pos,
            bmp_saucer_dim, 0))
        return true;
    return false;
    //
}

//Returns whether the given bitmap with position pos and dimensions dim intersects with the tank
static bool detect_tank_collision(point_t pos, point_t dim, const u32* bmp) {
    if (collision_detect_bmp(bmp, pos, dim, bmp_tank_15x8, tank.pos,
            bmp_tank_dim, 0)) {
        return true;
```

```
    }
    return false;
    //
}

//Returns whether or not the given bitmap with position pos and dimensions dim intersects with an alien
//  if true, stores in alien_row and alien_col the position of the intersected alien
static bool detect_alien_collision(u16* alien_row, u16* alien_col, point_t pos,
        point_t dim, const u32* bmp) {
    point_t alien_dim;
    alien_dim.x = GAME_ALIEN_COLS * GAME_ALIEN_SEPX;
    alien_dim.y = GAME_ALIEN_ROWS * GAME_ALIEN_SEPY;
    //First check if there is a collision with the whole block of aliens
    //  for efficiency
    if (!collision_detect(pos, dim, alien_block.pos, alien_dim))
        return false;

    //Calculate which alien is intersected using relative positions
    //  TODO fix for overlapping 2 aliens?
    point_t rel_pos;
    rel_pos.x = abs(pos.x - alien_block.pos.x);
    rel_pos.y = abs(pos.y - alien_block.pos.y);
    u16 col = rel_pos.x / GAME_ALIEN_SEPX;
    u16 row = rel_pos.y / GAME_ALIEN_SEPY;
    if (!check(alien_block.alien_status[row],col))
        return false;
    point_t alien_pos;
    alien_pos.x = alien_block.pos.x + GAME_ALIEN_SEPX * col;
    alien_pos.y = alien_block.pos.y + GAME_ALIEN_SEPY * row;

    alien_dim.x = BMP_ALIEN_W;
    alien_dim.y = BMP_ALIEN_H;
    if (collision_detect_bmp(bmp, pos, dim, bmp_aliens[alien_block.legs][row],
            alien_pos, alien_dim, 0)) {

        *alien_row = row;
        *alien_col = col;
        return true;
    }
    return false;
}

//Returns true if bitmap with given position and dimensions intersects with a bunker. If true,
//  returns in bunker_num and block_num which bunker and which block was intersected
static bool detect_bunker_collision(u16* bunker_num, u16* block_num,
        point_t projectile_pos, const u32* bmp) {

    point_t bunker_pos;
    bunker_pos.x = GAME_BUNKER_POS;
    bunker_pos.y = GAME_BUNKER_Y;
    point_t block_pos;
    //Iterate through bunkers, check for collision with each
    for (u16 i = 0; i < GAME_BUNKER_COUNT; i++) {
        //If collision with bunker detected
        if (collision_detect(projectile_pos, bmp_missile_dim, bunker_pos,
                bmp_bunker_dim)) {
            //Iterate through blocks in bunker; once collision has been detected, return
            for (u16 j = 0; j < GAME_BUNKER_BLOCK_COUNT; j++) {
                //If block is already dead, dont check
                if (bunkers[i].block[j].block_health == 0)
                    continue;

                block_pos.x = bunker_pos.x + BMP_BUNKER_BLOCK_W * (j
                        % GAME_BUNKER_WIDTH);
                block_pos.y = bunker_pos.y + BMP_BUNKER_BLOCK_H * (j
                        / GAME_BUNKER_WIDTH);
                if (collision_detect_bmp(bmp, projectile_pos, bmp_missile_dim,
                        bmp_bunker_blocks[j], block_pos, bmp_bunker_block_dim,
                        bmp_bunker_damages[bunkers[i].block[j].block_health])) {
                    *block_num = j;
                    *bunker_num = i;
                    return true;
                }
            }
```

```
        }
        //Next bunker position x
        bunker_pos.x += GAME_BUNKER_SEP;
    }

    return false;
}

//Detect if two bitmap hitboxes overlap
static bool collision_detect(point_t pos1, point_t dim1, point_t pos2,
        point_t dim2) {

    s16 top1 = pos1.y;
    s16 bottom2 = pos2.y + dim2.y;
    if (top1 > bottom2) //first is below second
        return false;
    s16 bottom1 = pos1.y + dim1.y;
    s16 top2 = pos2.y;
    if (top2 > bottom1) //first is above second
        return false;
    s16 left1 = pos1.x;
    s16 right2 = pos2.x + dim2.x;
    if (left1 > right2) //first is right of second
        return false;
    s16 right1 = pos1.x + dim1.x;
    s16 left2 = pos2.x;
    if (left2 > right1) //first is left of second
        return false;
    return true;
}

//Overlays two overlapping bitaps and checks if their set bits overlap. If so, returns true.
//  Provides pixel-resolution collision detection
static bool collision_detect_bmp(const u32* bmp1, point_t pos1, point_t dim1,
        const u32* bmp2, point_t pos2, point_t dim2, const u32* bmp_mask) {
    if (!collision_detect(pos1, dim1, pos2, dim2))
        return false;

    u16 shift1 = 0;
    u16 shift2 = 0;
    u16 y1 = 0;
    u16 y2 = 0;
    u16 count;
    u32 bmp_row1;
    u32 bmp_row2;

    if (pos1.x < pos2.x) {
        shift1 = pos2.x - pos1.x;
    } else {
        shift2 = pos1.x - pos2.x;
    }
    if (pos1.y < pos2.y) {
        y1 = pos2.y - pos1.y;
        count = dim1.y - y1;
    } else {
        y2 = pos1.y - pos2.y;
        count = dim2.y - y2;
    }

    while (count-- > 0) {
        bmp_row1 = bmp1[y1];
        bmp_row2 = bmp2[y2];
        if (bmp_mask)
            bmp_row2 &= bmp_mask[y2];
        if ((bmp_row1 >> shift1) & (bmp_row2 >> shift2))
            return true;
    }
    return false;
}

/*
 * input.c
 *
 *  Created on: Oct 20, 2017
 *      Author: superman
```

```
 */

#include "xil_types.h"
#include "input.h"
#include "gpio.h"
#include "control.h"

#define LEFT_BTN 0x08   //bit mask for left button
#define SHOOT_BTN 0x01  //bit mask for shoot button
#define RIGHT_BTN 0x02  //bit mask for right button
//state machine for tank movement and shooting
void input_tank_controls(void) {

    //Check if any butotn is pressed. Do appropriate action
    if (button_state & SHOOT_BTN) {
        control_tank_fire();
    }
    if (button_state & LEFT_BTN) {
        control_tank_move(LEFT);
    } else if (button_state & RIGHT_BTN) {
        control_tank_move(RIGHT);
    }
}
/*
 * render.c
 *
 * Handles screen drawing and updates
 *
 *  Created on: Sep 28, 2017
 *      Author:  Broderick Gardner
 *               Benjamin Gardner
 */

