Scorched “Earf” XESS

A Term Project
Submitted In Partial Fulfillment
Of

W4840 Embedded Systems Design
Spring 2005

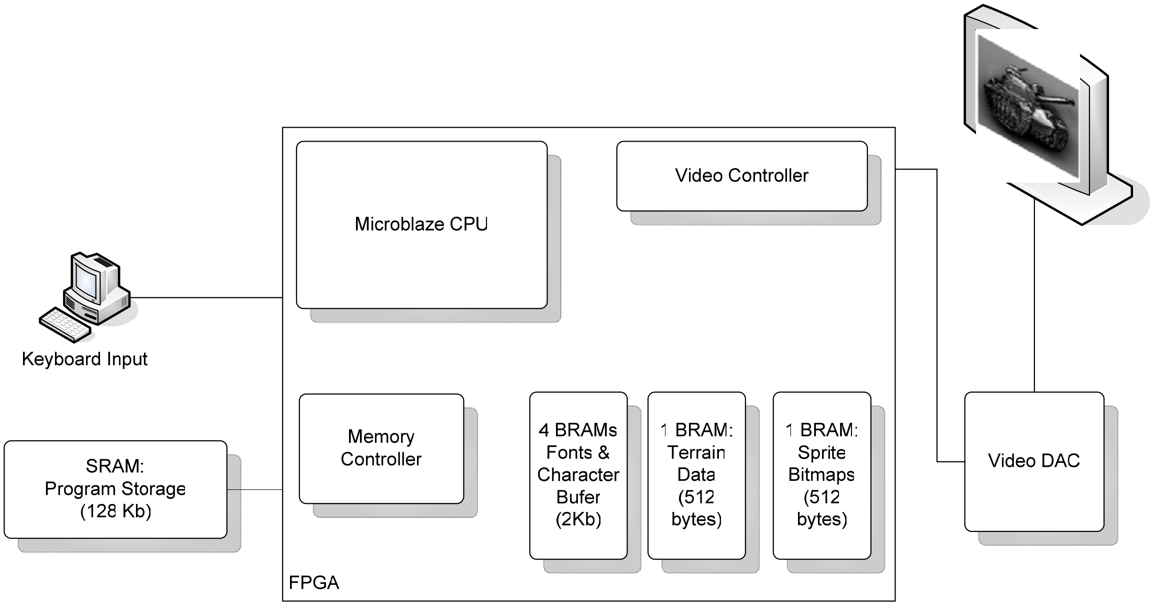
Jeremy Chou
Michael **Brian** Sumulong
Dennis M. Chua
{jc2465, mbs2114, dc2036}@columbia.edu

Introduction 3
Game Interface 4
Summary of Project Development 5
Scorched "Earf" XESS Components 6
Game Logic 6
Video Controller 8
Character Font Controller 8
Memory Controller 10
Sprite Generation 10
Gradient Background 12
Terrain Generation 13
Lessons Learned & Advice for Future Students 16
Jeremy Chou 16
Michael Sumulong 16
Dennis M. Chua 17
Relevant Source Code 19
Appendix 59

Introduction

Scorched Earth is a DOS console game belonging to the artillery game genre. It is a turn-based game for two players in which each controls a tank and attempts to destroy the other with ballistic projectiles. By setting the firing angle and the initial velocity of the projectile, the players alternately launch artillery at each other. Random weather, plus the irregular terrain of the game field adds challenge to the game.

In this project, we took what we learned in **W4840** to implement *Scorched Earth* on the XESS XSB-300E board. Working over a month and a half, we developed and tested VHDL and C code to manipulate the VGA display component of the board. Being a small team of three, we accomplished a fair amount of coding, testing and debugging on the XESS board. This report documents our experience and explains how we worked to make this project a success.



