CORNELL UNIVERSITY, SCHOOL OF ELECTRICAL AND COMPUTER ENGINEERING

Lunar Lander

ECE 5760

Jeremy Blum, Sima Mitra, Jason Wright

Thursday Lab Section

An FPGA implementation of the game Lunar Lander.

Introduction

The game Lunar Land was implemented on DE2 using a NiosII. The objective of the game was to provide a controlled descent of a lunar lander onto a lunar surface with a horizontal and vertical velocity below a threshold and for the lander to have no rotation. The lander would crash if these parameters were not met or it the lander collided with the edges of the screen. If landing was successful a winner message, "Perfect landing!" was displayed. In the event of a crash a loser message, "You died." was displayed along with an explosion avatar at the location of the lander. These messages would be displayed until the game was reset via a key 0 press.

The lander was controlled via three pushbuttons that controlled the left and right rotation and the thruster. The thruster propelled the lander in the direction of lander's current heading with a thrust 2 times that of the gravity of the environment. The lander was provided with limited supply of fuel that was depleted as the thruster was activated. The current fuel level was displayed in the upper left corner of the screen was a bar graph color coded to the amount of fuel remaining. If all fuel was depleted, the message, "No Fuel!" would be displayed and thrusting would be disabled.

A hardware timer was also implemented to display the current elapsed time in seconds on the seven segment displays on the FPGA. Upon reset, the timer would be reset to zero and the game would restart with the lunar lander in the upper right corner of the screen moving a left with a certain velocity and falling under the effect of gravity.

Design and Testing Methods

Automatic Sprite Generation using MATLAB with antialiasing and 8 bit color Mapping

Instead of having our CPU recompute the spaceship rotations in real-time, we decided the write a MATLAB program that would generate a C header file containing a 3D array representation of all possible spaceship sprite rotations and thruster configurations. Our primary reason for doing this was having a desire to make the rotated spaceships look as nice as possible. Since we chose a complex, non-radially symmetric chip design, using a simple rotation matrix results in some fairly bizarre-looking rotated ships. The solution for this is to implement interpolation and antialiasing. While it would have been possible to do this in C, MATLAB has built-in functions for doing this with *imrotate*, so we decided to use MATLAB for sprite generation instead. We designed the ship in the upright position, using a byte of data to represent each pixel color in 8 bit RGB. The sprite is a 30x30 array. The array is fed into MATLAB as an image and the *imrotate* command is used with interpolation and cropping to generate rotations of this matrix for every 5 degrees of rotation, up to 90 degrees in each direction. There is not sufficient M4K memory space to store all 37 rotations with the thruster flame both on and off, but there is enough to store all the rotations with a thruster on. So, we only store those rotations, and made the thruster flame a color that is not repeated elsewhere in the sprite. That color is checked for in the C program, and is only displayed if the thrust is active. The upright sprite with the thrust engaged looks like this:

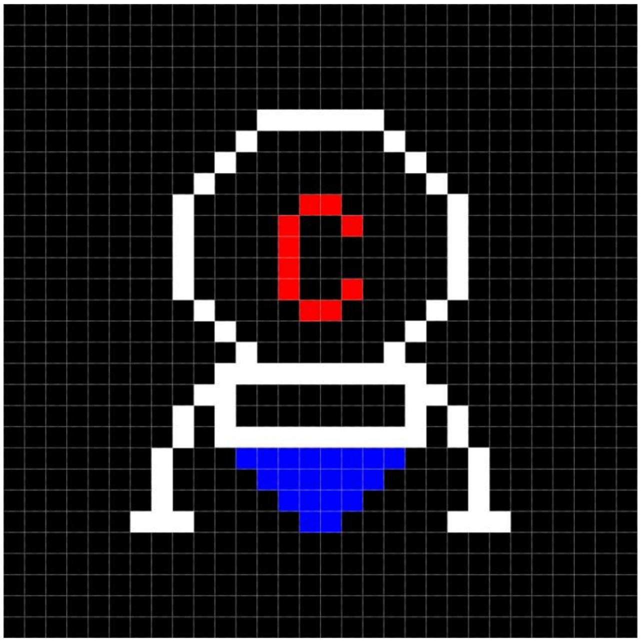

Figure 1. The base lander sprite

The sprite is rotated with bilinear interpolation, which, in matlab, results in the values of each cell being interpolated. Since each cell is represented by an 8-bit value, matlab will interpolate around these numbers. Unfortunately, when displayed on an 8-bit color display, this results in seemingly random colors representing the lander while it is rotating. To resolve this, a 24-bit to 8-bit color converter was written for matlab to facilitate proper rotation outputs. All the sprites are loaded into an h file automatically, which can be included from the main C file running on the Nios II CPU.

Inputs and Outputs

Five main inputs were implemented for the Lunar Lander game. The lander's left rotation was controlled with push-button 3 while the right rotation was controlled with push-buttons 3. The thruster was controlled with push-button 2. The game and timer reset was connected to push-button 0. The NiosII requires an external reset key to be connected for system, which is typically key0 in the DE2 Media Computer. In order to connect the game reset to key0, we chose to use switch 0 was the hardware reset. Therefore, in order for the program to run switch 0 must be set to 1.

The outputs of the gram include the 640x480 VGA out displayed on a screen to show the game and four 7-segment displays to display the elapsed time of the current game. The LEDs above the push-buttons were also configured to illuminate when there were pressed.

HIGH-LEVEL STRUCTURE

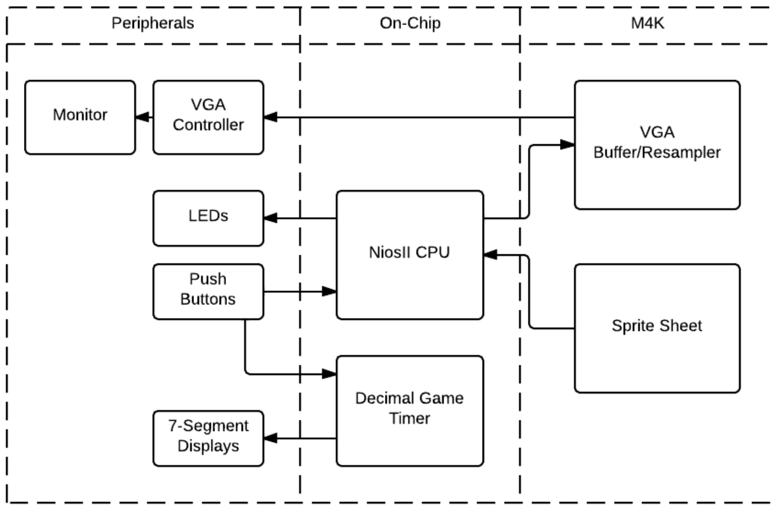

Figure 2. The register transfer level (RTL) diagram of the system. Some components created by SOPC builder are abstracted.

State-machine

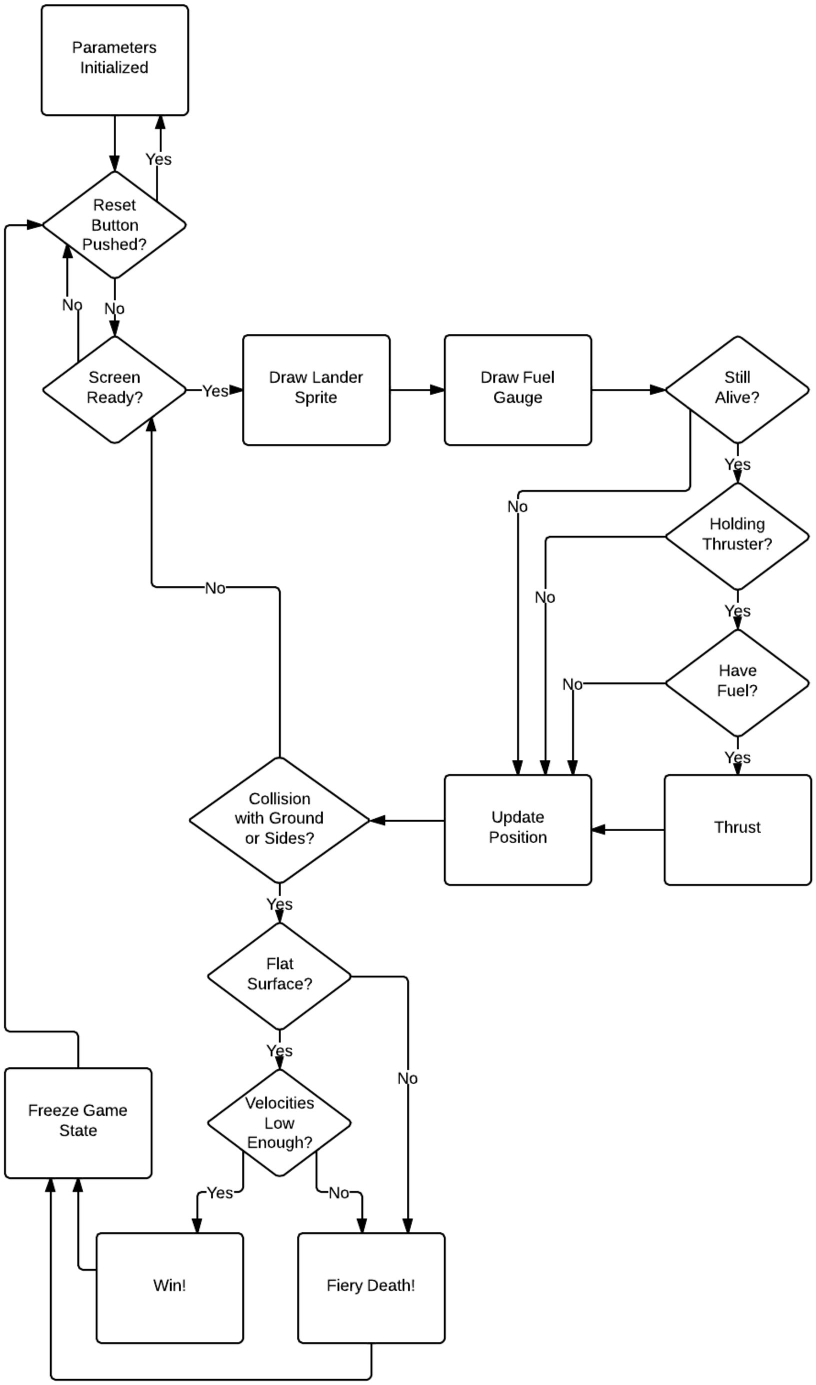

Figure 3. The state machine for the main program loop.