#include <stdlib.h>
#include <stdio.h>
#include "string.h"
#include <stdbool.h>

#include "render.h"
#include "xparameters.h"
#include "xio.h"
#include "xil_types.h"

#include "vdma.h"
#include "bmp.h"
#include "table.h"

#define BIT0 0x01

#define COLOR_GREEN 0x0000FF00
#define COLOR_BLACK 0x00000000
#define COLOR_WHITE 0x00FFFFFF
#define COLOR_RED   0xFFF00000
#define BULLET_COLOR COLOR_WHITE
#define BUNKER_ROWS 3   //Bunker dimensions in blocks
#define BUNKER_COLS 4

#define SAUCER_WIDTH 16
#define SAUCER_HEIGHT 7
#define SAUCER_STARTX 321 //x position of saucer starts off the screen
#define SAUCER_STARTY 15 //y position of saucer
#define SCORE_X 20 //x pos of score
#define LIVES_X 200 //x pos of lives
#define TANK_SPACE 20 //gap between tanks
#define TOP_SCREEN 1 //y pos of all top of screen sprites
#define SCORE_GAP_1 6
#define SCORE_GAP_2 11
#define SCORE_GAP_3 16
#define SCORE_GAP_4 21

#define WORDS_W 25 //word sprite width
#define WORDS_H 8 //word sprite height
#define FRAME_BUFFER_0_ADDR 0xC1000000  // Starting location in DDR where we will store the images that we display.
static u32* frame0 = (u32*) FRAME_BUFFER_0_ADDR;
static u32* frame1 = ((u32*) FRAME_BUFFER_0_ADDR) + GAME_SCREEN_H
```

```
        * GAME_SCREEN_W;//Maybe use this later?
#define OOR -20000  //Out of range value. Guaranteed to be out of range of screen,
//#define DEBUG

//macros
#define min(x,y) (((x)<(y))?(x):(y))    //Not used anymore
#define max(x,y) (((x)>(y))?(x):(y))
#define check(byte,bit) (byte&(table_bit[bit]))

//Static function prototypes
static void draw_bitmap(const u32* bmp, u32 color, s16 bmp_x, s16 bmp_y,
        s16 bmp_w, s16 bmp_h);
static inline void set_point(s32 x, s32 y, u32 color);
static void drawGreenLine(void);
static void update_score(u32 score);
void update_bmp_row(s16 x, s16 y, u32 old, u32 new, u32 color);

//Macros
#define clr_point(x,y) set_point((x),(y),GAME_BACKGROUND)

//Static variables
static bunker_t prev_bunkers[GAME_BUNKER_COUNT];
static tank_t prev_tank;
static saucer_t prev_saucer;
static alien_block_t prev_block;
static alien_missiles_t prev_missiles[GAME_MISSILE_COUNT];

#define RESET_LIVES 0
static u8 prev_lives = RESET_LIVES;

u32* getFrame() {
    return frame0;
}
#define NUMS 4
u32 prev_numbers[NUMS];
void render_init() {
    vdma_init(frame0, frame1);
    // Just paint some large red, green, blue, and white squares in different
    // positions of the image for each frame in the buffer (framePointer0 and framePointer1).
    render_restart();

}

void render_restart() {
    int row = 0, col = 0;
    //Init screen to background color

    for (row = 0; row < GAME_SCREEN_H; row++) {
        for (col = 0; col < GAME_SCREEN_W; col++) {
            frame0[row * GAME_SCREEN_W + col] = GAME_BACKGROUND;
            frame1[row * GAME_SCREEN_W + col] = GAME_BACKGROUND;
        }
    }

    //Draw all bunkers
    u16 bunker_num;
    for (bunker_num = 0; bunker_num < GAME_BUNKER_COUNT; bunker_num++) {

        prev_bunkers[bunker_num].changed = 1;
        for (int block_num = 0; block_num < GAME_BUNKER_BLOCK_COUNT; block_num++) {
            prev_bunkers[bunker_num].block[block_num].changed = 1;
            prev_bunkers[bunker_num].block[block_num].block_health = 0;
        }
    }

    prev_saucer.active = 0;
    prev_saucer.pos.x = SAUCER_STARTX;
    prev_saucer.pos.y = SAUCER_STARTY;

    prev_tank.pos.x = GAME_TANK_STARTX;
    prev_tank.pos.y = GAME_TANK_STARTY;
    prev_tank.state = EMPTY;
    prev_lives = 0;

    prev_tank.missile.pos.x = OOR;
```

```
    prev_tank.missile.pos.y = OOR;
    prev_tank.missile.active = 0;

    //Draw Score
    draw_bitmap(bmp_word_score_27x8, COLOR_WHITE, SCORE_X, TOP_SCREEN, WORDS_W,
            WORDS_H);
    u32 offset = WORDS_W + SCORE_X;
    draw_bitmap(number_zero_4x7, COLOR_GREEN, offset + SCORE_GAP_1, TOP_SCREEN,
            BMP_NUMBER_W, BMP_NUMBER_H);
    draw_bitmap(number_zero_4x7, COLOR_GREEN, offset + SCORE_GAP_2, TOP_SCREEN,
            BMP_NUMBER_W, BMP_NUMBER_H);
    draw_bitmap(number_zero_4x7, COLOR_GREEN, offset + SCORE_GAP_3, TOP_SCREEN,
            BMP_NUMBER_W, BMP_NUMBER_H);
    draw_bitmap(number_zero_4x7, COLOR_GREEN, offset + SCORE_GAP_4, TOP_SCREEN,
            BMP_NUMBER_W, BMP_NUMBER_H);

    //Draw Live
    draw_bitmap(bmp_word_lives_27x8, COLOR_WHITE, LIVES_X, TOP_SCREEN, WORDS_W,
            WORDS_H);

    //Draw green line
    drawGreenLine();
}

#define GAME_X 130 //start x pos for game word
#define GAME_Y 120 //start y pos
#define OVER_X 160 //start x pos for over word
void render_gameover() {
    draw_bitmap(bmp_word_game_27x8, COLOR_WHITE, GAME_X, GAME_Y, WORDS_W,
            WORDS_H);
    draw_bitmap(bmp_word_over_27x8, COLOR_WHITE, OVER_X, GAME_Y, WORDS_W,
            WORDS_H);
}