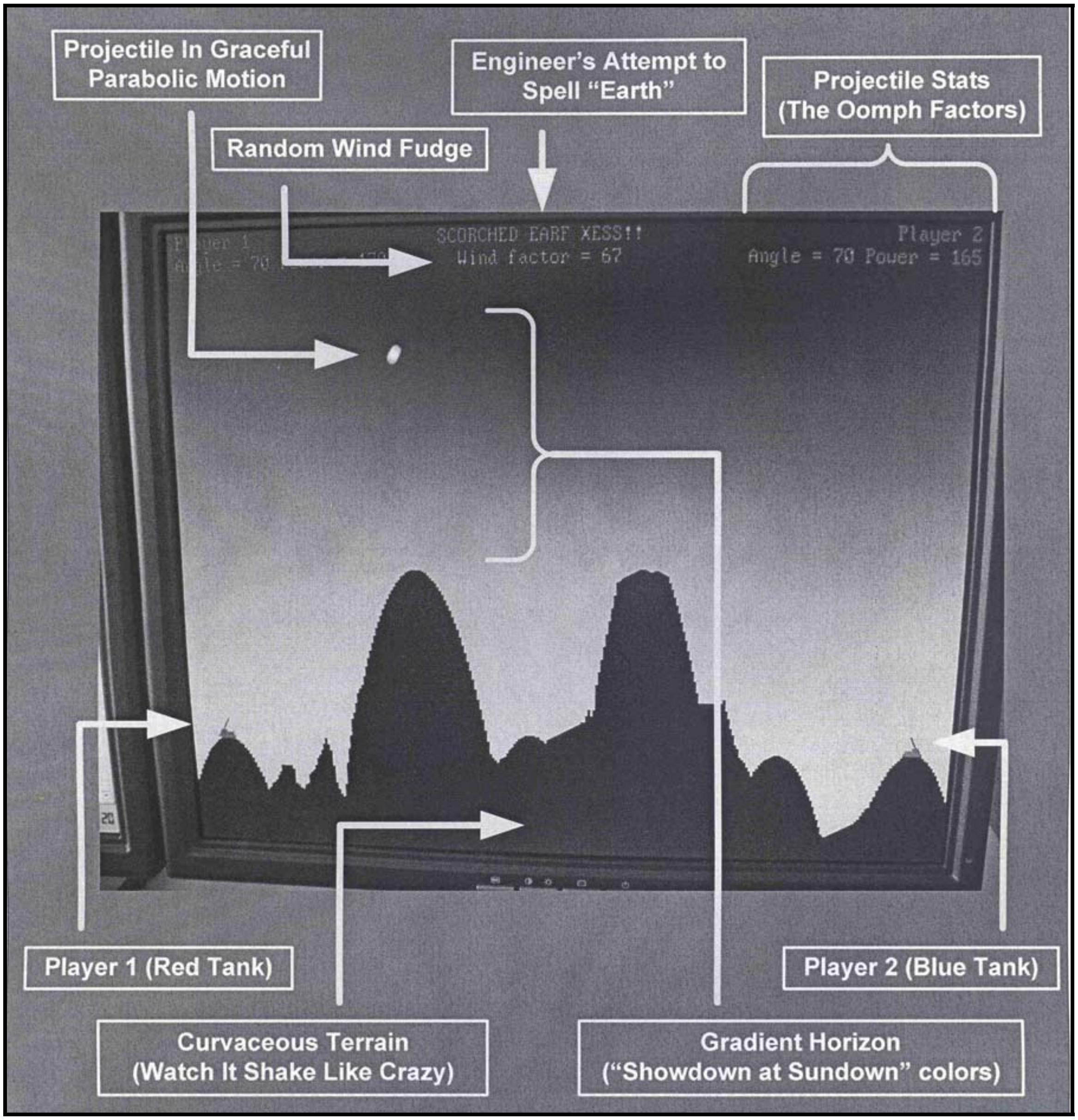
Links to Prof. Stephen Edward’s *W4840 Embedded Systems Design Spring 2005* class:

`http://www1.cs.columbia.edu/~sedwards/classes/2005/4840/index.html`

Links to the Scorched Earth DOS game:

```
http://www.mobygames.com/game/shots/gameId,402/
http://www.abandonia.com/game.php?ID=70
http://www.classicgaming.com/rotw/scorch.shtml
http://www.download-free-games.com/war_game_download/scorched_Earth.htm
```

Game Interface


Projectile In Graceful Parabolic Motion
Engineer's Attempt to Spell "Earth"
Projectile Stats (The Oomph Factors)
Random Wind Fudge
SCORCHED EARF XESS!!
Wind factor = 67
Player 1
Angle = 70
Player 2
Angle = 70 Power = 165
Player 1 (Red Tank)
Player 2 (Blue Tank)
Curvaceous Terrain (Watch It Shake Like Crazy)
Gradient Horizon ("Showdown at Sundown" colors)

Controls for the Red Tank

Press A to increase angle
Press D to decrease angle
Press W to increase power
Press S to decrease power

Controls for the Blue Tank

Press L to increase angle
Press J to decrease angle
Press I to increase power
Press K to decrease power

Press SPACEBAR to fire cannonball
Press TAB to turn wind factor ON/OFF

Summary of Project Development

The development of the project was split into two distinct phases. In the initial phase, we developed a prototype of the game based mainly on software, using lab 5 - 2004 as a base for our code.

Lab 5 - 2004 had all the elements to develop a prototype game relatively quickly. It included components such as a video buffer and an SRAM memory controller. The video buffer allowed us to control the color of every single pixel through software. The SRAM memory controller enabled us to develop large C programs without worrying about the constraints of the limited BRAM available. Eventually, we merged part of lab 2 - 2005 code into the lab 5 code to get text to display on the screen. We completed the initial phase by demonstrating the prototype game play during our 75% demo.

The second phase required us to develop almost all graphics in hardware while keeping the game logic and calculations in software.