VGA

Displaying the game on the monitor using VGA was implemented using the VGA module in SOPC builder based on the setting of a DE2 Media Computer. The settings were altered from an example Media Computer according to the direction listed in the Configuring the NIOS II section of this report. This implementation allowed us to use builtin C functions for drawing pixels, lines, rectangles, boxes and text on the screen.

Lander drawing

Upon every iteration of the main loop the lander would be erased and redrawn to display the updated information. The lander was drawn on the screen by iterating through the array of pixel values for the necessary rotation of the lander and drawing the non-zero values individually using the alt_up_pixel_buffer_dma_draw function. If the thruster was currently activated, the appropriate pixels would be displayed; otherwise these colors would be displayed as black. If it had been determine the lander had crashed the explosion sprite would be displayed instead using a similar technique.

Timer Implementation

A one second timer was implemented using the 50 MHz clock of the FPGA by simply counting up to 50E6 clock cycles. Conversion from hexadecimal to decimal was implemented by using a series of counter variables that increased when 1, 10, 100 second passed and rolled over to zero at the appropriate times. The 7-segment display was then configured to display these counters, allowing 9999 seconds to be displayed before the display rolled over. The counter would be reset when KEY0 was pressed.

Crash Detection and Landing

Using a 30x30 sprite, crash detection can be a bit tricky since the rotation angle of the lander coupled with the slope of the line it is intersecting will change the point at which a collision event should be triggered. In its perfectly upright position, the lander does not extend out of a 20x20 box on the interior of the 30x30 sprite. As it rotates, it no longer fits perfectly within this internal box, but is still fairly close. We found that using the bottom right and bottom left corners of this box as potential intersection points works well when testing their intersection with the landscape. Depending on the angle, part of one strut may go 2 pixels through the surface, or the lander might not perfectly hit the surface, but the approximation is close enough to make playing the game fair and enjoyable with this decision.

There are a total of 7 collision conditions that need to be detected and dealt with correctly. First, consider the left/right screen edges, and the top edge. Detecting collision with these edges is easy. If the top of the lander sprite tries reaches an y position of zero, then the lander has gone off the top of the screen. If the left side of the sprite reaches position zero in the x coordinates, then the lander has gone off the left side of the screen. If right side of the lander sprite reaches the max x value (640), then the lander has gone off the right side of the screen.

The trickiest crash scenarios are with line segments B and D shown in the figure below. To determine if the lander has collided with either of these lines, the slopes of the lines are calculated. Note, the x and y origin is in the upper left corner of the screen, so the coordinate system is flipped upside-down. Taking this into account, the lander's x position is determined, and, if it within the x bounds of the slanted line segment and the slope from the bottom corner of the line to the either of the bottom corners of the sprite is less than the slope of the line, then a collision has occurred and the lander will explode.

Crashes can also occur on the horizontal surfaces, but the lander can also land there, so more conditions must be checked. If the lowest y point of the 20 pixel lander box has intersected one of the horizontal lines within the x bounds, then a check is made for rotation, y velocity, and x velocity. If the Y velocity is below 2.0 pixels/sec, the x velocity is below 3.0 pixels/sec, and the lander is upright, then the ship will land successfully. Otherwise, the Lander will crash.

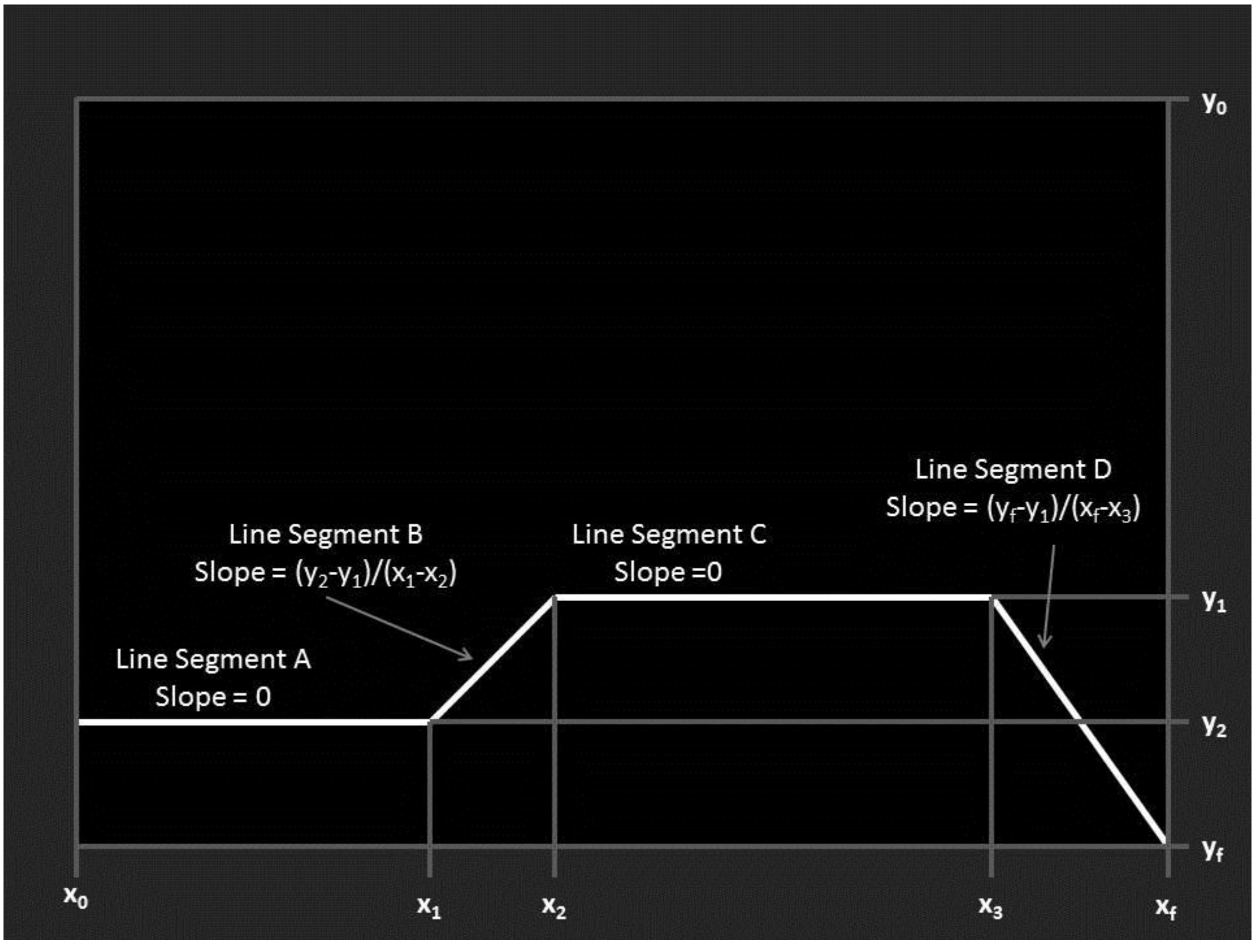

Figure 4. Graphical depiction of the gameplay environment.

The map, and key coordinate locations. Crashes are computed using the data in this figure.

FUEL GAUGE

The fuel gauge is drawn at every update step of the main program to give the player a measure of how much fuel is remaining. An unfortunately misspelled function *draw_fuel_guage(alt_up_pixel_buffer_dma_dev *pixel_buffer_dev, float fuel)* is defined to handle this automatically. A border rectangle is drawn using *alt_up_pixel_buffer_dma_draw_rectangle*, and two rectangles are drawn using *alt_up_pixel_buffer_dma_draw_box*. The left box changes color according to the amount of fuel left (starting at green, then yellow, then red), and the right box is always black (to match the background of the screen).

CONFIGURING THE NIOS II

We chose to use a NiosII CPU to easily send VGA commands based on on our program requirements. The University programs include a demo for setting up a basic multimedia CPU with a rendering example, but they are all based around a 320x240 resolution. Since we wanted to generate a 640x480 display, we applied the following steps to facilitate the use of a 640x480 video buffer:

We copied C:\altera\11.0\University_Program\NiosII_Computer_Systems\DE2\DE2_Media_Computer into our project folder

We opened Altera Monitor, went to New Project, and chose the directory and name

We used the custom system option so that we could use our SOPC builder adjustments.. For System Details, we selected the PTF file and SOF file from the directory we copied the DE2_Media_Computer folder to (they are in the verilog subfolder).