#define NUMBER_1 1
#define NUMBER_2 2
#define NUMBER_3 3
#define THOUSAND 1000
#define HUNDRED 100
#define TEN 10
static void update_score(u32 score) {
    static u32 prev_score = 0;
    if (score == prev_score)
        return;

    u32 thousand = score / THOUSAND;
    u32 hundred = (score - thousand * THOUSAND) / HUNDRED;
    u32 ten = (score - thousand * THOUSAND - hundred * HUNDRED) / TEN;

    u32 offset = WORDS_W + SCORE_X;
    if (prev_numbers[NUMBER_3] != thousand) {
        draw_bitmap(bmp_numbers[prev_numbers[NUMBER_3]], COLOR_BLACK,
                offset + SCORE_GAP_1, TOP_SCREEN, BMP_NUMBER_W, BMP_NUMBER_H);
        draw_bitmap(bmp_numbers[thousand], COLOR_GREEN, offset + SCORE_GAP_1,
                TOP_SCREEN, BMP_NUMBER_W, BMP_NUMBER_H);
        prev_numbers[NUMBER_3] = thousand;
    }
    if (prev_numbers[NUMBER_2] != hundred) {
        draw_bitmap(bmp_numbers[prev_numbers[NUMBER_2]], COLOR_BLACK,
                offset + SCORE_GAP_2, TOP_SCREEN, BMP_NUMBER_W, BMP_NUMBER_H);
        draw_bitmap(bmp_numbers[hundred], COLOR_GREEN, offset + SCORE_GAP_2,
                TOP_SCREEN, BMP_NUMBER_W, BMP_NUMBER_H);
        prev_numbers[NUMBER_2] = hundred;
    }
    if (prev_numbers[NUMBER_1] != ten) {
        draw_bitmap(bmp_numbers[prev_numbers[NUMBER_1]], COLOR_BLACK,
                offset + SCORE_GAP_3, TOP_SCREEN, BMP_NUMBER_W, BMP_NUMBER_H);
        draw_bitmap(bmp_numbers[ten], COLOR_GREEN, offset + SCORE_GAP_3,
                TOP_SCREEN, BMP_NUMBER_W, BMP_NUMBER_H);
        prev_numbers[NUMBER_1] = ten;
    }
    prev_score = score;
}

#define ALIENS_START 0
```

```
#define ALIENS_END 10
#define ALIENS_ROW_COUNT 5
#define ALIENS_MAX_Y (ALIENS_SEPY* ALIENS_ROW_COUNT)
#define ALIENS_ROW_LEN 11
#define ALIEN_BLOCK_H (GAME_ALIEN_SEPY * GAME_ALIEN_ROWS)
#define ALIEN_BLOCK_W (GAME_ALIEN_SEPX * GAME_ALIEN_COLS)
#define ALIEN_X_SHIFT 2
#define ALIEN_COLOR 0x00FFFFFF
#define ALIENS_ROW0 0   //Refer to rows of aliens
#define ALIENS_ROW1 1
#define ALIENS_ROW2 2
#define ALIENS_ROW3 3
#define ALIENS_ROW4 4
#define ALIENS_ROW5 5   //This row only exists when shifting down
//Updates alien row when shifted, no more than 2. This limitation is from the
//  padding I could put on the bitmaps.  There is are 2 pixels on either side
//  of the bitmaps that are 0, so shifting further would cause broken sprites.
static void update_alien_row(alien_block_t* alien_block,
        alien_block_t* prev_block, const u32* prev_alien, const u32* new_alien,
        u16 row) {

    u16 y = alien_block->pos.y + row * GAME_ALIEN_SEPY;
    u16 alien_y;
    for (alien_y = 0; alien_y < BMP_ALIEN_H; alien_y++) {
        s16 offset = alien_block->pos.x - prev_block->pos.x; //shift distance. No more than 2
        u32 new = new_alien[alien_y]; //pixels that will be shifted
        u32 prev = prev_alien[alien_y]; //Pixels currently drawn on screen
        new = (offset > 0) ? (new << offset) : (new >> -offset);

        u16 x = prev_block->pos.x; //Holds x coord to draw at.
        u16 n; //Current alien in the row
        for (n = ALIENS_START; n <= ALIENS_END; n++, x += GAME_ALIEN_SEPX) {
            if (alien_block->row_col.x == n && alien_block->row_col.y == row) {
                u32 lnew = (offset > 0) ? (bmp_alien_explosion_12x10[alien_y]
                        << offset) : (bmp_alien_explosion_12x10[alien_y]
                        >> -offset);
                u32 lprev;
                if (prev_block->row_col.x == n && prev_block->row_col.y == row) {
                    lprev = bmp_alien_explosion_12x10[alien_y];
                } else {
                    lprev = prev;
                }
                update_bmp_row(x, y + alien_y, lprev, lnew, ALIEN_COLOR);
            } else if (check(alien_block->alien_status[row],n)) { //Only draw alien if not dead
                update_bmp_row(x, y + alien_y, prev, new, ALIEN_COLOR);
            } else if (check(prev_block->alien_status[row],n)) {
                update_bmp_row(x, y + alien_y,
                        bmp_alien_explosion_12x10[alien_y], 0, ALIEN_COLOR);
            }
        }

    }
}

//Important because coordinates can go negative. Just not this negative

//Update the whole alien block.  This shifts left or right, down, or deletes an alien. Only 1 at a time
static void update_alien_block(alien_block_t* alien_block) {
    //Init prev_block to something reasonable and detectable for first update
    static alien_block_t prev_block = { { OOR, OOR }, 0, 0, { 0 }, OUT };
    if (alien_block->changed == 0)
        return;
    bool first = false; //Makes sure an empty bitmap is passed as the previous block
    //So the the whole alien is drawn on the first update
    if (prev_block.pos.x == OOR) {
        prev_block.pos.x = alien_block->pos.x - 1;
        prev_block.pos.y = alien_block->pos.y;
        first = true;
    }

    if (abs(alien_block->pos.x - prev_block.pos.x) <= ALIEN_X_SHIFT
            && (alien_block->pos.y == prev_block.pos.y)) {

        //Call update on each row
        update_alien_row(
```

```
            alien_block,
            &prev_block,
            (first ? bmp_alien_empty
                    : bmp_aliens[prev_block.legs][ALIENS_ROW0]),
            bmp_aliens[alien_block->legs][ALIENS_ROW0], ALIENS_ROW0);
    update_alien_row(
            alien_block,
            &prev_block,
            (first ? bmp_alien_empty
                    : bmp_aliens[prev_block.legs][ALIENS_ROW1]),
            bmp_aliens[alien_block->legs][ALIENS_ROW1], ALIENS_ROW1);
    update_alien_row(
            alien_block,
            &prev_block,
            (first ? bmp_alien_empty
                    : bmp_aliens[prev_block.legs][ALIENS_ROW2]),
            bmp_aliens[alien_block->legs][ALIENS_ROW2], ALIENS_ROW2);
    update_alien_row(
            alien_block,
            &prev_block,
            (first ? bmp_alien_empty
                    : bmp_aliens[prev_block.legs][ALIENS_ROW3]),
            bmp_aliens[alien_block->legs][ALIENS_ROW3], ALIENS_ROW3);
    update_alien_row(
            alien_block,
            &prev_block,
            (first ? bmp_alien_empty
                    : bmp_aliens[prev_block.legs][ALIENS_ROW4]),
            bmp_aliens[alien_block->legs][ALIENS_ROW4], ALIENS_ROW4);