We initially used lab 2 - 2005 as a base for our code but later realized that the compiled C code would not fit on the 4096 bits of BRAM that was available to us. This was due to the use of floating point numbers in some of our calculations.

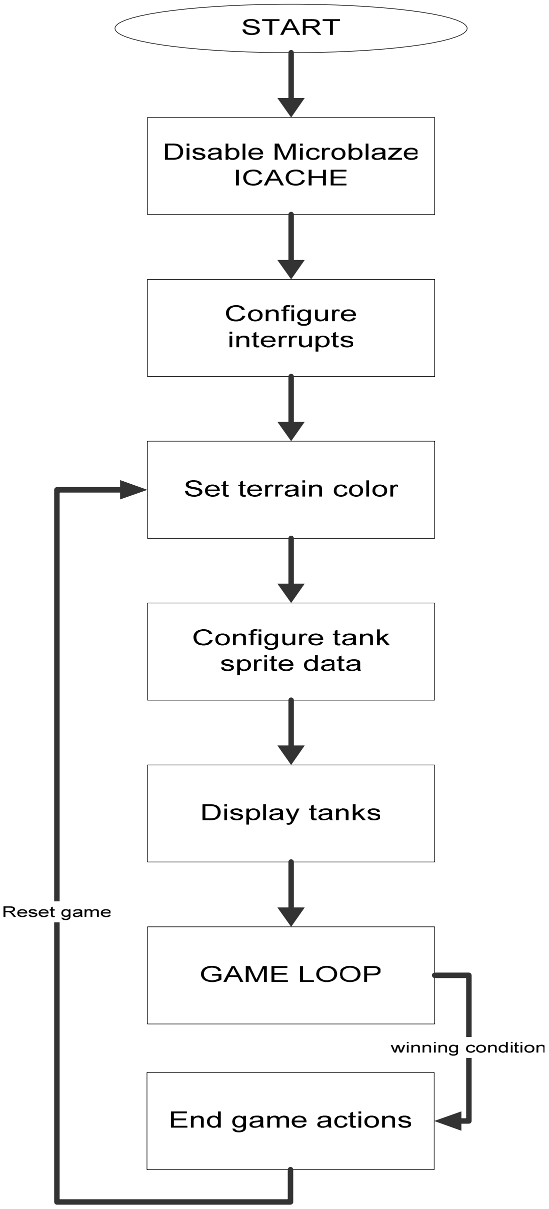
We had to make a decision between not using floating point numbers or finding a larger memory space for our program and we chose the latter. Lab 5 - 2004 already had the functionality to allow the program to be stored in the SRAM and therefore seemed to be the more feasible choice as 4KB of memory gives us very little elbow room.

There was a slight problem with the number of BRAMs being used initially in lab 5 - 2005 as we kept getting over the 16 BRAM limit. After talking to Marcio and Prof. Edwards, we found out that there is an instruction cache used by the Microblaze processor to speed up execution. This cache took up 5 BRAMS and we immediately disabled them by going into the .mhs file. This roadblock was surpassed and we were able to put in the terrain data, sprite bitmaps, font bitmaps, and character buffer in the free BRAMs.