Under "Program Type", choose "Program with Device Driver Support".

Under Source files, we added the C file with our code that defines our software state machine as well as all the rendering information.

We opened up verilog/DE2_Media_Computer.qpf in Quartus

We opened up SOPC Builder by going to Tools > SOPC Builder

We deleted the VGA_Scaler module

Within the Pixel_Buffer module, we changed Width to 640, Height to 480, and Color Space to 8-bit RGB [r r r g g g b b].

We changed the Incoming Format to 8-bit RGB in the VGA_Pixel_RGB_Resampler Module

We changed the Alpha Blending Mode to Normal

We expanded the Expand VGA_Pixel_RGB_Resampler and accessed the avalon_rgb_source. We checked Alpha_Blending.avalon_background_sink

We then rebuilt the .ptf file

Finally, in Altera Monitor, we re-compiled the project and load it onto the board.

ARITHMETIC SYSTEM

We chose to use floating point numbers in order to provide enough precision for calculating the velocity of the lander and the amount of fuel remaining. Other parameters, like the lander sprites, that did not require this precision were saved as chars to minimize memory usage.

DOCUMENTATION

EXAMPLE IMAGES

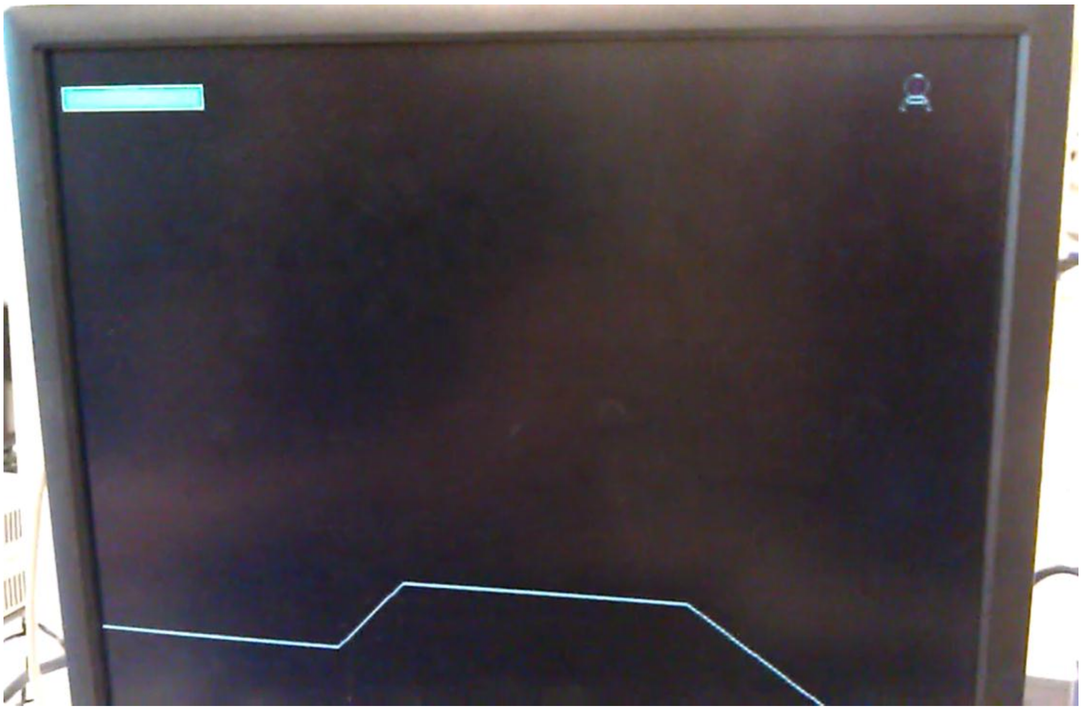

Figure 5: Begining a new game of lunar lander. The lander starts in the upper right corner of the screen with a non-zero horizontal velocity and with 100% fuel (shown in green).

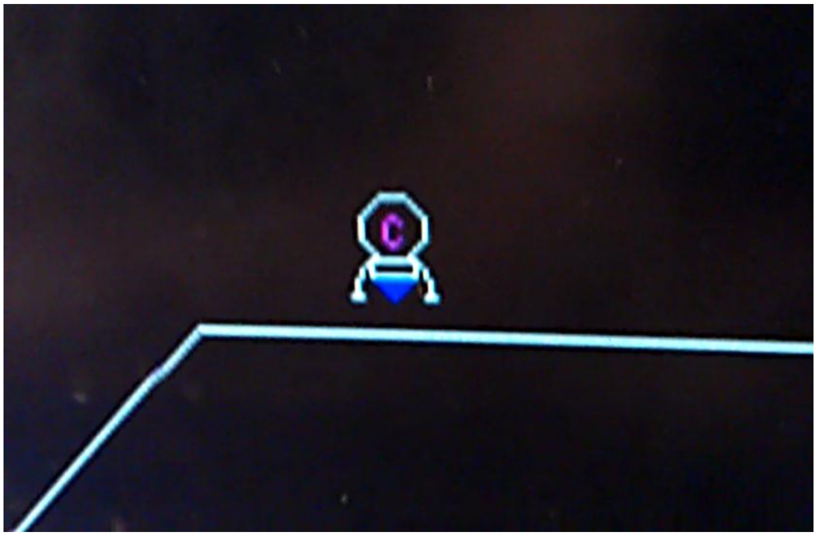

Figure 6: THe lander with thruster activated.

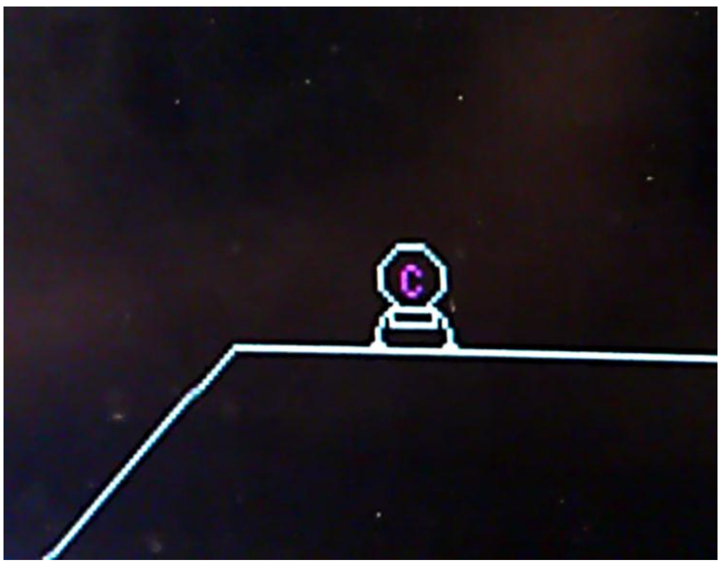

Figure 7: A successful landing.

Figure 8: The winnder message displayed after successful landing.

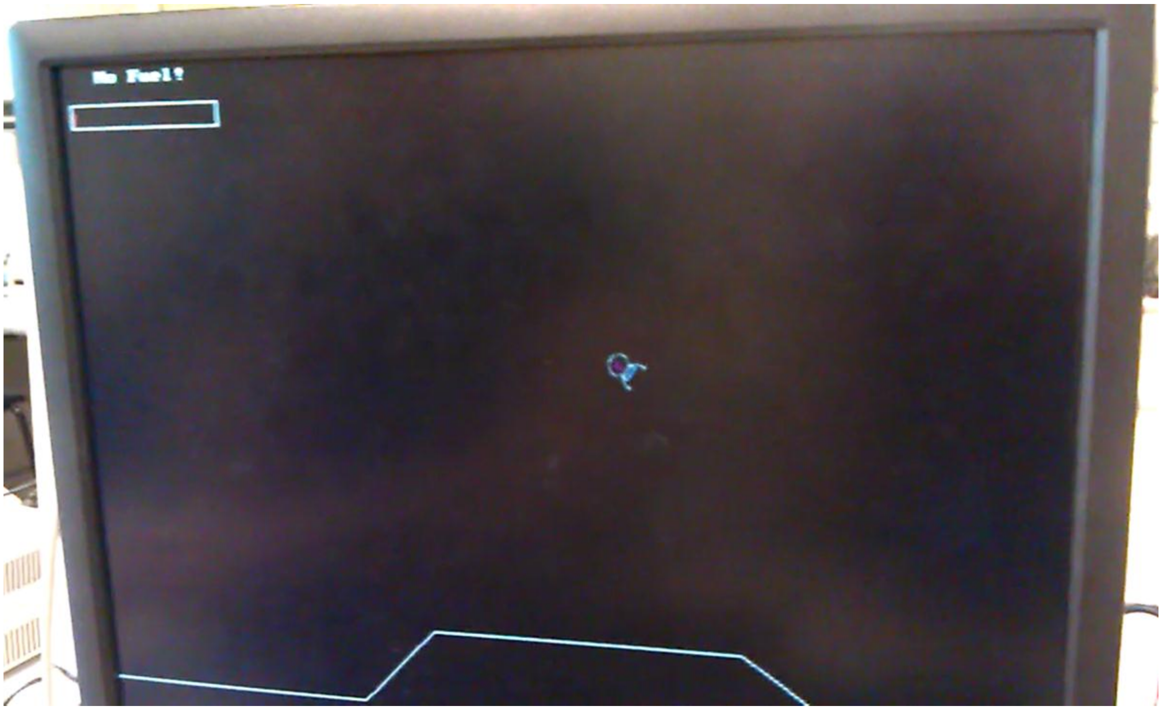

Figure 9: If all fuel is depleted a message is displayed and thrusting is disabled. The lander falls under the direction of gravity.