} //If y value is different. x values guaranteed to be the same
else if (alien_block->pos.y != prev_block.pos.y && (alien_block->pos.x
        == prev_block.pos.x)) {

    s16 y = prev_block.pos.y;
    s16 dy = GAME_ALIEN_DROP;
    u16 x = prev_block.pos.x;

    u16 alien_row;
    u16 rowy = 0;
    for (alien_row = 0; alien_row < GAME_ALIEN_ROWS; alien_row++, rowy
            +=GAME_ALIEN_SEPY) {

        const u32* old_bmp = bmp_aliens[prev_block.legs][alien_row];
        const u32* new_bmp = bmp_aliens[alien_block->legs][alien_row];
        s16 row;
        s16 rowp;
        for (row = 0, rowp = -dy; row < BMP_ALIEN_H + dy; row++, rowp++) {
            u32 old_brow;
            u32 new_brow;
            if (rowp < 0) {
                new_brow = 0;
            } else {
                new_brow = new_bmp[rowp];
            }

            if (row >= BMP_ALIEN_H) {
                old_brow = 0;
            } else {
                old_brow = old_bmp[row];
            }
            u16 lx = 0;
            u16 alien_x = 0;
            for (alien_x = 0; alien_x < GAME_ALIEN_COLS; alien_x++, lx
                    +=GAME_ALIEN_SEPX) {
                u32 new;
                u32 old;
                if (alien_block->row_col.x == alien_x
                        && alien_block->row_col.y == alien_row) {

                    new = (rowp < 0) ? (0)
                            : (bmp_alien_explosion_12x10[rowp]);
                    old = old_brow;

                } else if (!check(alien_block->alien_status[alien_row],alien_x)) { //Only draw alien if not dead
```

```
                        new = 0;
                        old = new = (rowp < 0) ? (0)
                                : (bmp_alien_explosion_12x10[rowp]);
                    } else {
                        new = new_brow;
                        old = old_brow;
                    }

                    update_bmp_row(x + lx, y + rowy + row, old, new,
                            ALIEN_COLOR);

                }
            }
        }

    }
// If block hasnt moved, just check for killed aliens
    else if (alien_block->pos.x == prev_block.pos.x && alien_block->pos.y
            == prev_block.pos.y) {
            //blocks are the same
        u16 i;
        for (i = 0; i < ALIENS_ROW_COUNT; i++) {
            //Assuming aliens only die and do not resurrect
            //If there is a change, then something died
            u16 bit = prev_block.alien_status[i] ^ alien_block->alien_status[i];
            u16 n = 0;
            if (bit) {
                while (bit >>= 1) { //Which bit is it?
                    n++;
                }
                    //Draw over alien to delete
                draw_bitmap(bmp_alien_explosion_12x10, GAME_BACKGROUND,
                        prev_block.pos.x + n * GAME_ALIEN_SEPX,
                        alien_block->pos.y + i * GAME_ALIEN_SEPY, BMP_ALIEN_W,
                        BMP_ALIEN_H);
            }
        }
    }
    //If the block is moving too far or in both x and y directions, do a full erase and write
    else {
        const u32* bmp;
        s16 x;
        s16 y = prev_block.pos.y;
        for (u16 row = 0; row < GAME_ALIEN_ROWS; row++, y += GAME_ALIEN_SEPY) {

            bmp = bmp_aliens[prev_block.legs][row];
            x = prev_block.pos.x;
            for (u16 col = 0; col < GAME_ALIEN_COLS; col++, x
                    += GAME_ALIEN_SEPX) {
                draw_bitmap(bmp, GAME_BACKGROUND, x, y, BMP_ALIEN_W,
                        BMP_ALIEN_H);
            }
        }

        y = alien_block->pos.y;
        for (u16 row = 0; row < GAME_ALIEN_ROWS; row++, y += GAME_ALIEN_SEPY) {
            bmp = bmp_aliens[alien_block->legs][row];
            x = alien_block->pos.x;
            for (u16 col = 0; col < GAME_ALIEN_COLS; col++, x
                    +=GAME_ALIEN_SEPX) {
                draw_bitmap(bmp, ALIEN_COLOR, x, y, BMP_ALIEN_W, BMP_ALIEN_H);
            }
        }
    }
    //prev_block must be now equal to alien_block, so copy it over
    memcpy(&prev_block, alien_block, sizeof(alien_block_t));
}

#define MAX_TANK_DISTANCE 2
//Update tank position
static void update_tank(tank_t* tank) {

    if (!tank->changed)
        return;
    //Tank is initially drawn in init, so prev_tank can be initialized to
```

```
    //   the start location of the tank
    const u32* old_bmp = bmp_tanks[prev_tank.state];
    const u32* tank_bmp = bmp_tanks[tank->state]; //Get tank bitmap

    u16 tank_y = tank->pos.y; //This doesnt change
    u16 y; //This does, as we move down each pixel row in the tank
    s16 offset = tank->pos.x - prev_tank.pos.x; //Shift distance

    if (abs(offset) <= MAX_TANK_DISTANCE) {

        for (y = 0; y < BMP_TANK_H; y++) { //Iterate over pixel rows in tank

            u32 new = tank_bmp[y];
            u32 prev = old_bmp[y];
            new = (offset > 0) ? (new << offset) : (new >> -offset);

            update_bmp_row(prev_tank.pos.x, y + tank_y, prev, new, COLOR_GREEN);
        }
    } else {
        draw_bitmap(old_bmp, GAME_BACKGROUND, prev_tank.pos.x, prev_tank.pos.y,
                bmp_tank_dim.x, bmp_tank_dim.y);
        draw_bitmap(tank_bmp, COLOR_GREEN, tank->pos.x, tank->pos.y,
                bmp_tank_dim.x, bmp_tank_dim.y);

    }
    tank->changed = 0;
    prev_tank.pos.x = tank->pos.x;
    prev_tank.state = tank->state;
}