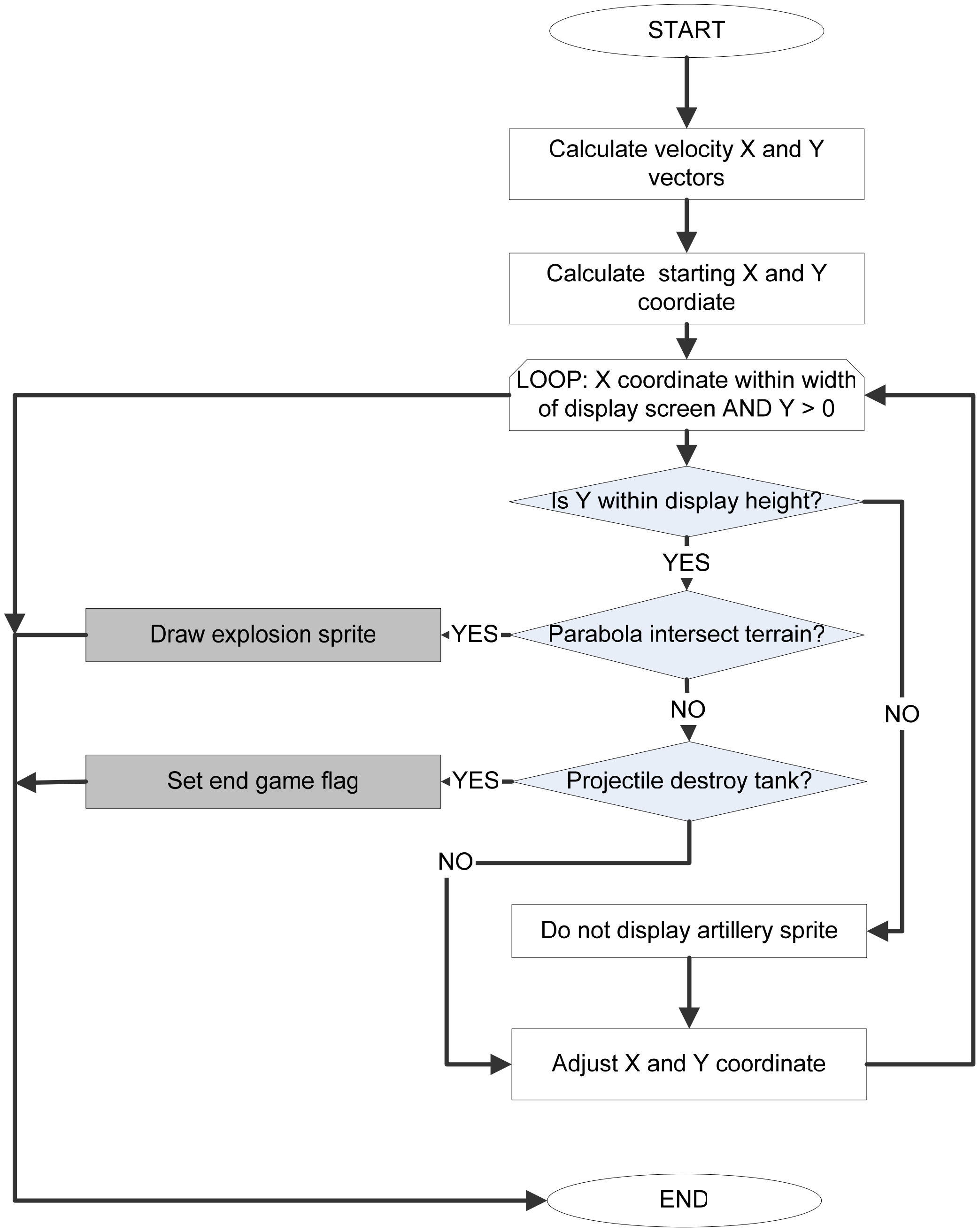
Scorched "Earf" XESS Components

Game Logic

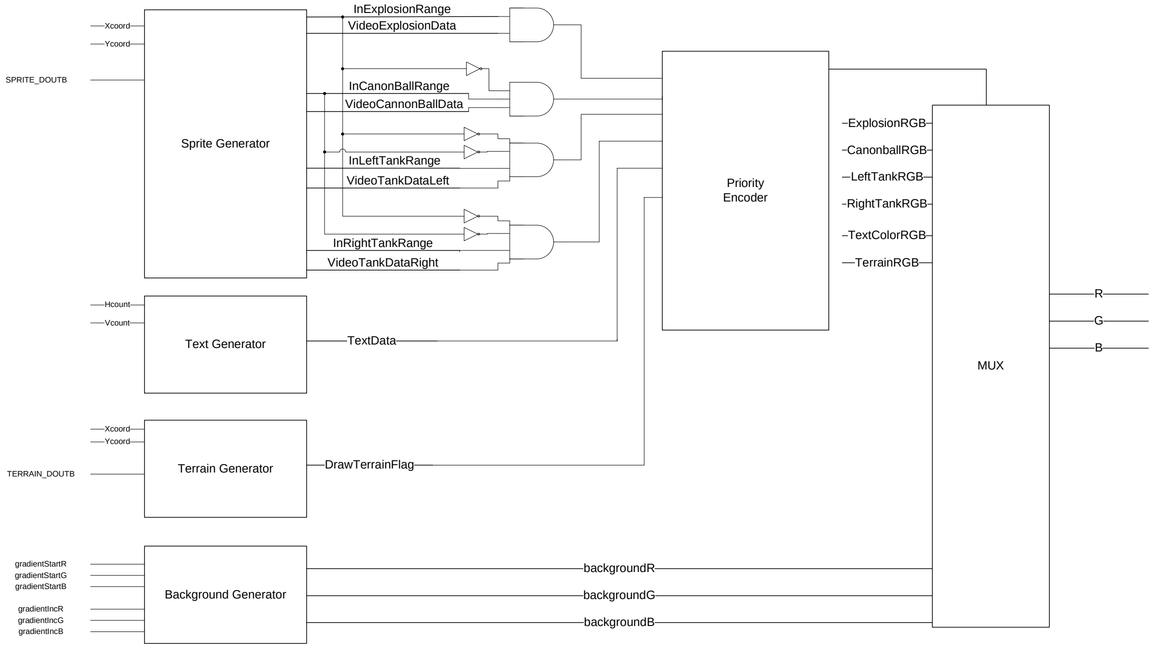
Main Program Loop



Projectile Parbola Path and Collision Logic


START
Calculate velocity X and Y vectors
Calculate starting X and Y coordiate
LOOP: X coordinate within width of display screen AND Y > 0
Is Y within display height?
YES
Parabola intersect terrain?
YES
Draw explosion sprite
NO
NO
Projectile destroy tank?
YES
Set end game flag
NO
Do not display artillery sprite
Adjust X and Y coordinate
END

