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

#include <stdio.h>
#include <stdint.h>
#include <stdlib.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 "interface.h"
#define WORDS_HEIGHT 5      // height of score and lives
#define TANK_HEIGHT 8       // our tank is 8 high
#define GAME_X 320          // How wide our game screen is
#define LINE_Y 225          // Where the line at the bottom goes

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

#define LIVES_WIDTH 24      // How wide our lives display is
#define SCORE_WIDTH 28      // How wide our score is
#define TANK_WIDTH 15       // How wide our tank is
#define NUMBER_WIDTH 4      // How wide each number is
#define GREEN 0x0000FF00    // Hex for green
#define WHITE 0xFFFFFFF     // These
#define BLACK 0x0000000     // are colors
#define RED 0xFFF0000       // Shocking pink is the best one
#define SHOCKING_PINK 0xFF6FFF
#define MOTHER_SHIP_POINT_COLOR SHOCKING_PINK

#define WORDS_ROW_OFFSET 7      // which row to place words lives and row
#define LIVES_COL_OFFSET 220    // which col to place lives
#define SCORE_COL_OFFSET 15     // which col to place score
#define GAME_COL_OFFSET 110     // Game Over position
#define GAME_ROW_OFFSET 120     // Game Over position
#define OVER_COL_OFFSET 150     // Game Over position
#define OVER_ROW_OFFSET 120     // Game Over position
#define SHIP_ROW 22             // row of the ship

#define DIGIT_ONE   55  // scores first digit
#define DIGIT_TWO   50  // scores second digit
#define DIGIT_THREE 45  // scores third digit
#define DIGIT_FOUR  40  // scores fourth digit
#define DIGIT_FIVE  35  // scores fifth digit
#define DIGIT_SIX   30  // scores sixth digit



// Packs each horizontal line of the figures into a single 32 bit word.
```

```c
#define packword15(b14,b13,b12,b11,b10,b9,b8,b7,b6,b5,b4,b3,b2,b1,b0)  \
((b14 << 14) | (b13 << 13) | (b12 << 12) | (b11 << 11) | (b10 << 10) | \
 (b9  << 9 ) | (b8  << 8 ) | (b7  << 7 ) | (b6  << 6 ) | (b5  << 5 ) | \
 (b4  << 4 ) | (b3  << 3 ) | (b2  << 2 ) | (b1  << 1 ) | (b0  << 0 ) )

static const uint32_t tank_15x8[TANK_HEIGHT] = {
packword15(0,0,0,0,0,0,0,1,0,0,0,0,0,0,0),
packword15(0,0,0,0,0,0,1,1,1,0,0,0,0,0,0),
packword15(0,0,0,0,0,0,1,1,1,0,0,0,0,0,0),
packword15(0,1,1,1,1,1,1,1,1,1,1,1,1,1,0),
packword15(1,1,1,1,1,1,1,1,1,1,1,1,1,1,1),
packword15(1,1,1,1,1,1,1,1,1,1,1,1,1,1,1),
packword15(1,1,1,1,1,1,1,1,1,1,1,1,1,1,1),
packword15(1,1,1,1,1,1,1,1,1,1,1,1,1,1,1)
};