Figure 10: A crash landing with loser message.

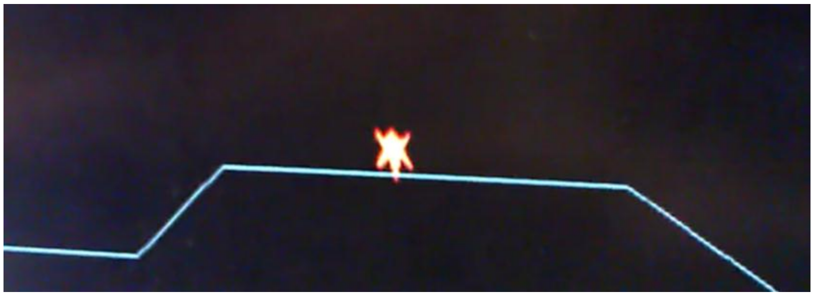

Figure 11: the explosion sprite.

Program Listing

Verilog

```
module DE2_Media_Computer (
    // Inputs
    CLOCK_50,
    CLOCK_27,
    EXT_CLOCK,
    KEY,
    SW,

    //  Communication
    UART_RXD,

    // Audio
    AUD_ADCDAT,

/*****************************************************************************
*/
    // Bidirectionals
    GPIO_0,
    GPIO_1,

    // Memory (SRAM)
    SRAM_DQ,

    // Memory (SDRAM)
    DRAM_DQ,

    // PS2 Port
    PS2_CLK,
    PS2_DAT,
```

```
    // Audio
    AUD_BCLK,
    AUD_ADCLRCK,
    AUD_DACLRCK,

    // Char LCD 16x2
    LCD_DATA,

    // AV Config
    I2C_SDAT,

/*****************************************************************************
*/
    // Outputs
    TD_RESET,

    //  Simple
    LEDG,
    LEDR,

    HEX0,
    HEX1,
    HEX2,
    HEX3,
    HEX4,
    HEX5,
    HEX6,
    HEX7,

    //  Memory (SRAM)
    SRAM_ADDR,

    SRAM_CE_N,
    SRAM_WE_N,
    SRAM_OE_N,
    SRAM_UB_N,
    SRAM_LB_N,

    //  Communication
    UART_TXD,

    // Memory (SDRAM)
    DRAM_ADDR,

    DRAM_BA_1,
    DRAM_BA_0,
    DRAM_CAS_N,
    DRAM_RAS_N,
    DRAM_CLK,
    DRAM_CKE,
    DRAM_CS_N,
    DRAM_WE_N,
    DRAM_UDQM,
    DRAM_LDQM,

    // Audio
    AUD_XCK,
```

```
    AUD_DACDAT,

    // VGA
    VGA_CLK,
    VGA_HS,
    VGA_VS,
    VGA_BLANK,
    VGA_SYNC,
    VGA_R,
    VGA_G,
    VGA_B,

    // Char LCD 16x2
    LCD_ON,
    LCD_BLON,
    LCD_EN,
    LCD_RS,
    LCD_RW,

    // AV Config
    I2C_SCLK,
);

/*****************************************************************************
*
 *                             Parameter Declarations
*

*****************************************************************************
/


/*****************************************************************************
*
 *                               Port Declarations
*

*****************************************************************************
/
// Inputs
input                   CLOCK_50;
input                   CLOCK_27;
input                   EXT_CLOCK;
input           [3:0]   KEY;
input           [17:0]  SW;


//  Communication
input                   UART_RXD;

//  Audio
input                   AUD_ADCDAT;

// Bidirectionals
inout           [35:0]  GPIO_0;
inout           [35:0]  GPIO_1;
```

```
//  Memory (SRAM)
inout       [15:0]  SRAM_DQ;

//  Memory (SDRAM)
inout       [15:0]  DRAM_DQ;

//  PS2 Port
inout               PS2_CLK;
inout               PS2_DAT;

//  Audio
inout               AUD_BCLK;
inout               AUD_ADCLRCK;
inout               AUD_DACLRCK;

//  AV Config
inout               I2C_SDAT;

//  Char LCD 16x2
inout       [ 7: 0] LCD_DATA;

// Outputs
output              TD_RESET;

//  Simple
output      [8:0]   LEDG;
output      [17:0]  LEDR;

output      [6:0]   HEX0;
output      [6:0]   HEX1;
output      [6:0]   HEX2;
output      [6:0]   HEX3;
output      [6:0]   HEX4;
output      [6:0]   HEX5;
output      [6:0]   HEX6;
output      [6:0]   HEX7;

//  Memory (SRAM)
output      [17:0]  SRAM_ADDR;

output              SRAM_CE_N;
output              SRAM_WE_N;
output              SRAM_OE_N;
output              SRAM_UB_N;
output              SRAM_LB_N;

//  Communication
output              UART_TXD;

//  Memory (SDRAM)
output      [11:0]  DRAM_ADDR;

output              DRAM_BA_1;
output              DRAM_BA_0;
output              DRAM_CAS_N;
output              DRAM_RAS_N;
output              DRAM_CLK;
```

```
output              DRAM_CKE;
output              DRAM_CS_N;
output              DRAM_WE_N;
output              DRAM_UDQM;
output              DRAM_LDQM;

//  Audio
output              AUD_XCK;
output              AUD_DACDAT;

//  VGA
output              VGA_CLK;
output              VGA_HS;
output              VGA_VS;
output              VGA_BLANK;
output              VGA_SYNC;
output      [ 9: 0] VGA_R;
output      [ 9: 0] VGA_G;
output      [ 9: 0] VGA_B;

//  Char LCD 16x2
output              LCD_ON;
output              LCD_BLON;
output              LCD_EN;
output              LCD_RS;
output              LCD_RW;

//  AV Config
output              I2C_SCLK;

/*****************************************************************************
*
 *                  Internal Wires and Registers Declarations
*

*****************************************************************************
/
// Internal Wires

// Internal Registers

// State Machine Registers

/*****************************************************************************
*
 *                         Finite State Machine(s)
*

*****************************************************************************
/
 reg[3:0] seconds_ones;
 reg[3:0] seconds_tens;
 reg[3:0] seconds_hundreds;
 reg[3:0] seconds_thousands;

 reg[26:0] counter_ones;
```

```
//every 1 seconds
always@(posedge CLOCK_50 or negedge KEY[0])
begin
    if(!KEY[0])
    begin
        seconds_ones        <=  0;
        seconds_tens        <= 0;
        seconds_hundreds    <= 0;
        seconds_thousands <= 0;
        counter_ones        <=  0;
    end
    else
    begin
        if(counter_ones == 49999999 )
        begin
            seconds_ones    <=  seconds_ones+1;
           counter_ones <=  0;
        end

        else
            counter_ones    <=  counter_ones+1;


        if (seconds_ones == 10)
        begin
            seconds_ones <= 0;
            seconds_tens <= seconds_tens+1;
        end

        if (seconds_tens == 10)
        begin
            seconds_tens <=0;
            seconds_hundreds <= seconds_hundreds+1;
        end

        if (seconds_hundreds == 10)
        begin
            seconds_hundreds <= 0;
            seconds_thousands <= seconds_thousands+1;
        end


    end
end


HexDigit H0(HEX0, seconds_ones);
HexDigit H1(HEX1, seconds_tens);
HexDigit H2(HEX2, seconds_hundreds);
HexDigit H3(HEX3, seconds_thousands);

assign HEX4 = 7'h7F;
assign HEX5 = 7'h7F;
assign HEX6 = 7'h7F;
assign HEX7 = 7'h7F;
```

```
/*****************************************************************************
*
 *                              Sequential Logic
*

*****************************************************************************
/


/*****************************************************************************
*
 *                             Combinational Logic
*

*****************************************************************************
/

// Output Assignments
assign TD_RESET     = 1'b1;
assign GPIO_0[ 0]   = 1'bZ;
assign GPIO_0[ 2]   = 1'bZ;
assign GPIO_0[16]   = 1'bZ;
assign GPIO_0[18]   = 1'bZ;
assign GPIO_1[ 0]   = 1'bZ;
assign GPIO_1[ 2]   = 1'bZ;
assign GPIO_1[16]   = 1'bZ;
assign GPIO_1[18]   = 1'bZ;

/*****************************************************************************
*
 *                               Internal Modules
*

*****************************************************************************
/
nios_system NiosII (
    // 1) global signals:
    .clk                                    (CLOCK_50),
    .clk_27                             (CLOCK_27),
    .reset_n                                (SW[0]),
    .sys_clk                                (),
    .vga_clk                                (),
    .sdram_clk                              (DRAM_CLK),
    .audio_clk                              (AUD_XCK),

    // the_AV_Config
    .I2C_SDAT_to_and_from_the_AV_Config     (I2C_SDAT),
    .I2C_SCLK_from_the_AV_Config            (I2C_SCLK),

    // the_Audio
    .AUD_ADCDAT_to_the_Audio                (AUD_ADCDAT),
    .AUD_BCLK_to_and_from_the_Audio         (AUD_BCLK),
    .AUD_ADCLRCK_to_and_from_the_Audio      (AUD_ADCLRCK),
    .AUD_DACLRCK_to_and_from_the_Audio      (AUD_DACLRCK),
    .AUD_DACDAT_from_the_Audio              (AUD_DACDAT),
```

```
    // the_Char_LCD_16x2
    .LCD_DATA_to_and_from_the_Char_LCD_16x2 (LCD_DATA),
    .LCD_ON_from_the_Char_LCD_16x2          (LCD_ON),
    .LCD_BLON_from_the_Char_LCD_16x2        (LCD_BLON),
    .LCD_EN_from_the_Char_LCD_16x2          (LCD_EN),
    .LCD_RS_from_the_Char_LCD_16x2          (LCD_RS),
    .LCD_RW_from_the_Char_LCD_16x2          (LCD_RW),

    // the_Expansion_JP1
    .GPIO_0_to_and_from_the_Expansion_JP1   ({GPIO_0[35:19], GPIO_0[17],
GPIO_0[15:3], GPIO_0[1]}),

    // the_Expansion_JP2
    .GPIO_1_to_and_from_the_Expansion_JP2   ({GPIO_1[35:19], GPIO_1[17],
GPIO_1[15:3], GPIO_1[1]}),

    // the_Green_LEDs
    .LEDG_from_the_Green_LEDs               (LEDG),

    // the_HEX3_HEX0
    //.HEX0_from_the_HEX3_HEX0              (HEX0),
    //.HEX1_from_the_HEX3_HEX0              (HEX1),
    //.HEX2_from_the_HEX3_HEX0              (HEX2),
    //.HEX3_from_the_HEX3_HEX0              (HEX3),

    // the_HEX7_HEX4
    //.HEX4_from_the_HEX7_HEX4              (HEX4),
    //.HEX5_from_the_HEX7_HEX4              (HEX5),
    //.HEX6_from_the_HEX7_HEX4              (HEX6),
    //.HEX7_from_the_HEX7_HEX4              (HEX7),

    // the_PS2_Port
    .PS2_CLK_to_and_from_the_PS2_Port       (PS2_CLK),
    .PS2_DAT_to_and_from_the_PS2_Port       (PS2_DAT),

    // the_Pushbuttons
    .KEY_to_the_Pushbuttons                 (KEY[3:0]),

    // the_Red_LEDs
    .LEDR_from_the_Red_LEDs                 (LEDR),

    // the_SDRAM
    .zs_addr_from_the_SDRAM                 (DRAM_ADDR),
    .zs_ba_from_the_SDRAM                   ({DRAM_BA_1, DRAM_BA_0}),
    .zs_cas_n_from_the_SDRAM                (DRAM_CAS_N),
    .zs_cke_from_the_SDRAM                  (DRAM_CKE),
    .zs_cs_n_from_the_SDRAM                 (DRAM_CS_N),
    .zs_dq_to_and_from_the_SDRAM            (DRAM_DQ),
    .zs_dqm_from_the_SDRAM                  ({DRAM_UDQM, DRAM_LDQM}),
    .zs_ras_n_from_the_SDRAM                (DRAM_RAS_N),
    .zs_we_n_from_the_SDRAM                 (DRAM_WE_N),

    // the_SRAM
    .SRAM_DQ_to_and_from_the_SRAM           (SRAM_DQ),
    .SRAM_ADDR_from_the_SRAM                (SRAM_ADDR),
    .SRAM_LB_N_from_the_SRAM                (SRAM_LB_N),
    .SRAM_UB_N_from_the_SRAM                (SRAM_UB_N),
```

```
    .SRAM_CE_N_from_the_SRAM                (SRAM_CE_N),
    .SRAM_OE_N_from_the_SRAM                (SRAM_OE_N),
    .SRAM_WE_N_from_the_SRAM                (SRAM_WE_N),

    // the_Serial_port
    .UART_RXD_to_the_Serial_Port            (UART_RXD),
    .UART_TXD_from_the_Serial_Port          (UART_TXD),

    // the_Slider_switches
    .SW_to_the_Slider_Switches              (SW),

    // the_VGA_Controller
    .VGA_CLK_from_the_VGA_Controller        (VGA_CLK),
    .VGA_HS_from_the_VGA_Controller         (VGA_HS),
    .VGA_VS_from_the_VGA_Controller         (VGA_VS),
    .VGA_BLANK_from_the_VGA_Controller      (VGA_BLANK),
    .VGA_SYNC_from_the_VGA_Controller       (VGA_SYNC),
    .VGA_R_from_the_VGA_Controller          (VGA_R),
    .VGA_G_from_the_VGA_Controller          (VGA_G),
    .VGA_B_from_the_VGA_Controller          (VGA_B)
);

endmodule

///////////////////////////////////////////////
// Decode one hex digit for LED 7-seg display
module HexDigit(segs, num);
    input [3:0] num ;       //the hex digit to be displayed
    output [6:0] segs ;     //actual LED segments
    reg [6:0] segs ;
    always @ (num)
    begin
        case (num)
                4'h0: segs = 7'b1000000;
                4'h1: segs = 7'b1111001;
                4'h2: segs = 7'b0100100;
                4'h3: segs = 7'b0110000;
                4'h4: segs = 7'b0011001;
                4'h5: segs = 7'b0010010;
                4'h6: segs = 7'b0000010;
                4'h7: segs = 7'b1111000;
                4'h8: segs = 7'b0000000;
                4'h9: segs = 7'b0010000;
                default segs = 7'b1111111;
        endcase
    end
endmodule
///////////////////////////////////////////////
```