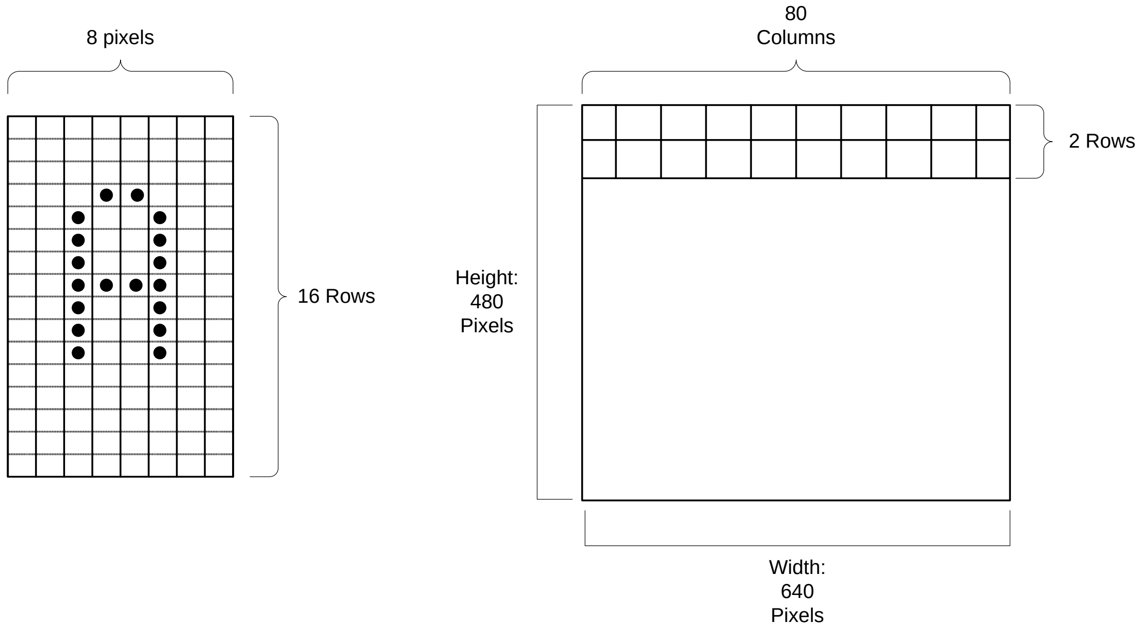
Video Controller



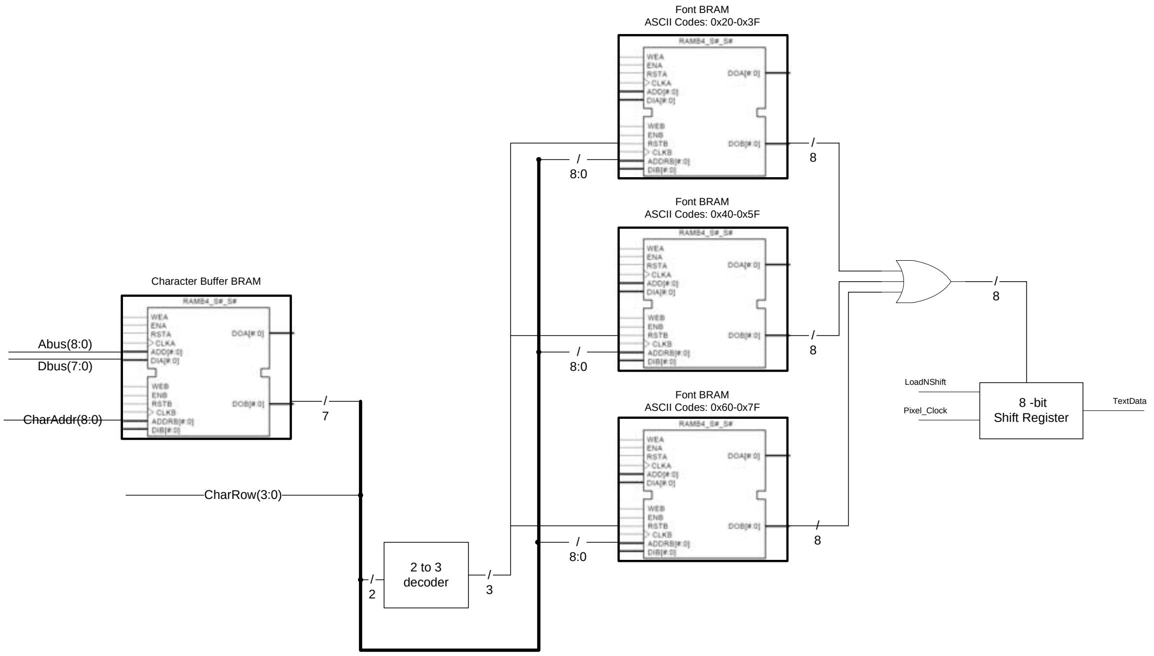
Character Font Controller

We needed a way to display text on the screen to indicate game play status such as angle, power, wind, etc. It requires the ability to be updated through software on the fly and it must not rely on any sort of video pixel buffer. It also must be able to change color whenever we needed it to. Most of this code was already done in lab 2 - 2005 but the functionality had to be implemented into lab 5 - 2004. The components that the text controller required were, of course, the video DAC, BRAMs to hold the character buffer and bitmapped fonts, as well as access to the OPB for the software to control the text on the screen.

Each character on the screen is 8 pixels wide by 16 pixels tall and with a 640x480 resolution, there are 80 characters per row. We decided to only use two lines for text, so that more space can be devoted to the game. These two lines take up 160 bytes in one BRAM because the character buffer uses a 7-bit ASCII code to refer to each particular character. This ASCII code is used as an address lookup into the font BRAM to pick which bitmap we need to display. The font BRAM contains 96 of the 128 ASCII characters (the first 32 are omitted). Each character bitmap uses 16 bytes per character (1 bit = 1 pixel) and a total of three BRAMs (1536 bytes) for the entire character set.



The characters are displayed on the screen using a shift register with a single bit being pushed out to the video DAC every clock cycle. The shift register is loaded every eight clock cycles with a single row of a character font bitmap. The particular character is chosen by what row and column the video display is at and what ASCII code is in the character buffer at that position. Depending on the pixel row that the video display is at, the corresponding font row (of 16) will be loaded into the shift register.



Memory Controller

The memory controller is split into two segments, one for SRAM access and one for register and BRAM access. The SRAM memory space is from 0x00800000 to 0x008FFFFF and the register/BRAM space is from 0x01800000 to 0x0180FFFF. The memory controller uses the lab 5 - 2004 to handle SRAM requests. The register/BRAM access uses a separate chip select, chip enable, and transfer acknowledgement signal for its memory reads and writes.