#define packword4(b3,b2,b1,b0) ((b3  << 3 ) | (b2  << 2 ) | (b1  << 1 ) | (b0  << 0 ))
static const uint32_t ZERO_4x5[] ={ // sprite 0
    packword4(1,1,1,1), packword4(1,0,0,1), packword4(1,0,0,1),
    packword4(1,0,0,1), packword4(1,1,1,1)};
static const uint32_t ONE_4x5[] = { // sprite 1
    packword4(0,1,1,0), packword4(0,0,1,0), packword4(0,0,1,0),
    packword4(0,0,1,0), packword4(0,1,1,1)};
static const uint32_t TWO_4x5[] = { // sprite 2
    packword4(1,1,1,1), packword4(0,0,0,1), packword4(1,1,1,1),
    packword4(1,0,0,0), packword4(1,1,1,1)};
static const uint32_t THREE_4x5[] = { // sprite 3
    packword4(1,1,1,1), packword4(0,0,0,1), packword4(1,1,1,1),
    packword4(0,0,0,1), packword4(1,1,1,1)};
static const uint32_t FOUR_4x5[] = { // sprite 4
    packword4(1,0,0,1), packword4(1,0,0,1), packword4(1,1,1,1),
    packword4(0,0,0,1), packword4(0,0,0,1)};
static const uint32_t FIVE_4x5[] = { // sprite 5
    packword4(1,1,1,1), packword4(1,0,0,0), packword4(1,1,1,1),
    packword4(0,0,0,1), packword4(1,1,1,1)};
static const uint32_t SIX_4x5[] = { // sprite 6
    packword4(1,1,1,1), packword4(1,0,0,0), packword4(1,1,1,1),
    packword4(1,0,0,1), packword4(1,1,1,1)};
static const uint32_t SEVEN_4x5[] = { // sprite 7
    packword4(1,1,1,1), packword4(0,0,0,1), packword4(0,0,0,1),
    packword4(0,0,0,1), packword4(0,0,0,1)};
static const uint32_t EIGHT_4x5[] = { // sprite 8
    packword4(1,1,1,1), packword4(1,0,0,1), packword4(1,1,1,1),
    packword4(1,0,0,1), packword4(1,1,1,1)};
static const uint32_t NINE_4x5[] = { // sprite 9
    packword4(1,1,1,1), packword4(1,0,0,1), packword4(1,1,1,1),
    packword4(0,0,0,1), packword4(0,0,0,1)};


#define
packword28(b27,b26,b25,b24,b23,b22,b21,b20,b19,b18,b17,b16,b15,b14,b13,b12,b11,b10,b9,b8,b
7,b6,b5,b4,b3,b2,b1,b0) \
((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 ) |                         \
 (b7  << 7 ) | (b6  << 6 ) | (b5  << 5 ) | (b4  << 4 ) | (b3  << 3 ) | (b2  << 2 ) | (b1
<< 1 ) | (b0  << 0 ) )
static const uint32_t SCORE_28x5[SCORE_WIDTH] = { // sprite "SCORE"
    packword28(0,1,1,1,1,0,1,1,1,1,0,1,1,1,1,0,1,1,1,1,0,0,1,1,1,1,0,0),
    packword28(1,0,0,0,0,0,1,0,0,0,0,1,0,0,1,0,1,0,0,0,1,0,1,0,0,0,0,0),
    packword28(0,1,1,1,0,0,1,0,0,0,0,1,0,0,1,0,1,1,1,1,0,0,1,1,1,0,0,0),
    packword28(0,0,0,0,1,0,1,0,0,0,0,1,0,0,1,0,1,0,0,0,1,0,1,0,0,0,0,0),
    packword28(1,1,1,1,0,0,1,1,1,1,0,1,1,1,1,0,1,0,0,0,1,0,1,1,1,1,0,0)};
static const uint32_t GAME_28x5[SCORE_WIDTH] = { // sprite "GAME"
    packword28(0,1,1,1,1,0,0,0,1,0,0,0,1,0,0,0,1,0,1,1,1,1,0,0,0,0,0,0),
    packword28(1,0,0,0,0,0,0,1,0,1,0,0,1,1,0,1,1,0,1,0,0,0,0,0,0,0,0,0),
    packword28(1,0,1,1,1,0,1,0,0,0,1,0,1,0,1,0,1,0,1,1,1,1,0,0,0,0,0,0),
    packword28(1,0,0,0,1,0,1,1,1,1,1,0,1,0,0,0,1,0,1,0,0,0,0,0,0,0,0,0),
    packword28(1,1,1,1,0,0,1,0,0,0,1,0,1,0,0,0,1,0,1,1,1,1,0,0,0,0,0,0)};
static const uint32_t OVER_28x5[SCORE_WIDTH] = { // sprite "OVER"
    packword28(0,1,1,0,0,1,0,0,0,1,0,1,1,1,0,1,1,1,0,0,0,0,0,0,0,0,0,0),
    packword28(1,0,0,1,0,1,0,0,0,1,0,1,0,0,0,1,0,0,1,0,0,0,0,0,0,0,0,0),
    packword28(1,0,0,1,0,1,0,0,0,1,0,1,1,1,0,1,1,1,0,0,0,0,0,0,0,0,0,0),
    packword28(1,0,0,1,0,0,1,0,1,0,0,1,0,0,0,1,0,1,0,0,0,0,0,0,0,0,0,0),
    packword28(0,1,1,0,0,0,0,1,0,0,0,1,1,1,0,1,0,0,1,0,0,0,0,0,0,0,0,0)};
static const uint32_t WIN_28x5[SCORE_WIDTH] = { // sprite "WIN"
    packword28(0,0,0,0,0,1,0,1,0,1,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0),
    packword28(1,0,0,0,0,1,0,1,0,1,1,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0),
    packword28(1,0,1,0,0,1,0,1,0,1,0,1,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0),
    packword28(1,0,1,0,1,0,0,1,0,1,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0),
    packword28(0,1,1,1,0,0,0,1,0,1,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0)};