#define SAUCER_X 321
#define SAUCER_Y 15
#define HUNDRED 100
#define ZERO 0
//Update tank position
static void update_saucer(saucer_t* saucer) {
    //saucer is initially drawn in init, so prev_saucer can be initialized to
    //   the start location of the saucer

    if (prev_saucer.alive && !saucer->alive) {

        draw_bitmap(bmp_saucer_16x7, COLOR_BLACK, saucer->pos.x, saucer->pos.y,
                BMP_SAUCER_W, BMP_SAUCER_H);
        u16 hundred = 0;
        if (saucer->points >= HUNDRED) {
            hundred = saucer->points / HUNDRED;
            draw_bitmap(bmp_numbers[hundred], COLOR_WHITE, saucer->pos.x,
                    saucer->pos.y, BMP_NUMBER_W, BMP_NUMBER_H);
        }
        u16 ten = (saucer->points - hundred * HUNDRED) / TEN;
        draw_bitmap(bmp_numbers[ten], COLOR_WHITE, saucer->pos.x + SCORE_GAP_1,
                saucer->pos.y, BMP_NUMBER_W, BMP_NUMBER_H);
        draw_bitmap(bmp_numbers[ZERO], COLOR_WHITE,
                saucer->pos.x + SCORE_GAP_2, saucer->pos.y, BMP_NUMBER_W,
                BMP_NUMBER_H);
        memcpy(&prev_saucer, saucer, sizeof(saucer_t));
        return;
    } else if (!prev_saucer.alive && saucer->alive) {

        u16 hundred = 0;
        if (prev_saucer.points >= HUNDRED) {
            hundred = prev_saucer.points / HUNDRED;
            draw_bitmap(bmp_numbers[hundred], COLOR_BLACK, prev_saucer.pos.x,
                    prev_saucer.pos.y, BMP_NUMBER_W, BMP_NUMBER_H);
        }
        u16 ten = (prev_saucer.points - hundred * HUNDRED) / TEN;
        draw_bitmap(bmp_numbers[ten], COLOR_BLACK,
                prev_saucer.pos.x + SCORE_GAP_1, prev_saucer.pos.y,
                BMP_NUMBER_W, BMP_NUMBER_H);
        draw_bitmap(bmp_numbers[ZERO], COLOR_BLACK,
                prev_saucer.pos.x + SCORE_GAP_2, prev_saucer.pos.y,
                BMP_NUMBER_W, BMP_NUMBER_H);

        memcpy(&prev_saucer, saucer, sizeof(saucer_t));
    }
```

```
    if (saucer->pos.x == prev_saucer.pos.x || !saucer->alive)
        return;
    const u32* saucer_bmp = bmp_saucer_16x7; //Get saucer bitmap
    u16 y = saucer->pos.y; //This doesnt change
    u16 saucer_y; //This does, as we move down each pixel row in the saucer
    for (saucer_y = 0; saucer_y < BMP_SAUCER_H; saucer_y++) { //Iterate over pixel rows in saucer
        s16 offset = saucer->pos.x - prev_saucer.pos.x; //Shift distance
        u32 delta; //These are the same as in update_alien_row
        u32 set_delta;
        u32 reset_delta;
        u32 new = saucer_bmp[saucer_y];
        u32 prev = saucer_bmp[saucer_y];
        if (offset > 0) {
            new = new << offset;
        } else if (offset < 0) {
            offset = -offset;
            new = new >> offset;
        }
        delta = new ^ prev;

        u16 x;
        x = prev_saucer.pos.x;
        set_delta = delta & ~prev;
        reset_delta = delta & prev;

        while (set_delta || reset_delta) {
            if (set_delta & BIT0) {
                set_point(x, y + saucer_y, COLOR_RED);
            }
            if (reset_delta & BIT0) {
                clr_point (x,y+saucer_y);
            }

            reset_delta >>= 1;
            set_delta >>= 1;
            x++;
        }
    }
    memcpy(&prev_saucer, saucer, sizeof(saucer_t));

}

void update_bmp_row(s16 x, s16 y, u32 old, u32 new, u32 color) {
    u32 delta = old ^ new;
    u32 set_delta = delta & ~old;
    u32 reset_delta = delta & old;

    s16 lx = x;
    while (set_delta || reset_delta) {
        if (set_delta & BIT0) {
            set_point(lx, y, color);
        }
        if (reset_delta & BIT0) {
            clr_point (lx, y);
        }

#define SHIFT1 1
        reset_delta >>= SHIFT1;
        set_delta >>= SHIFT1;
        lx++;
    }
}

//Update bunkers when eroded
static void update_bunkers(bunker_t* bunkers) {

    bunker_t* bunker;
    bunker_t* prev_bunker;
    u16 bunker_num;
    //Iterate over bunkers
    for (bunker_num = 0; bunker_num < GAME_BUNKER_COUNT; bunker_num++) {
        bunker = bunkers + bunker_num;
        prev_bunker = prev_bunkers + bunker_num;
```

```
        if (!bunker->changed)
            continue;

        u16 bunker_x = GAME_BUNKER_POS + GAME_BUNKER_SEP * bunker_num;
        u16 bunker_y = GAME_BUNKER_Y;
        u16 block_num;
        for (block_num = 0; block_num < GAME_BUNKER_BLOCK_COUNT; block_num++) {
            if (!bunker->block[block_num].changed)
                continue;
            //xil_printf("\tblock %d changed\n\r", block_num);
            u16 x = bunker_x + BMP_BUNKER_BLOCK_W * (block_num
                % GAME_BUNKER_WIDTH);
            u16 y = bunker_y + BMP_BUNKER_BLOCK_H * (block_num
                / GAME_BUNKER_WIDTH);

            u16 block_row;
            for (block_row = 0; block_row < BMP_BUNKER_BLOCK_H; block_row++) {
                //draw_bitmap
                u32
                        old_row =
                                bmp_bunker_blocks[block_num][block_row]
                                        & bmp_bunker_damages[prev_bunker->block[block_num].block_health][block_row];
                u32
                        new_row =
                                bmp_bunker_blocks[block_num][block_row]
                                        & bmp_bunker_damages[bunker->block[block_num].block_health][block_row];

                update_bmp_row(x, y, old_row, new_row, COLOR_GREEN);
                y++;
            }

            bunker->block[block_num].changed = 0;
        }
        bunker->changed = 0;
        memcpy(prev_bunker, bunker, sizeof(bunker_t));
    }
}

#define MISSILE_SHIFT 2

//Update missile sprites, tank and alien missiles
static void update_missiles(tank_t * tank, alien_missiles_t* alien_missiles) {
    do {
        u16 py; //old y
        u16 n;
        //Iterate over missiles
        for (n = 0; n < GAME_MISSILE_COUNT; n++) {

            alien_missiles_t* prev_missile = prev_missiles + n;
            alien_missiles_t* missile = alien_missiles + n;
            //Tank bullet
            if (missile->pos.xy == prev_missile->pos.xy && missile->active
                    == prev_missile->active)
                continue;
            if (prev_missile->active == 0) {
                prev_missile->pos.xy = missile->pos.xy;
                prev_missile->guise = missile->guise;
                prev_missile->type = missile->type;
            }
            s16 y = prev_missile->pos.y;
            s16 dy = missile->pos.y - prev_missile->pos.y;
            u16 x = prev_missile->pos.x;

            const u32* old_bmp;
            const u32* new_bmp;
            if (prev_missile->active)
                old_bmp
                        = bmp_alien_missiles[prev_missile->type][prev_missile->guise];
            else
                old_bmp = bmp_empty_projectile;
            if (missile->active)
                new_bmp = bmp_alien_missiles[missile->type][missile->guise];
            else
                new_bmp = bmp_empty_projectile;
            s16 row; //from 0 to bmp height + dy. When less than bmp_height, is row index for non shifted bitmap
```