This memory controller uses a one-hot state machine adapted from Marcio's/Thing-A-Ma-Flipper's code. It initially is set to an idle state and then is triggered by the chip select. Then it sets the chip enable high in the next state. The next state is when the transfer acknowledge to the OPB is set high. But because we have two memory accesses (SRAM and register/BRAMs) we must check whether the register/BRAM memory controller has its chip enable set. If it does then we will read/write onto the OPB data lines if not the the SRAM memory controller shall have access. The same method is used for the transfer acknowledge.

Sprite Generation

Each individual sprite are composed from 16 x 16 bitmap images that is initialized and stored in one of the BRAMs. There are 10 bitmap images for the tanks (1 per 10 degree angle of the turret), 1 for the cannonball and 1 for the explosion.

Sprite BRAM

Initially, I decided to use an existing 512 x 8 bit BRAM to read the bitmap images. Thanks to Professor Edward’s suggestion, I instantiated a 256 x 16 bit dual-ported BRAM instead to make my job easier. This enabled me to read 16 bits of data at a time from the sprite BRAM. Port A of this BRAM is configured to be 8 bits data in and data out, while Port B of this BRAM is configured to be a 16 bit I/O port. This configuration gave us the flexibility of reading and writing 8 bits or 16 bits to and from the BRAM. Although reading and writing 8 bits to the BRAM is nice to have, this feature was never exploited in our project. It may be useful for future implementations.

Sprite Addressing

In order to successfully read data from the 16 bit wide BRAM, an 8 bit address line is required. The address line is composed by concatenating a 4 bit address that selects a sprite bitmap and a 4 bit font row that selects the desired row of the sprite. Data is constantly being read from the sprite BRAM, but is only valid when the video controller's (X, Y) pixel count is in the range of a sprite's 16 x 16 block span. However, the data is loaded into the array only when the video controller's pixel count reaches one of the sprite's starting (X, Y) coordinate. After reading the data in one clock cycle, the data is stored in an array and shifted out to the VGA component, starting with the MSB.

Sprite Shift Register

A 16 bit shift register was dedicated to shifting out the sprite's data to the VGA component. Just like the shift register for the character fonts, the MSB is shifted out to the VGA component. But, unlike the shift register for the character fonts, zeroes are shifted in from the right to clear the buffer, so that the tank is only drawn once. This applied to shifting out data for the Left Tank, the Cannonball, and the Explosion sprites. But to draw the Right Tank, we need to get the mirror image of the Left Tank. This was easily accomplished by shifting out the sprite data in reverse order, starting with the LSB. Subsequently, zeroes are shifted in from the left to empty the buffer.