#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 ) )
static const uint32_t LIVES_24x5[LIVES_WIDTH] = { // sprite "LIVES"
    packword24(1,0,0,0,0,1,0,1,0,0,0,1,0,1,1,1,1,0,0,1,1,1,1,1),
    packword24(1,0,0,0,0,1,0,1,0,0,0,1,0,1,0,0,0,0,1,0,0,0,0,0),
    packword24(1,0,0,0,0,1,0,1,0,0,0,1,0,1,1,1,0,0,0,1,1,1,1,0),
    packword24(1,0,0,0,0,1,0,0,1,0,1,0,0,1,0,0,0,0,0,0,0,0,0,1),
    packword24(1,1,1,1,0,1,0,0,0,1,0,0,0,1,1,1,1,0,1,1,1,1,1,0)};


//-----------------------------------------------------------------------------
// Internal functions not defined in our .h
void interface_draw_line();     // Draws the line at the bottom of the screen
void interface_draw_tanks();    // Draws the "extra life" tanks
void interface_draw_lives();    //Draws the lives to the screen;
void interface_draw_score();    // Draws the Score to the screen
void interface_init_numbers();  //Draw the score in numbers
void interface_update_digit(const uint32_t number[], uint32_t digit);   // writes digit
void interface_digit(uint32_t value, uint32_t digit);           // Also.
void interface_draw_game_over();// Draws game over to the screen
void interface_update_ship_digit(const uint32_t number[], uint32_t digit, bool erase);
// End defining internal functions
//------------------------------------------------------------------------------

uint32_t * frame;               // How to write to the screen
```

```
int32_t lives = 3;              // How many lives do we have?
uint32_t score = 0;             // keep track of game score

//initialize the score board to all zeros
void interface_init_numbers(){                                //set the frame
    uint32_t row, col;                                            //declare vars
    for(row=0;row<WORDS_HEIGHT;row++){                        //through width
         for(col=0;col<NUMBER_WIDTH;col++){                   //and height
             if((ZERO_4x5[row] & (1<<(NUMBER_WIDTH-col-1)))){   //and draw score
                 util_draw_pixel(frame,row+WORDS_ROW_OFFSET,col+SCORE_COL_OFFSET
                         + DIGIT_ONE, GREEN);   //draw first digit
                 util_draw_pixel(frame,row+WORDS_ROW_OFFSET,col+SCORE_COL_OFFSET
                         + DIGIT_TWO, GREEN);   //draw second digit
                 util_draw_pixel(frame,row+WORDS_ROW_OFFSET,col+SCORE_COL_OFFSET
                         + DIGIT_THREE, GREEN); //draw third digit
                 util_draw_pixel(frame,row+WORDS_ROW_OFFSET,col+SCORE_COL_OFFSET
                         + DIGIT_FOUR, GREEN);  //draw fourth digit
                 util_draw_pixel(frame,row+WORDS_ROW_OFFSET,col+SCORE_COL_OFFSET
                         + DIGIT_FIVE, GREEN);  //draw fifth digit
                 util_draw_pixel(frame,row+WORDS_ROW_OFFSET,col+SCORE_COL_OFFSET
                         + DIGIT_SIX, GREEN);   //draw sixth digit
             }
         }
     }
}



#define INTERFACE_DIGIT_MOD 10
#define INTERFACE_START_COL 55
#define INTERFACE_COL_OFFSET 5
#define INTERFACE_SINGLE_DIGIT_DIVISION 1
#define INTERFACE_NUM_DIGITS 6