```
            s16 rowp; //from -dy to bmp_height + dy. When positive, is bitmap row index for shifted bitmap
            for (row = 0, rowp = -dy; row < BMP_BULLET_H + dy; row++, rowp++) {
                u32 old_brow;
                u32 new_brow;
                if (rowp < 0) {
                    new_brow = 0;
                } else {
                    new_brow = new_bmp[rowp];
                }
                if (row >= BMP_BULLET_H) {
                    old_brow = 0;
                } else {
                    old_brow = old_bmp[row];
                }

                update_bmp_row(x, y + row, old_brow, new_brow, COLOR_WHITE);
            }
            //update prev state to current state
            prev_missile->guise = missile->guise;
            prev_missile->pos.xy = missile->pos.xy;
            prev_missile->active = missile->active;

        }

        //Tank bullet
        if (prev_tank.missile.pos.xy == tank->missile.pos.xy
                && prev_tank.missile.active == tank->missile.active)
            break;
        if (prev_tank.missile.active == 0) {
            prev_tank.missile.pos.xy = tank->missile.pos.xy;
        }
        s16 y = tank->missile.pos.y;
        s16 dy = prev_tank.missile.pos.y - tank->missile.pos.y;
        u16 x = prev_tank.missile.pos.x;

        const u32* old_bmp;
        const u32* new_bmp;
        if (prev_tank.missile.active)
            old_bmp = bmp_bullet_straight_3x5;
        else
            old_bmp = bmp_empty_projectile;
        if (tank->missile.active)
            new_bmp = bmp_bullet_straight_3x5;
        else
            new_bmp = bmp_empty_projectile;
        s16 row; //Same as above but in the reverse direction. row is index for shifted bitmap (shifted up)
        s16 rowp; //while rowp is index for unshifted bitmap (old)
        for (row = 0, rowp = -dy; row < BMP_BULLET_H + dy; row++, rowp++) {
            u32 old_brow;
            u32 new_brow;
            if (row >= BMP_BULLET_H) {
                new_brow = 0;
            } else {
                new_brow = new_bmp[row];
            }

            if (rowp < 0) {
                old_brow = 0;
            } else {
                old_brow = old_bmp[rowp];
            }

            update_bmp_row(x, y + row, old_brow, new_brow, COLOR_WHITE);
        }
        prev_tank.missile.pos.xy = tank->missile.pos.xy;
        prev_tank.missile.active = tank->missile.active;
    } while (false);//do{}while(0); is to allow breaking out
    return;
}

#define LIFE_X 230
#define LIFE_COUNT 3 //number of tank lives
#define LIFE_DEC -1
#define LIFE_INC 1
void update_tank_life(u8 tank_lives) {
```

```
    if (tank_lives == prev_lives)
        return;
    u16 life = prev_lives;
    s16 delta = (tank_lives > prev_lives) ? (LIFE_INC) : (LIFE_DEC);
    u16 offset; //create the offset for the tanks
    for (; life != tank_lives; life += delta) {

        //draw black
        if (delta < 0) {
            offset = LIFE_X + (life + LIFE_DEC) * TANK_SPACE; //calc x pos offset
            draw_bitmap(bmp_tank_15x8, COLOR_BLACK, offset, TOP_SCREEN,
                    BMP_TANK_W, BMP_TANK_H);
        } else {
            offset = LIFE_X + life * TANK_SPACE; //calc x pos offset
            draw_bitmap(bmp_tank_15x8, COLOR_GREEN, offset, TOP_SCREEN,
                    BMP_TANK_W, BMP_TANK_H);
        }
    }
    prev_lives = tank_lives;
}

//Externally accessible render function. Calls local functions to render graphics
void render(tank_t* tank, alien_block_t* alienBlock,
        alien_missiles_t* alien_missiles, bunker_t* bunkers, saucer_t* saucer,
        u32 score) {

    update_missiles(tank, alien_missiles);
    update_tank_life(tank->lives);
    update_score(score);
    update_bunkers(bunkers);
    update_tank(tank);
    update_saucer(saucer);
    update_alien_block(alienBlock);
}

#define RES_SCALE 2
//Utility function, converts between game resolution and screen resolution
static inline void set_point(s32 x, s32 y, u32 color) {
    if (x < 0 || x >= GAME_W || y < 0 || y >= GAME_H)
        return; //IF coordinates are outside bounds, do not draw
    //Scale x and y to screen coordinates
    x *= RES_SCALE;
    y *= RES_SCALE * GAME_SCREEN_W;
    //Draw 2x2 game pixel
    frame0[y + x] = color;
    frame0[y + x + 1] = color;
    y += GAME_SCREEN_W;
    frame0[y + x] = color;
    frame0[y + x + 1] = color;
}

//Utility function for drawing bitmaps on game screen.
static void draw_bitmap(const u32* bmp, u32 color, s16 bmp_x, s16 bmp_y,
        s16 bmp_w, s16 bmp_h) {
    u32 bmp_row; //pixel row of bitmap
    s16 y; //local y
    for (y = 0; y < bmp_h; y++) {
        bmp_row = bmp[y];
        s16 x = bmp_x;
        //Shift out pixels
        for (; bmp_row; bmp_row >>= 1) {
            if (bmp_row & BIT0) {
                set_point(x, y + bmp_y, color);
            }
            x++;//Move along (All American Rejects)
        }
    }
}

//Utility function for drawing a horizontal green line on game screen

static void drawGreenLine() {
    u32 row;
    u32 col;
```

```
    for (row = 0; row < 480; row++) {
        for (col = 0; col < 640; col++) {
            if (row >= 450 && row <= 452)
                frame0[row * 640 + col] = COLOR_GREEN;
        }
    }
}

/*
 * table.c
 *
 * Lookup tables and other optimizations.
 *
 *  Created on: Oct 5, 2017
 *      Author:  Broderick Gardner
 *               Benjamin Gardner
 */

#include "xil_types.h"

//Utility macro and defines for lookup table
#define BIT(x) (1 << (x))
#define BIT0 BIT(0)
#define BIT1 BIT(1)
#define BIT2 BIT(2)
#define BIT3 BIT(3)
#define BIT4 BIT(4)
#define BIT5 BIT(5)
#define BIT6 BIT(6)
#define BIT7 BIT(7)
#define BIT8 BIT(8)
#define BIT9 BIT(9)
#define BIT10 BIT(10)
#define BIT11 BIT(11)
#define BIT12 BIT(12)
#define BIT13 BIT(13)
#define BIT14 BIT(14)
#define BIT15 BIT(15)