Collaborating with Software

For each individual sprite, the C program is able to write an X, Y coordinate to hardware through registers (latches). This X, Y coordinate is used as the starting point for drawing the sprites. In order to address the correct sprite, the address to the OPB must range inclusively between FFB1 and FFBA. Additionally, the 4 bit sprite selector, as described in Sprite Addressing above, is latched as well.

Bitmap Generation

Here is an example of how the sprite BRAM is initialized:

```
attribute INIT_09 of RAMB4_SPRITE : label is -- Tank Angle 90 bottomhalf/tophalf
    "ffffffff7ffe3ffc3ffc1f001f000400040004000400040004000400040004000400";
```

The string is represented in hex, so each digit refers to 4 bits. The first font row of the tank is encoded by the right most 4 digits or 16 bits, and so on.

In order to generate the above string, each sprite was traced out on a 16 x 16 graph paper. The derived bitmap images from the graph paper are shown at the Appendix.

Gradient Background

Display a horizontally gradiated background using all 30-bits of RGB color. It should be able to have a starting RGB value at the top of the screen and gradually end up at an ending RGB value at the bottom of the screen. The algorithm to accomplish this task is as follows:

```
For each color (i.e. red, green, and blue) have a starting and ending
value of 10-bits, StartRed and EndRed.

Length = 480

IncRed = (StartRed - EndRed) / Length
IncGreen = (StartGreen - EndGreen) / Length
IncBlue = (StartBlue - EndBlue) / Length

RedOut[0] = StartRed
GreenOut[0] = StartGreen
BlueOut[0] = StartBlue

for (i = 1; i < Length; i++) {
  RedOut[i] += IncRed;
  GreenOut[i] += IncGreen;
  BlueOut[i] += IncBlue;
}
```

Unfortunately, VHDL has limited support for floating point numbers, which makes things more omplicated than it should be. Thus, when the increment value is less than 1, this algorithm does not work. Therefore, a different algorithm must be used to generate the gradient. If the increment is less than one, the value of the color increments very slowly compared to the current line it is on. As an approximation, the color value can increment by 1 every x number of pixels, where is the inverse of the increment value.

There were some bugs that came up after implementing this modified algorithm. When a particular gradient value would keep incrementing, it would not stop and continue until it loops over its counter value. A conditional was put into place to stop the increment from happening once it reached the maximum value.

Terrain Generation

We began to develop the terrain component by considering system constraints.

One constraint was the limited number of BRAMs we had. In order to conserve this, we chose to allocate at most one BRAM block for terrain data.

The next factor to consider was the size of each terrain data. Given that 32 pixels out of the 480 pixel height of the display screen was set aside for text display, our terrain data could easily fit in the smallest size, that is 8 bits. Thus we instantiated the BRAM components using RAMB4_S8_S8 module template.

Since BRAM is divided into two parts for bi-directional access to the data, we had to assign I/O conventions.

We decided to assign the A ports for data input. During gameplay, the C program updates the terrain data to display an "Earf"quake effect. We also decided to assign the A ports for data output. The C program also retrieves the terrain data when calculating the plot for the artillery parabola. (The parabola terminates upon intersecting the terrain.)

In order to address the A component, we used the addressing system of lab 5 2004, which was the basis of our final code. The convention here is to have bits 15 down to 9 of the OPB address lines as the address lines for the BRAM. Relying on a one-hot encoding, specific BRAMS are enabled. In effect, all BRAMs use the OPB address lines to address specific data, but at any time only one is enabled.

The OPB_RNW signal is used to determine a read or write operation. During gameplay, the C program updates the terrain by offsetting the points with a sine wave.

```
for (x = 0; x < 320; x++){
    buffer1[x] = XIo_In8(BRAM_ADDR__TERRAIN_DATA + x);
}

while(num_bounce > 0){
    for (x = 0; x < 320; x++) {
        y = (int) ((float)buffer1[x] + (calc_amplitude(jitter) * height) );
        if (y < 1) y = 1;
        XIo_Out8(BRAM_ADDR__TERRAIN_DATA + x, y);
        jitter += 1;
    }

    tmpR = XIo_In8(BRAM_ADDR__TERRAIN_DATA + 597/2);
    tmpL = XIo_In8(BRAM_ADDR__TERRAIN_DATA + 26/2);

    XIo_Out16(REG_ADDR__RIGHT_TANKY, tmpR + 250 - 23 );
    XIo_Out16(REG_ADDR__LEFT_TANKY, tmpL + 250 - 20 );
    num_bounce--;

    for(i=0; i < 10000; i++); //delay the bouncing
}

for (x = 0; x < 320; x++){
    XIo_Out8(BRAM_ADDR__TERRAIN_DATA + x, buffer1[x]);
}
```

We assigned the B ports for reading by the video display system.