//increment the score by value
void interface_increment_score(uint32_t value){
    uint32_t i, temp_score; // initialize variables
    uint32_t mod = INTERFACE_DIGIT_MOD;             // set the modulus value
    uint32_t divide = INTERFACE_SINGLE_DIGIT_DIVISION;          // set the value to
divide by
    uint32_t digit_loc = INTERFACE_START_COL;       // set the column location of first
digit
    score += value;                 // increment the game score by value
    temp_score = score;             // set a temporary score to edit

    for(i = 0; i < INTERFACE_NUM_DIGITS; i++){              // loop through all six digits
        uint32_t number = temp_score % mod; // modulus the score
        number = number / divide;           // convert to a single digit value
        temp_score = temp_score - number;   // update the temporary score
        interface_digit(number,digit_loc);  // print to screen

        digit_loc -= INTERFACE_COL_OFFSET;  // update to the next digit column location
        divide *= INTERFACE_DIGIT_MOD;   // increment the number we divide by
        mod *= INTERFACE_DIGIT_MOD;      // increment the modulus number
    }
}

// convert a integer to a sprite to enable us to draw to screen
// value is the integer to print to screen
// digit is the column location of the digit to print to
void interface_digit(uint32_t value, uint32_t digit){
    switch(value){                                  // value the integer
        case 0:                                     // if value = 0
            interface_update_digit(ZERO_4x5,digit); // print 0 to location
            break;
        case 1:                                     // value = 1
            interface_update_digit(ONE_4x5,digit);  // print 1 to location
            break;
        case 2:                                     // value = 2
            interface_update_digit(TWO_4x5,digit);  // print 2 to location
            break;
        case 3:                                     // value = 3
            interface_update_digit(THREE_4x5,digit);// print 3 to location
            break;
        case 4:                                     // value = 4
            interface_update_digit(FOUR_4x5,digit); // print 4 to location
            break;
        case 5:                                     // value = 5
            interface_update_digit(FIVE_4x5,digit); // print 5 to location
            break;
        case 6:                                     // value = 6
            interface_update_digit(SIX_4x5,digit);  // print 6 to location
            break;
        case 7:                                     // value = 7
            interface_update_digit(SEVEN_4x5,digit);// print 7 to location
            break;
        case 8:                                     // value = 8
            interface_update_digit(EIGHT_4x5,digit);// print 8 to location
            break;
        case 9:                                     // value = 9
            interface_update_digit(NINE_4x5,digit); // print 9 to location
            break;
    }
}

//Draw the digit to the score
//number[] is the sprite of 1,2,3 ect.
//digit is the column offset of the screen to print to
void interface_update_digit(const uint32_t number[], uint32_t digit){
    uint32_t row, col;                                          //init row and col
    for(row=0;row<WORDS_HEIGHT;row++){                      // Go through width
         for(col=0;col<NUMBER_WIDTH;col++){                 // and height
             if((number[row] & (1<<(NUMBER_WIDTH-col-1)))){ // if  sprite
                 util_draw_pixel(frame,row+WORDS_ROW_OFFSET,col+SCORE_COL_OFFSET
                         + digit, GREEN);   // print to pixel green
             }else{                         // if value = 0
                 util_draw_pixel(frame,row+WORDS_ROW_OFFSET,col+SCORE_COL_OFFSET
                         + digit, BLACK);   // print to pixel black
             }
         }
     }
}

//initialize the entire screen
void interface_init_board(uint32_t * framePointer){
    frame = framePointer;           // Set the pointer to the screen
    interface_draw_score();         // Draw a score (0)
    interface_draw_lives();         // Draw "lives"
    interface_draw_line();          // Draw the line at the bottom
    interface_draw_tanks();         // Draw our extra lives
    interface_init_numbers();       // Make numbers good
}


//This draws the word score to the screen.
void interface_draw_score(){
    uint32_t row, col;
    for(row=0;row<WORDS_HEIGHT;row++){                      // Go through width
         for(col=0;col<SCORE_WIDTH;col++){                  // and height
             if((SCORE_28x5[row] & (1<<(SCORE_WIDTH-col-1)))){// and draw score
                 util_draw_pixel(frame,row+WORDS_ROW_OFFSET,col+SCORE_COL_OFFSET
                         , WHITE);      // draw white
             }
         }
     }
}