Program (C)

```
#include "altera_up_avalon_video_pixel_buffer_dma.h"
#include "altera_up_avalon_video_character_buffer_with_dma.h"
#include "sys/alt_stdio.h"
#include "sprites.h"
```

```c
#include "explosion.h"
#include <math.h>

#define PI 3.14159265

/*Landscape Coordinates */
int x_land_0 = 0;
int x_land_1 = 200;
int x_land_2 = 250;
int x_land_3 = 450;
int x_land_f = 640;
int y_land_1 = 350;
int y_land_2 = 400;
int y_land_f = 480;

void draw_landscape(alt_up_pixel_buffer_dma_dev *);
void draw_sprite(alt_up_pixel_buffer_dma_dev *, unsigned int x, unsigned int
y, unsigned int theta, int thrusting, int stillAlive);
void draw_fuel_guage(alt_up_pixel_buffer_dma_dev *, float fuel);

char you_died[40] = "You died!\0";
char you_won[40] = "Perfect Landing!\0";
char no_fuel[40] = "No Fuel!\0";

/*****************************************************************************
****
 * This program demonstrates use of the character and pixel buffer HAL code
for
 * the DE2 Media computer. It:
 *  -- places a blue box on the VGA display, and places a text string inside
the box.
 *      -- draws a big A on the screen, for ALTERA
 *      -- "bounces" a colored box around the screen
******************************************************************************
***/
int main(void){
    alt_up_pixel_buffer_dma_dev *pixel_buffer_dev;
    alt_up_char_buffer_dev *char_buffer_dev;

    /* used for drawing coordinates */
    float x1, y1, x2, y2, deltax, deltay, delay = 0.0;
    float fuel = 100.0;
    float g = 0.02;
    float thrust = 0.08;
    float theta = 18.0;
    float turning = 0.3;

    float initial_x_velocity = -2.0;

    int thrusting = 0;

    int stillAlive = 1; //I'm doing science and I'm still alive.


    int initx = 600;
    int inity = 10;
```

```
    int flat_area_1 = 0;
    int flat_area_2 = 0;

    char hasFuel = 1;

    /* initialize the pixel buffer HAL */
    pixel_buffer_dev = alt_up_pixel_buffer_dma_open_dev
("/dev/VGA_Pixel_Buffer");
    if ( pixel_buffer_dev == NULL)
        alt_printf ("Error: could not open VGA pixel buffer device\n");
    else
        alt_printf ("Opened character VGA pixel buffer device\n");
    /* clear the graphics screen */
    alt_up_pixel_buffer_dma_clear_screen(pixel_buffer_dev, 0);

    /* output text message in the middle of the VGA monitor */
    char_buffer_dev = alt_up_char_buffer_open_dev ("/dev/VGA_Char_Buffer");
    if (char_buffer_dev == NULL)
    {
        alt_printf ("Error: could not open character buffer device\n");
        return -1;
    }
    else
        alt_printf ("Opened character buffer device\n");
    //alt_up_char_buffer_string (char_buffer_dev, text_top_row, 35, 29);
    //alt_up_char_buffer_string (char_buffer_dev, text_bottom_row, 35, 30);

    /* now draw a background box for the text */


    /* now draw the landscape */
    draw_landscape (pixel_buffer_dev);

    /*Draw the full fuel guage */
    draw_fuel_guage(pixel_buffer_dev, fuel);

    x1 = initx;
    y1 = inity;
    x2 = initx + 30;
    y2 = inity + 30;
    alt_up_pixel_buffer_dma_draw_box(pixel_buffer_dev, x1, y1, x2, y2, 0xff,
0);
    //alt_up_pixel_buffer_dma_draw_rectangle(pixel_buffer_dev, x1, y1, x2,
y2, 0xF800, 0);
    //alt_up_pixel_buffer_dma_draw_line(pixel_buffer_dev, x1, y1, x2, y2,
0x07e0, 0);
    //alt_up_pixel_buffer_dma_draw_line(pixel_buffer_dev, x1, y2, x2, y1,
0x07e0, 0);
    //alt_up_pixel_buffer_dma_swap_buffers(pixel_buffer_dev);

    /* set the direction in which the box will move */
    deltax = initial_x_velocity;
    deltay = 0;

    float slope_b = ((float)y_land_2-(float)y_land_1)/((float)x_land_1-
(float)x_land_2);
```

```
    float slope_d = ((float)y_land_f-(float)y_land_1)/((float)x_land_f-
(float)x_land_3);

    while(1)
    {

        int * green_leds = (int *) GREEN_LEDS_BASE; /* red_leds is a pointer
to the LEDRs */
        int * pushbuttons = (int *) PUSHBUTTONS_BASE; /* points to
pushbuttons */
        *(green_leds) = *(pushbuttons); /* Green LEDG[k] is set equal to
PB[k] */

        //RESEST PARAMETERS
        if ((*pushbuttons) & 0x01){
            alt_up_pixel_buffer_dma_clear_screen(pixel_buffer_dev, 0);
            stillAlive = 1;
            char you_alive[40] = "                \0";
            alt_up_char_buffer_string (char_buffer_dev, you_alive, 5, 0);
            alt_up_pixel_buffer_dma_draw_box(pixel_buffer_dev, (int)x1,
(int)y1, (int)x2, (int)y2, 0, 0);
            x1 = initx;
            y1 = inity;
            deltax = initial_x_velocity;
            fuel = 100.0;
            deltay = 0.0;
            theta = 18.0;
            turning = 0.3;
            hasFuel = 1;
            draw_fuel_guage(pixel_buffer_dev, fuel);
        }


        if
(alt_up_pixel_buffer_dma_check_swap_buffers_status(pixel_buffer_dev) == 0)
        {
            /* If the screen has been drawn completely then we can draw a new
image. This
             * section of the code will only be entered once every 60th of a
second, because
             * this is how long it take the VGA controller to copy the image
from memory to
             * the screen. */
            delay = 1 - delay;

            if (delay == 0)
            {
                /* The delay is inserted to slow down the animation from 60
frames per second
                 * to 30. Every other refresh cycle the code below will
execute. We first erase
                 * the box with Erase Rectangle */
                alt_up_pixel_buffer_dma_draw_box(pixel_buffer_dev, (int)x1-3,
(int)y1-3, (int)x2+3, (int)y2+3, 0, 0);
                //alt_up_pixel_buffer_dma_draw_line(pixel_buffer_dev, x1, y1,
x2, y2, 0, 0);
```

```
                //alt_up_pixel_buffer_dma_draw_line(pixel_buffer_dev, x1, y2,
x2, y1, 0, 0);
                draw_sprite(pixel_buffer_dev, (int)x1, (int)y1, (int)theta,
thrusting, stillAlive);
                draw_fuel_guage(pixel_buffer_dev, fuel);
                //Right Thruster
                if (stillAlive) {
                    //Main Thruster
                    if ((*pushbuttons) & 0x04)
                    {
                        if(hasFuel){
                            fuel = fuel - 0.65;
                            //draw_fuel_guage(pixel_buffer_dev, fuel);


//alt_up_pixel_buffer_dma_draw_box(pixel_buffer_dev, 0, 0, 50, 50, 3, 0);
                            deltay = (deltay - thrust*cos(((theta*5.0)-
90.0)*PI/180));

                            deltax = (deltax - thrust*sin(((theta*5.0)-
90.0)*PI/180));

                            thrusting = 1;
                        }
                    } else {
                        thrusting = 0;
                    }

                    if ((*pushbuttons) & 0x02)
                    {
                        //alt_up_pixel_buffer_dma_draw_box(pixel_buffer_dev,
0, 0, 50, 50, 3, 0);
                        //deltax = deltax + 1.0;
                        if (theta < (37.0 - turning)) {
                            theta = theta + turning;
                        }
                    }
                    //Left Thruster
                    if ((*pushbuttons) & 0x08)
                    {
                        //alt_up_pixel_buffer_dma_draw_box(pixel_buffer_dev,
0, 0, 50, 50, 3, 0);
                        //deltax = deltax - 1.0;
                        if (theta > (0.0 + turning)) {
                            theta = theta - turning;
                        }
                    }


                } else {
                    deltax = 0.0;
                    deltay = 0.0;
                }

                // move the rectangle
                x1 = x1 + deltax;
                //x2 = x2 + deltax;
                y1 = y1 + deltay;
                //y2 = y2 + deltay;
```

```
                //Dead if you go off right side of the screen
                if ((deltax > 0.0) && (x1 >=
alt_up_pixel_buffer_dma_x_res(pixel_buffer_dev) - 31.0))
                {
                    x1 = alt_up_pixel_buffer_dma_x_res(pixel_buffer_dev) -
31.0;
                    deltax = 0.0;