//Lookup table for bit position masks
const u16 table_bit[] = { BIT0, BIT1, BIT2, BIT3, BIT4, BIT5, BIT6, BIT7, BIT8,
        BIT9, BIT10, BIT11, BIT12, BIT13, BIT14, BIT15 };
/*
 * timing.c
 *
 *  Created on: Oct 17, 2017
 *      Author: superman
 */
#include <stdio.h>
#include <stdlib.h>
#include "time.h"
#include "xil_types.h"
#include "timing.h"
#include "control.h"
#include "game.h"
#include "render.h"

static enum {
    INIT = 0, //do nothing state
    RUNNING = 1,
    GAME_OVER = 2,
    WIN = 3
} state = INIT; //enum for declaring the states

#define ALIEN_PERIOD 80 //freqeuency of alien movement
#define BULLET_PERIOD 2 //frequency of bullet movement
#define MISSILE_PERIOD 12 //frequency of missile movement
#define SAUCER_PERIOD 20 //frequency of saucer movement
#define SAUCER_INIT 200 //initial frequency of saucer
#define ALIEN_MISS_PERIOD 200 //frequency of alien missile shooting
#define ALIEN_MISS_INIT 600 //initial frequency of alien missile shooting
#define EXPLODE_PERIOD 5
#define EXPLODE_WAIT 10

void timing_game_tick() {
```

```
    static u32 alien_period = ALIEN_PERIOD; //alien move
    static u32 bullet_period = BULLET_PERIOD; //bullet move
    static u32 missile_period = MISSILE_PERIOD; //missile move
    static u32 saucer_period = SAUCER_INIT; //saucer move
    static u32 alien_miss_period = ALIEN_MISS_INIT; //missile shoot
    static u32 explode_period = EXPLODE_PERIOD;

    switch (state) {
    case INIT:
        srand(time(0)); //random seed
        control_init(); //initialize the game
        render_init();
        state = RUNNING;
        break;
    case RUNNING:
        if (alien_period) //if alien timer is above 0
            alien_period--; //decrement
        if (bullet_period) //repeat logic for all other timers
            bullet_period--;
        if (missile_period) //missile timer
            missile_period--;
        if (saucer_period) //saucer timer
            saucer_period--;
        if (alien_miss_period) //missile shooting timer
            alien_miss_period--;
        if (explode_period)
            explode_period--;

        if (alien_period == 0) { //check to see if timers have decremented to 0
            control_update_alien_position();
            alien_period = ALIEN_PERIOD;
        } else if (missile_period == 0) {
            control_update_missiles();
            missile_period = MISSILE_PERIOD;
        } else if (bullet_period == 0) {
            bullet_period = BULLET_PERIOD;
            control_update_bullet();
        } else if (saucer_period == 0) {
            saucer_period = SAUCER_PERIOD;
            control_saucer_move(); //if active then move
        } else if (alien_miss_period == 0) {
            alien_miss_period = ALIEN_MISS_PERIOD;
            control_alien_fire_missile();
        } else if (explode_period == 0) {
            control_tank_explode();
            explode_period = EXPLODE_PERIOD;
        }

        break;
    case GAME_OVER:
        render_gameover();
        break;
    case WIN:
        timing_restart_game();
        break;
    }

    control_run();
}

void timing_set_gameover() {
    state = GAME_OVER;
}

void timing_set_win() {
    state = WIN;
}

void timing_restart_game() {
    control_init();
    render_restart();
    state = RUNNING;
}

#define RANDOM_UP_RANGE 750 //upper bound of the random wait time
```

```
#define RANDOM_LOW_RANGE 500 //lower bound
u32 saucer_pause() {
    //generate random number for the saucer wait period
    u32 random_no = (rand() % (RANDOM_UP_RANGE - RANDOM_LOW_RANGE)
            + RANDOM_LOW_RANGE);
    return random_no;
}
/*
 * gpio.c
 *
 *  Created on: Oct 17, 2017
 *      Author: superman
 */
#include <stdio.h>
#include "xil_types.h"
#include "platform.h"
#include "xparameters.h"
#include "xaxivdma.h"
#include "xio.h"
#include "xgpio.h"
#include "mb_interface.h"   // provides the microblaze interrupt enables, etc.
#include "xintc_l.h"        // Provides handy macros for the interrupt controller.
#include "gpio.h"
#include "timer.h"
//variables
XGpio gpLED; // This is a handle for the LED GPIO block.
XGpio gpPB; // This is a handle for the push-button GPIO block.
volatile u32 button_state = 0;
volatile u32 gpio_button_flag;
void gpio_init() {
    u32 success;
    success = XGpio_Initialize(&gpPB, XPAR_PUSH_BUTTONS_5BITS_DEVICE_ID);
    // Set the push button peripheral to be inputs.
    XGpio_SetDataDirection(&gpPB, 1, 0x0000001F);
    // Enable the global GPIO interrupt for push buttons.
    XGpio_InterruptGlobalEnable(&gpPB);
    // Enable all interrupts in the push button peripheral.
    XGpio_InterruptEnable(&gpPB, 0xFFFFFFFF);
}

void gpio_interrupt_handler() {


    // Clear the GPIO interrupt.
    XGpio_InterruptGlobalDisable(&gpPB); // Turn off all PB interrupts for now.
    button_state = XGpio_DiscreteRead(&gpPB, 1); // Get the current state of the buttons.

    //timer_set_debounce(); //Initiate debouncing

    XGpio_InterruptClear(&gpPB, 0xFFFFFFFF); // Ack the PB interrupt.
    XGpio_InterruptGlobalEnable(&gpPB); // Re-enable PB interrupts.
}
/*
 * Copyright (c) 2010-2011 Xilinx, Inc.  All rights reserved.
 *
 * Xilinx, Inc.
 * XILINX IS PROVIDING THIS DESIGN, CODE, OR INFORMATION "AS IS" AS A
 * COURTESY TO YOU.  BY PROVIDING THIS DESIGN, CODE, OR INFORMATION AS
 * ONE POSSIBLE   IMPLEMENTATION OF THIS FEATURE, APPLICATION OR
 * STANDARD, XILINX IS MAKING NO REPRESENTATION THAT THIS IMPLEMENTATION
 * IS FREE FROM ANY CLAIMS OF INFRINGEMENT, AND YOU ARE RESPONSIBLE
 * FOR OBTAINING ANY RIGHTS YOU MAY REQUIRE FOR YOUR IMPLEMENTATION.
 * XILINX EXPRESSLY DISCLAIMS ANY WARRANTY WHATSOEVER WITH RESPECT TO
 * THE ADEQUACY OF THE IMPLEMENTATION, INCLUDING BUT NOT LIMITED TO
 * ANY WARRANTIES OR REPRESENTATIONS THAT THIS IMPLEMENTATION IS FREE
 * FROM CLAIMS OF INFRINGEMENT, IMPLIED WARRANTIES OF MERCHANTABILITY
 * AND FITNESS FOR A PARTICULAR PURPOSE.
 *
 */