//This draws the word lives to the screen.
void interface_draw_lives(){
    uint32_t row, col;
    for(row=0;row<WORDS_HEIGHT;row++){                      // Go through width
         for(col=0;col<LIVES_WIDTH;col++){                  // and height
             if((LIVES_24x5[row] & (1<<(LIVES_WIDTH-col-1)))){// and draw Lives
                 util_draw_pixel(frame, row + WORDS_ROW_OFFSET, col +
                         LIVES_COL_OFFSET, WHITE);      // draw white
             }
         }
     }
}
// This draws the green line at the bottom of the screen
void interface_draw_line(){
    uint32_t row, col;                              // Initialize
    row = LINE_Y;                               // variables
    for(col=0;col<GAME_X;col++){                // Go along the screen and draw
        util_draw_pixel(frame, row, col, GREEN);//draw green
    }
}

// This draws the extra tanks to the screen
void interface_draw_tanks(){
    uint32_t row, col;                                          // Init loop vars
     for(row=0;row<TANK_HEIGHT;row++){                      // Go through width
         for(col=0;col<TANK_WIDTH;col++){                   // and height
             if((tank_15x8[row] & (1<<(TANK_WIDTH-col-1)))) {// and draw 3 tanks
                 util_draw_pixel(frame, row+EXTRA_TANK_Y_OFFSET,
                         col+EXTRA_TANK_0, GREEN);
                 util_draw_pixel(frame, row+EXTRA_TANK_Y_OFFSET,
                         col+EXTRA_TANK_1, GREEN);
                 util_draw_pixel(frame, row+EXTRA_TANK_Y_OFFSET,
                         col+EXTRA_TANK_2, GREEN);
             }
         }
     }
}
```

```
// This draws the game over screen
void interface_draw_game_over(){
    uint32_t row, col;
    for(row=0;row<WORDS_HEIGHT;row++){                      // Go through width
         for(col=0;col<SCORE_WIDTH;col++){                  // and height
             if((GAME_28x5[row] & (1<<(SCORE_WIDTH-col-1)))){// and draw score
                 util_draw_pixel(frame, row + GAME_ROW_OFFSET,
                         col + GAME_COL_OFFSET, RED);       // draw white
             }
         }
     }
    for(row=0;row<WORDS_HEIGHT;row++){                      // Go through width
         for(col=0;col<SCORE_WIDTH;col++){                  // and height
             if((OVER_28x5[row] & (1<<(SCORE_WIDTH-col-1)))){// and draw score
                 util_draw_pixel(frame, row + OVER_ROW_OFFSET,
                         col + OVER_COL_OFFSET, RED);       // draw white
             }
         }
     }
}

// This kills a tank
void interface_kill_tank(){
    lives--;                                // Take a live
    if(lives < 0){                          // maybe game over
        interface_draw_game_over();         // Game over
        interface_game_over();
    }


    uint32_t row, col;
    switch(lives){                                          // lives left
    case 2:                                                 // lives = 2
        for(row=0;row<TANK_HEIGHT;row++){                   // Go through width
             for(col=0;col<TANK_WIDTH;col++){               // and height
                 if((tank_15x8[row] & (1<<(TANK_WIDTH-col-1)))) {// draw 3 tanks
                     util_draw_pixel(frame, row+EXTRA_TANK_Y_OFFSET,
                             col+EXTRA_TANK_2, BLACK);
                 }
             }
         }
        break;
    case 1:                                                 // lives = 1
        for(row=0;row<TANK_HEIGHT;row++){                   // Go through width
             for(col=0;col<TANK_WIDTH;col++){               // and height
                 if((tank_15x8[row] & (1<<(TANK_WIDTH-col-1)))) {// draw 3 tanks
                     util_draw_pixel(frame, row+EXTRA_TANK_Y_OFFSET,
                             col+EXTRA_TANK_1, BLACK);
                 }
             }
         }
        break;
    case 0:                                                 //zero lives left
        for(row=0;row<TANK_HEIGHT;row++){                   // Go through width
             for(col=0;col<TANK_WIDTH;col++){               // and height
                 if((tank_15x8[row] & (1<<(TANK_WIDTH-col-1)))) {// draw 3 tanks
                     util_draw_pixel(frame, row+EXTRA_TANK_Y_OFFSET,
                             col+EXTRA_TANK_0, BLACK);
```

```c
                    }
                }
            }
        break;
    }
}

// We have game over!
void interface_game_over(){
    interface_draw_game_over();     // draw "game over"
    //xil_printf("game over\n\r");  // print it.
    exit(1);                        // and kill program
}