                    stillAlive = 0;
                }
                //Dead if you go off the left side of the screen
                else if ((deltax < 0.0) && (x1 <= 0.0))
                {
                    x1 = 0.0;
                    deltax = 0.0;
                    stillAlive = 0;
                }

                //Calculate slopes to sprite box to determine collision state
                /* float slope_bottom_left_b  = (((float)y1 + 25.0)-
(float)y_land_2)/(((float)x1 + 5.0 ) - (float)x_land_2);
                float slope_bottom_right_b = (((float)y1 + 25.0)-
(float)y_land_2)/(((float)x1 + 25.0) - (float)x_land_2); */
                float slope_bottom_left_d  = ((float)y_land_f-((float)y1 +
25.0))/((float)x_land_f - ((float)x1 + 5.0));
                float slope_bottom_right_d = ((float)y_land_f-((float)y1 +
25.0))/( (float)x_land_f - ((float)x1 + 25.0));


                float m = ((float)y_land_1 -
(float)y_land_2)/((float)x_land_2 - (float)x_land_1);
                float b = ((float)y_land_2) - ((float)x_land_1)*m;

                flat_area_1 = 0;
                flat_area_2 = 0;

                if (((x1+5.0) > x_land_0) && ((x1 + 25.0) < x_land_1))
                    flat_area_1 = 1;
                else if (((x1 + 5.0) > x_land_2) && ((x1 + 25.0) < x_land_3))
                    flat_area_2 = 1;

                //Did you hit the landscape?
                if (
                    (deltay > 0.0)
                    &&
                    (
                        (
                            //Intersection with Slope a (flat line)
                            (y1 >= y_land_2 - 25.0) && flat_area_1
                        )
                        /* ||
                        (
                            //Intersection with Slope b
                            ((x1+5.0) > x_land_1) && ((x1+25.0) < x_land_2)
&& ( ( abs(slope_bottom_left_b) > abs(slope_b)) ||
(abs(slope_bottom_right_b) < abs(slope_b)) )
                        )  */
```

```
                        ||
                        (
                        //test
                            ((x1+5.0) > x_land_1) && ((x1+25.0) < x_land_2)
&& ((y1+25.0)> m*(x1+25.0)+b)
                        )
                        ||
                        (
                            ((x1+5.0) < x_land_2) && (y1+25.0>y_land_1) &&
((x1+25.0) > x_land_2)
                        )
                        ||
                        (
                            //Intersection with Slope c (flat line)
                            (y1 >= y_land_1 - 25.0) && flat_area_2
                        )
                        ||
                        (
                            //Intersection with slope d
                            ((x1+5.0) > x_land_3) && ((x1+25.0) < x_land_f)
&& ( (slope_bottom_left_d < slope_d) || (slope_bottom_right_d < slope_d) )
                        )

                    )
                   )
                {


                    //Are we in a flat landing area?
                    if ((flat_area_1 || flat_area_2) && deltay < 2.0 &&
deltax < 3.0 && theta == 18)
                    {
                        deltay = 0.0;
                        deltax = 0.0;
                        stillAlive = 2;
                    }
                    else
                    {
                        //y1 =
alt_up_pixel_buffer_dma_y_res(pixel_buffer_dev) - 31.0;
                        deltay = 0.0;
                        stillAlive = 0;
                    }

                }
                //Dead if you go off the top of the screen
                else if ((deltay < 0.0) && (y1 <= 0.0))
                {
                    y1 = 0.0;
                    deltay = -deltay;
                    stillAlive = 0;
                }

                //Dead if you run out of fuel
                else if (fuel <= 0.0)
                {
                    hasFuel = 0;
```

```
                    //fuel = 100.0;
                    alt_up_char_buffer_string (char_buffer_dev, no_fuel, 5,
0);
                    thrusting = 0;
                }

                x2 = x1 + 30.0;
                y2 = y1 + 30.0;

                // redraw Rectangle with diagonal lines
                //alt_up_pixel_buffer_dma_draw_rectangle(pixel_buffer_dev,
x1, y1, x2, y2, 0xF800, 0);
                //alt_up_pixel_buffer_dma_draw_line(pixel_buffer_dev, x1, y1,
x2, y2, 0x07e0, 0);
                //alt_up_pixel_buffer_dma_draw_line(pixel_buffer_dev, x1, y2,
x2, y1, 0x07e0, 0);

                //%%%%//alt_up_pixel_buffer_dma_draw_box(pixel_buffer_dev,
x1, y1, x2, y2, 0x60, 0);
                //draw_sprite(pixel_buffer_dev, x1, y1);

                // redraw the box in the foreground
                //%%%%//alt_up_pixel_buffer_dma_draw_box(pixel_buffer_dev,
34*4, 28*4, 50*4, 32*4, 0xf0, 0);

                //draw_sprite(pixel_buffer_dev, (int)x1, (int)y1, (int)theta,
thrusting);
                draw_landscape (pixel_buffer_dev);

                deltay = deltay + g;

                //you exploded!
                if ((stillAlive == 0)||(stillAlive == 2)) {
                    if(stillAlive ==0){
                        alt_up_char_buffer_string (char_buffer_dev, you_died,
5, 0);
                    }else{
                        alt_up_char_buffer_string (char_buffer_dev, you_won,
5, 0);
                    }

                    deltax = 0.0;
                    deltay = 0.0;
                }
                //you won!


            }

            /* Execute a swap buffer command. This will allow us to check if
the screen has
             * been redrawn before generating a new animation frame. */
            alt_up_pixel_buffer_dma_swap_buffers(pixel_buffer_dev);
        }
    }
}
```

```
/* draws a landscape */
void draw_landscape(alt_up_pixel_buffer_dma_dev *pixel_buffer_dev ){
    //Line segment 1: (x_land_0, y_land_2) <-> (x_land_1, y_land_2)
    alt_up_pixel_buffer_dma_draw_line(pixel_buffer_dev, x_land_0, y_land_2,
x_land_1, y_land_2, 0xffff, 0);

    //Line segment 2: (x_land_1, y_land_2) <-> (x_land_2, y_land_1)
    alt_up_pixel_buffer_dma_draw_line(pixel_buffer_dev, x_land_1, y_land_2,
x_land_2, y_land_1, 0xffff, 0);

    //Line segment 3: (x_land_2, y_land_1) <-> (x_land_3, y_land_1)
    alt_up_pixel_buffer_dma_draw_line(pixel_buffer_dev, x_land_2, y_land_1,
x_land_3, y_land_1, 0xffff, 0);

    //Line Segment 4: (x_land_3, y_land_1) <-> (x_land_f, y_land_f)
    alt_up_pixel_buffer_dma_draw_line(pixel_buffer_dev, x_land_3, y_land_1,
x_land_f, y_land_f, 0xffff, 0);

}

int groundCollision(unsigned int x, unsigned int y) {
    if (1) {
        return 0;
    }
}


void draw_sprite(alt_up_pixel_buffer_dma_dev *pixel_buffer_dev , unsigned int
x, unsigned int y, unsigned int theta, int thrusting, int stillAlive) {
    int i = 0;
    int j = 0;
    for(i = 0; i < 30; i++) {
        for(j = 0; j < 30; j++) {
            char color = landers[theta][i][j];
            if(!stillAlive){
                color = explode[i][j];
            }
            //char color = landers[theta][i][j];
            if (color) {
                if ((!thrusting) && (color > (char)0) && (color < (char)5))
                {
                    color = 0;
                }
                alt_up_pixel_buffer_dma_draw(pixel_buffer_dev, (int)((color
<< 8) + color), x+j, y+i );
            }
        }
    }
}

void draw_fuel_guage(alt_up_pixel_buffer_dma_dev *pixel_buffer_dev, float
fuel) {
    //Draw the surrounding rectangle
    alt_up_pixel_buffer_dma_draw_rectangle(pixel_buffer_dev, 20, 20, 124, 34,
0xFFFF, 0);
    //Draw the empty fuel guage
```

```
    alt_up_pixel_buffer_dma_draw_box(pixel_buffer_dev, 22+(int)(fuel), 22,
122, 32, 0x0000, 0);

    int color = 0x0000;
    if (fuel > 66.0)
        color = 0x1c1c;
    else if (fuel > 33.0)
        color = 0xfcfc;
    else
        color = 0xe0e0;

        alt_up_pixel_buffer_dma_draw_box(pixel_buffer_dev, 22, 22,
22+(int)fuel, 32, color, 0);

}
```