#include "xparameters.h"
#include "xil_cache.h"

#include "platform_config.h"
```

```c
#ifdef STDOUT_IS_PS7_UART
#include "xuartps.h"
#elif defined(STDOUT_IS_16550)
#include "xuartns550_l.h"
#endif

#define UART_BAUD 9600

void
enable_caches()
{
#ifdef __PPC__
    Xil_ICacheEnableRegion(CACHEABLE_REGION_MASK);
    Xil_DCacheEnableRegion(CACHEABLE_REGION_MASK);
#elif __MICROBLAZE__
#ifdef XPAR_MICROBLAZE_USE_ICACHE
    Xil_ICacheEnable();
#endif
#ifdef XPAR_MICROBLAZE_USE_DCACHE
    Xil_DCacheEnable();
#endif
#endif
}

void
disable_caches()
{
    Xil_DCacheDisable();
    Xil_ICacheDisable();
}

void
init_uart()
{
#ifdef STDOUT_IS_PS7_UART
    /* Use the PS UART for Zynq devices */
    XUartPs Uart_Ps_0;
    XUartPs_Config *Config_0 = XUartPs_LookupConfig(UART_DEVICE_ID);
    XUartPs_CfgInitialize(&Uart_Ps_0, Config_0, Config_0->BaseAddress);
    XUartPs_SetBaudRate(&Uart_Ps_0, UART_BAUD);
#elif defined(STDOUT_IS_16550)
    XUartNs550_SetBaud(STDOUT_BASEADDR, XPAR_XUARTNS550_CLOCK_HZ, UART_BAUD);
    XUartNs550_SetLineControlReg(STDOUT_BASEADDR, XUN_LCR_8_DATA_BITS);
#endif
}

void
init_platform()
{
    enable_caches();
    init_uart();
}

void
cleanup_platform()
{
    disable_caches();
}
/*
 * timer.c
 *
 *  Created on: Oct 17, 2017
 *      Author: superman
 */
#include "xil_types.h"

#include "timer.h"
#include "gpio.h"

//Defines
#define DEBOUNCE_COUNT 8

//Variables
volatile u16 debounce_cnt = 0;
volatile u16 timer_flag = 0;
```

```c
volatile u16 timer_missed = 0;

void timer_interrupt_handler() {
    if (timer_flag == 1) {
        timer_missed = 1;
    }
    timer_flag = 1;
    if (debounce_cnt && !(--debounce_cnt)) {
        gpio_button_flag = 1;
    }
}

void timer_set_debounce() {
    debounce_cnt = DEBOUNCE_COUNT;
}
/*
 * vdma.c
 *
 * Contains given code for video dma initialization
 *
 * Code here written by Dr. Hutchings and left unchanged.  Includes magic numbers.
 *
 *  Created on: Oct 4, 2017
 *      Author: Broderick Gardner
 *              Benjamin Gardner
 */
#include <stdio.h>

#include "vdma.h"
#include "xaxivdma.h"
#include "xparameters.h"
#include "xio.h"
#include "xil_types.h"

static XAxiVdma videoDMAController;

void vdma_init(u32 * frame0, u32 * frame1) {
    int Status; // Keep track of success/failure of system function calls.


    // 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;
    myFrameConfig.ReadDelayTimerCount = 10;
    myFrameConfig.WriteFrameCount = 2;
    myFrameConfig.WriteDelayTimerCount = 10;
    Status = XAxiVdma_SetFrameCounter(&videoDMAController, &myFrameConfig);
    if (Status != XST_SUCCESS) {
        xil_printf("Set frame counter failed %d\r\n", Status);
        if (Status == XST_VDMA_MISMATCH_ERROR)
            xil_printf("DMA Mismatch Error\r\n");
    }
    // Now we tell the driver about the geometry of our frame buffer and a few other things.
    // Our image is 480 x 640.
    XAxiVdma_DmaSetup myFrameBuffer;
    myFrameBuffer.VertSizeInput = 480; // 480 vertical pixels.
    myFrameBuffer.HoriSizeInput = 640 * 4; // 640 horizontal (32-bit pixels).
```

```
    myFrameBuffer.Stride = 640 * 4; // Dont' worry about the rest of the values.
    myFrameBuffer.FrameDelay = 0;
    myFrameBuffer.EnableCircularBuf = 1;
    myFrameBuffer.EnableSync = 0;
    myFrameBuffer.PointNum = 0;
    myFrameBuffer.EnableFrameCounter = 0;
    myFrameBuffer.FixedFrameStoreAddr = 0;
    if (XST_FAILURE == XAxiVdma_DmaConfig(&videoDMAController, XAXIVDMA_READ,
            &myFrameBuffer)) {
        xil_printf("DMA Config Failed\r\n");
    }
    // We need to give the frame buffer pointers to the memory that it will use. This memory
    // is where you will write your video data. The vdma IP/driver then streams it to the HDMI
    // IP.
    myFrameBuffer.FrameStoreStartAddr[0] = frame0;
    myFrameBuffer.FrameStoreStartAddr[1] = frame1;

    if (XST_FAILURE == XAxiVdma_DmaSetBufferAddr(&videoDMAController,
            XAXIVDMA_READ, myFrameBuffer.FrameStoreStartAddr)) {
        xil_printf("DMA Set Address Failed Failed\r\n");
    }
    // Print a sanity message if you get this far.
    xil_printf("Woohoo! I made it through initialization.\n\r");
    // Now, let's get ready to start displaying some stuff on the screen.
    // The variables framePointer and framePointer1 are just pointers to the base address
    // of frame 0 and frame 1.

    // This tells the HDMI controller the resolution of your display (there must be a better way to do this).
    XIo_Out32(XPAR_AXI_HDMI_0_BASEADDR, 640 * 480);

    // Start the DMA for the read channel only.
    if (XST_FAILURE == XAxiVdma_DmaStart(&videoDMAController, XAXIVDMA_READ)) {
        xil_printf("DMA START FAILED\r\n");
    }
    // We have two frames, let's park on frame 0. Use frameIndex to index them.
    // Note that you have to start the DMA process before parking on a frame.
    if (XST_FAILURE == XAxiVdma_StartParking(&videoDMAController, 0,
            XAXIVDMA_READ)) {
        xil_printf("vdma parking failed\n\r");
    }

}
```