// Draw the win screen
void interface_success(){
    uint32_t row, col;
    for(row=0;row<WORDS_HEIGHT;row++){                  // Go through width
         for(col=0;col<SCORE_WIDTH;col++){              // and height
             if((WIN_28x5[row] & (1<<(SCORE_WIDTH-col-1)))){// and draw score
                 util_draw_pixel(frame, row + GAME_ROW_OFFSET,
                         col + GAME_COL_OFFSET, RED);       // draw white
             }
         }
     }
    //xil_printf("you win!\n\r");
    exit(1);        // Kill the program
}

// convert a integer to a sprite to enable us to draw to screen
// value is the integer to print to screen
// digit is the column location of the digit to print to
void interface_ship_digit(const uint32_t value, uint32_t digit, bool erase){
    switch(value){                                  // value the integer
        case 0:                                     // if value = 0
            interface_update_ship_digit(ZERO_4x5,digit, erase); // print 0 to location
            break;
        case 1:                                     // value = 1
            interface_update_ship_digit(ONE_4x5,digit,  erase); // print 1 to location
            break;
        case 2:                                     // value = 2
            interface_update_ship_digit(TWO_4x5,digit,  erase); // print 2 to location
            break;
        case 3:                                     // value = 3
            interface_update_ship_digit(THREE_4x5,digit,  erase);// print 3 to location
            break;
        case 4:                                     // value = 4
            interface_update_ship_digit(FOUR_4x5,digit,  erase);    // print 4 to location
            break;
        case 5:                                     // value = 5
            interface_update_ship_digit(FIVE_4x5,digit,  erase);    // print 5 to location
            break;
        case 6:                                     // value = 6
            interface_update_ship_digit(SIX_4x5,digit,  erase); // print 6 to location
            break;
        case 7:                                     // value = 7
            interface_update_ship_digit(SEVEN_4x5,digit,  erase);// print 7 to location
            break;
        case 8:                                     // value = 8
            interface_update_ship_digit(EIGHT_4x5,digit,  erase);// print 8 to location
            break;
        case 9:                                     // value = 9
            interface_update_ship_digit(NINE_4x5,digit,  erase);    // print 9 to location
            break;
    }
}

//Draw the digit to the score
//number[] is the sprite of 1,2,3 ect.
//digit is the column offset of the screen to print to
void interface_update_ship_digit(const uint32_t number[], uint32_t digit, bool erase){
    uint32_t color = erase ? BLACK : MOTHER_SHIP_POINT_COLOR;
    uint32_t row, col;  //initialize row and column
    for(row=0;row<WORDS_HEIGHT;row+
+){                                                                  // Go through width
         for(col=0;col<NUMBER_WIDTH;col+
+){                                                              // and height
             if((number[row] &
(1<<(NUMBER_WIDTH-col-1)))){                                              // if value
in sprite = 1
                 util_draw_pixel(frame, row + SHIP_ROW, col + SCORE_COL_OFFSET + digit,
color); // print to pixel green

 }else{
/ if value = 0
                 util_draw_pixel(frame, row + SHIP_ROW, col + SCORE_COL_OFFSET + digit,
BLACK); // print to pixel black
             }
         }
     }
}


#define INTERFACE_NUM_MOTHERSHIP_DIGITS 3
// print the alien points of ship
void interface_alien_ship_points(uint32_t mother_ship_points, uint32_t col_loc, bool
erase){
//  xil_printf("printing points %d\n\r", mother_ship_points);

    uint32_t i, temp_score; // initialize variables
    uint32_t mod = INTERFACE_DIGIT_MOD;             // set the modulus value
    uint32_t divide = INTERFACE_SINGLE_DIGIT_DIVISION;          // set the value to
divide by
    temp_score = mother_ship_points;            // set a temporary score to edit
    for(i = 0; i < INTERFACE_NUM_MOTHERSHIP_DIGITS; i++){               // loop through
all six digits
        uint32_t number = temp_score % mod; // modulus the score
        number = number / divide;           // divide the number to convert to a single
digit value
        temp_score = temp_score - number;   // update the temporary score
        interface_ship_digit(number,col_loc,erase);     // print to screen

        col_loc -= INTERFACE_COL_OFFSET;    // update to the next digit column location
        divide *= INTERFACE_DIGIT_MOD;   // increment the number we divide by
        mod *= INTERFACE_DIGIT_MOD;      // increment the modulus number
    }
```

```
}
```