MATLAB

SPRITE_GENERATOR.M

```
%000 - black
%255 - white
%240 - red
%096 - orange

lander_0_r = [   000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000 000 000 000 000 000 000 000;
                 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000 000 000 000 000 000 000 000;
                 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000 000 000 000 000 000 000 000;
                 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000 000 000 000 000 000 000 000;
                 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000 000 000 000 000 000 000 000;
                 000 000 000 000 000 000 000 000 000 000 000 000 255 255 255
255 255 255 000 000 000 000 000 000 000 000 000 000 000 000;
                 000 000 000 000 000 000 000 000 000 000 000 255 000 000 000
000 000 000 255 000 000 000 000 000 000 000 000 000 000 000;
                 000 000 000 000 000 000 000 000 000 000 255 000 000 000 000
000 000 000 000 255 000 000 000 000 000 000 000 000 000 000;
                 000 000 000 000 000 000 000 000 000 255 000 000 000 000 000
000 000 000 000 000 255 000 000 000 000 000 000 000 000 000;
                 000 000 000 000 000 000 000 000 255 000 000 000 000 000 255
255 000 000 000 000 000 255 000 000 000 000 000 000 000 000;
                 000 000 000 000 000 000 000 000 255 000 000 000 000 255 000
000 255 000 000 000 000 255 000 000 000 000 000 000 000 000;
                 000 000 000 000 000 000 000 000 255 000 000 000 000 255 000
000 000 000 000 000 000 255 000 000 000 000 000 000 000 000;
                 000 000 000 000 000 000 000 000 255 000 000 000 000 255 000
000 000 000 000 000 000 255 000 000 000 000 000 000 000 000;
                 000 000 000 000 000 000 000 000 255 000 000 000 000 255 000
000 255 000 000 000 000 255 000 000 000 000 000 000 000 000;
                 000 000 000 000 000 000 000 000 000 255 000 000 000 000 255
255 000 000 000 000 255 000 000 000 000 000 000 000 000 000;
                 000 000 000 000 000 000 000 000 000 000 255 000 000 000 000
000 000 000 000 255 000 000 000 000 000 000 000 000 000 000;
                 000 000 000 000 000 000 000 000 000 000 000 255 000 000 000
000 000 000 255 000 000 000 000 000 000 000 000 000 000 000;
                 000 000 000 000 000 000 000 000 000 000 255 255 255 255 255
255 255 255 255 255 000 000 000 000 000 000 000 000 000 000;
                 000 000 000 000 000 000 000 000 000 255 255 000 000 000 000
000 000 000 000 255 255 000 000 000 000 000 000 000 000 000;
                 000 000 000 000 000 000 000 000 255 000 255 000 000 000 000
000 000 000 000 255 000 255 000 000 000 000 000 000 000 000;
                 000 000 000 000 000 000 000 000 255 000 255 255 255 255 255
255 255 255 255 255 000 255 000 000 000 000 000 000 000 000;
                 000 000 000 000 000 000 000 255 000 000 000 000 000 000 000
000 000 000 000 000 000 000 255 000 000 000 000 000 000 000;
                 000 000 000 000 000 000 000 255 000 000 000 000 000 000 000
000 000 000 000 000 000 000 255 000 000 000 000 000 000 000;
```

```
                000 000 000 000 000 000 000 255 000 000 000 000 000 000 000
000 000 000 000 000 000 000 255 000 000 000 000 000 000 000;
                000 000 000 000 000 000 255 255 255 000 000 000 000 000 000
000 000 000 000 000 000 255 255 255 000 000 000 000 000 000;
                000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000 000 000 000 000 000 000 000;
                000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000 000 000 000 000 000 000 000;
                000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000 000 000 000 000 000 000 000;
                000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000 000 000 000 000 000 000 000;
                000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 ];

lander_0_g = [   000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000 000 000 000 000 000 000 000;
                000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000 000 000 000 000 000 000 000;
                000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000 000 000 000 000 000 000 000;
                000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000 000 000 000 000 000 000 000;
                000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000 000 000 000 000 000 000 000;
                000 000 000 000 000 000 000 000 000 000 000 000 255 255 255
255 255 255 000 000 000 000 000 000 000 000 000 000 000 000;
                000 000 000 000 000 000 000 000 000 000 000 255 000 000 000
000 000 000 255 000 000 000 000 000 000 000 000 000 000 000;
                000 000 000 000 000 000 000 000 000 000 255 000 000 000 000
000 000 000 000 255 000 000 000 000 000 000 000 000 000 000;
                000 000 000 000 000 000 000 000 000 255 000 000 000 000 000
000 000 000 000 000 255 000 000 000 000 000 000 000 000 000;
                000 000 000 000 000 000 000 000 255 000 000 000 000 000 000
000 000 000 000 000 000 255 000 000 000 000 000 000 000 000;
                000 000 000 000 000 000 000 000 255 000 000 000 000 000 000
000 000 000 000 000 000 255 000 000 000 000 000 000 000 000;
                000 000 000 000 000 000 000 000 255 000 000 000 000 000 000
000 000 000 000 000 000 255 000 000 000 000 000 000 000 000;
                000 000 000 000 000 000 000 000 255 000 000 000 000 000 000
000 000 000 000 000 000 255 000 000 000 000 000 000 000 000;
                000 000 000 000 000 000 000 000 255 000 000 000 000 000 000
000 000 000 000 000 000 255 000 000 000 000 000 000 000 000;
                000 000 000 000 000 000 000 000 000 255 000 000 000 000 000
000 000 000 000 000 255 000 000 000 000 000 000 000 000 000;
                000 000 000 000 000 000 000 000 000 000 255 000 000 000 000
000 000 000 000 255 000 000 000 000 000 000 000 000 000 000;
                000 000 000 000 000 000 000 000 000 000 000 255 000 000 000
000 000 000 255 000 000 000 000 000 000 000 000 000 000 000;
                000 000 000 000 000 000 000 000 000 000 255 255 255 255 255
255 255 255 255 255 000 000 000 000 000 000 000 000 000 000;
                000 000 000 000 000 000 000 000 000 255 255 000 000 000 000
000 000 000 000 255 255 000 000 000 000 000 000 000 000 000;
                000 000 000 000 000 000 000 000 255 000 255 000 000 000 000
000 000 000 000 255 000 255 000 000 000 000 000 000 000 000;
                000 000 000 000 000 000 000 000 255 000 255 255 255 255 255
255 255 255 255 255 000 255 000 000 000 000 000 000 000 000;
```

```
                000 000 000 000 000 000 000 255 000 000 000 000 000 000 000
000 000 000 000 000 000 000 255 000 000 000 000 000 000 000;
                000 000 000 000 000 000 000 255 000 000 000 000 000 000 000
000 000 000 000 000 000 000 255 000 000 000 000 000 000 000;
                000 000 000 000 000 000 000 255 000 000 000 000 000 000 000
000 000 000 000 000 000 000 255 000 000 000 000 000 000 000;
                000 000 000 000 000 000 255 255 255 000 000 000 000 000 000
000 000 000 000 000 000 255 255 255 000 000 000 000 000 000;
                000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000 000 000 000 000 000 000 000;
                000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000 000 000 000 000 000 000 000;
                000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000 000 000 000 000 000 000 000;
                000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000 000 000 000 000 000 000 000;
                000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 ];

lander_0_b =  [000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000 000 000 000 000 000 000 000;
                000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000 000 000 000 000 000 000 000;
                000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000 000 000 000 000 000 000 000;
                000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000 000 000 000 000 000 000 000;
                000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000 000 000 000 000 000 000 000;
                000 000 000 000 000 000 000 000 000 000 000 000 255 255 255
255 255 255 000 000 000 000 000 000 000 000 000 000 000 000;
                000 000 000 000 000 000 000 000 000 000 000 255 000 000 000
000 000 000 255 000 000 000 000 000 000 000 000 000 000 000;
                000 000 000 000 000 000 000 000 000 000 255 000 000 000 000
000 000 000 000 255 000 000 000 000 000 000 000 000 000 000;
                000 000 000 000 000 000 000 000 000 255 000 000 000 000 000
000 000 000 000 000 255 000 000 000 000 000 000 000 000 000;
                000 000 000 000 000 000 000 000 255 000 000 000 000 000 255
255 000 000 000 000 000 255 000 000 000 000 000 000 000 000;
                000 000 000 000 000 000 000 000 255 000 000 000 000 255 000
000 255 000 000 000 000 255 000 000 000 000 000 000 000 000;
                000 000 000 000 000 000 000 000 255 000 000 000 000 255 000
000 000 000 000 000 000 255 000 000 000 000 000 000 000 000;
                000 000 000 000 000 000 000 000 255 000 000 000 000 255 000
000 000 000 000 000 000 255 000 000 000 000 000 000 000 000;
                000 000 000 000 000 000 000 000 255 000 000 000 000 255 000
000 255 000 000 000 000 255 000 000 000 000 000 000 000 000;
                000 000 000 000 000 000 000 000 000 255 000 000 000 000 255
255 000 000 000 000 255 000 000 000 000 000 000 000 000 000;
                000 000 000 000 000 000 000 000 000 000 255 000 000 000 000
000 000 000 000 255 000 000 000 000 000 000 000 000 000 000;
                000 000 000 000 000 000 000 000 000 000 000 255 000 000 000
000 000 000 255 000 000 000 000 000 000 000 000 000 000 000;
                000 000 000 000 000 000 000 000 000 000 255 255 255 255 255
255 255 255 255 255 000 000 000 000 000 000 000 000 000 000;
                000 000 000 000 000 000 000 000 000 255 255 000 000 000 000
000 000 000 000 255 255 000 000 000 000 000 000 000 000 000;
```

```
                000 000 000 000 000 000 000 000 255 000 255 000 000 000 000
000 000 000 000 255 000 255 000 000 000 000 000 000 000 000;
                000 000 000 000 000 000 000 000 255 000 255 255 255 255 255
255 255 255 255 255 000 255 000 000 000 000 000 000 000 000;
                000 000 000 000 000 000 000 255 000 000 000 255 255 255 255
255 255 255 255 000 000 000 255 000 000 000 000 000 000 000;
                000 000 000 000 000 000 000 255 000 000 000 000 255 255 255
255 255 255 000 000 000 000 255 000 000 000 000 000 000 000;
                000 000 000 000 000 000 000 255 000 000 000 000 000 255 255
255 255 000 000 000 000 000 255 000 000 000 000 000 000 000;
                000 000 000 000 000 000 255 255 255 000 000 000 000 000 255
255 000 000 000 000 000 255 255 255 000 000 000 000 000 000;
                000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000 000 000 000 000 000 000 000;
                000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000 000 000 000 000 000 000 000;
                000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000 000 000 000 000 000 000 000;
                000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000 000 000 000 000 000 000 000;
                000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 ];
%Lander Sprites
lander_0_r  = uint8(lander_0_r);
lander_0_g  = uint8(lander_0_g);
lander_0_b  = uint8(lander_0_b);

lander_0 = uint8(convert24to8(lander_0_r, lander_0_g, lander_0_b));

for x = 5:5:90
    lander_p_r{x} = imrotate(lander_0_r, x, 'bilinear', 'crop');
    lander_p_g{x} = imrotate(lander_0_g, x, 'bilinear', 'crop');
    lander_p_b{x} = imrotate(lander_0_b, x, 'bilinear', 'crop');
    lander_p{x} = convert24to8(lander_p_r{x}, lander_p_g{x}, lander_p_b{x});
end

for x = 5:5:90
    lander_n_r{x} = imrotate(lander_0_r, -x, 'bilinear', 'crop');
    lander_n_g{x} = imrotate(lander_0_g, -x, 'bilinear', 'crop');
    lander_n_b{x} = imrotate(lander_0_b, -x, 'bilinear', 'crop');
    lander_n{x} = uint8(convert24to8(lander_n_r{x}, lander_n_g{x},
lander_n_b{x}));
end

%Save this into a file that can be imported into the c code as a header
%file.

fileID = fopen('sprites.h','w');


%Make 3D array of all the 2D rotation options.
fprintf(fileID, '//3D Array of all Lander Rotations (-90->90)\n');

    fprintf(fileID, 'char landers[37][30][30]={\n');
    %populate the negative rotations

    %print -90 to -5 Lander Sprites
```

```
for a=90:-5:5
    fprintf(fileID, '{');
    for x = 1:30
        if x == 1
            fprintf(fileID,'{');
        else
            fprintf(fileID,' {');
        end
        fprintf(fileID,'%3d, ', lander_n{a}(x,1:end-1));
        if x ~= length(lander_n{a})
            fprintf(fileID,'%3d},\n', lander_n{a}(x,end));
        else
            fprintf(fileID, '%3d}', lander_n{a}(x,end));
        end
    end
    fprintf(fileID, '},\n\n');
end

%Print 0 Lander Sprite
fprintf(fileID, '{');
for x = 1:30
    if x == 1
        fprintf(fileID,'{');
    else
        fprintf(fileID,' {');
    end
    fprintf(fileID,'%3d, ', lander_0(x,1:end-1));
    if x ~= length(lander_0)
        fprintf(fileID,'%3d},\n', lander_0(x,end));
    else
        fprintf(fileID, '%3d}', lander_0(x,end));
    end
end
fprintf(fileID, '},\n\n');

%print 5 to 90 Lander Sprites
for a=5:5:90
    fprintf(fileID, '{');
    for x = 1:30
        if x == 1
            fprintf(fileID,'{');
        else
            fprintf(fileID,' {');
        end
        fprintf(fileID,'%3d, ', lander_p{a}(x,1:end-1));
        if x ~= length(lander_p{a})
            fprintf(fileID,'%3d},\n', lander_p{a}(x,end));
        else
            fprintf(fileID, '%3d}', lander_p{a}(x,end));
        end
    end
    if a ~= 90
        fprintf(fileID, '},\n\n');
    else
        fprintf(fileID, '}');
    end
end
```

```
    fprintf(fileID, '};\n\n');



fclose(fileID);
```

CONVERT24TO8.M

```
function [out] = convert24to8(R, G, B)
    out = R;
    [h,w] = size(R);
    for x = 1:h
        for y = 1:w
            rval = round(R(x,y) * (7/255));
            gval = round(G(x,y) * (7/255));
            bval = round(B(x,y) * (3/255));
            outval = [dec2bin(rval, 3), dec2bin(gval, 3), dec2bin(bval, 2)];
            out(x,y) = bin2dec(outval);
        end
    end
end
```