How to generate the fixed terrain data was the last task to consider. We wanted it to appear like a random outline, yet spare ourselves from manually plotting each point. Since we had a parabola function, we altered this routine to generate curves of varying appearance. Composing these curve points, with a little "noise" added in, resulted in the terrain outline. We wrote a small Java program to translate these data points in the reversed-format required by the BRAMS and pasted the result to the VHDL file.

```
RAMB4_TERRAIN : RAMB4_S8_S8
port map (
 DOA   => TERRAIN_DOUTA,
 ADDRA => ABus(8 downto 0),
 CLKA  => OPB_Clk,
 DIA   => DBus(7 downto 0),
 ENA   => '1',
 RSTA  => RST(1),
 WEA   => WE(1),
 DOB   => TERRAIN_DOUTB,
 ADDRB => Xcoord (9 downto 1),
 CLKB  => Pixel_Clock,
 DIB   => X"00",
 ENB   => '1',
 RSTB  => '0',
 WEB   => '0');
```

Drawing the terrain was another major goal. The terrain is one of several graphics the game displays. Others are tank and artillery sprites and text fonts. A priority encoder arbitrates which of these elements are to be displayed by the video system based on the x-coordinate of the video display "gun". This x-coordinate value is used to index the BRAM to retreive the terrain data. The following VHDL code latches this value.

```
Xcoord <= Hcount - HSYNC - HBACK_PORCH;
Ycoord <= Vcount - VSYNC - VBACK_PORCH;
```

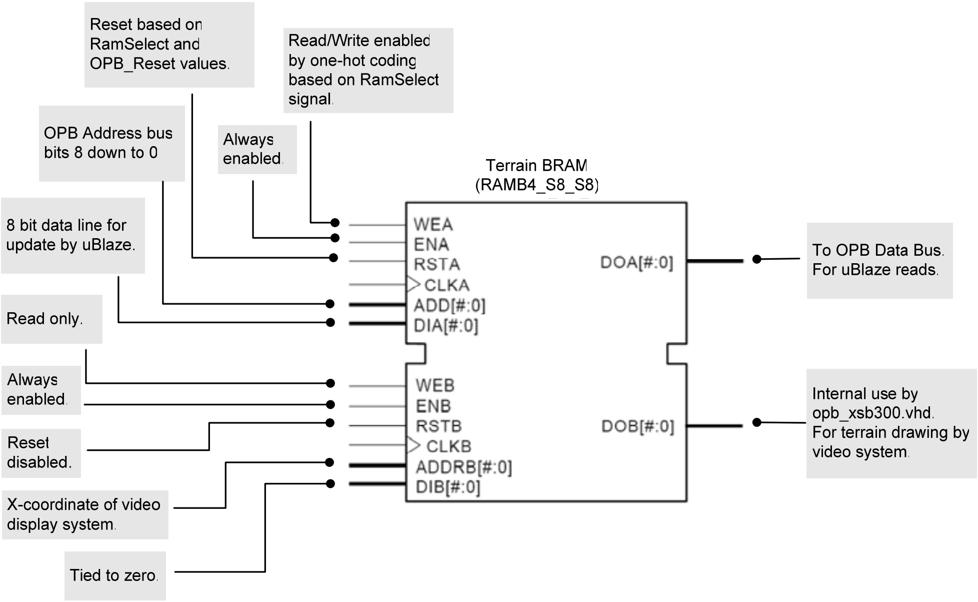
We bounded the terrain region between 0 and 640 on the x-coordinate and the terrain offset from the absolute zero of the display screen. (Absolute zero is at the upper left corner.) Whether the terrain is to be displayed or not is determined by the following VHDL code:

```
drawTerrainFlag <= '1' when ( (Xcoord >= 0) and (Xcoord < 640) ) and
  ( Ycoord >= (terrainHeight + 250) ) else '0';
```

The `drawTerrainFlag` is used by the priority encoder to ultimately determine whether or not to draw the terrain.

```
elsif ( drawTerrainFlag = '1' ) then
  -- Terrain
  r <= terrainColor(29 downto 20);
  g <= terrainColor(19 downto 10);
  b <= terrainColor(9 downto 0);
```

Below is a block diagram of the terrain BRAM.


Reset based on RamSelect and OPB_Reset values.
Read/Write enabled by one-hot coding based on RamSelect signal.
OPB Address bus bits 8 down to 0
Always enabled.
Terrain BRAM (RAMB4_S8_S8)
8 bit data line for update by uBlaze.
WEA
ENA
RSTA
CLKA
ADD[#:0]
DIA[#:0]
DOA[#:0]
To OPB Data Bus. For uBlaze reads.
Read only.
Always enabled.
Reset disabled.
X-coordinate of video display system.
Tied to zero.
WEB
ENB
RSTB
CLKB
ADDRB[#:0]
DIB[#:0]
DOB[#:0]
Internal use by opb_xsb300.vhd. For terrain drawing by video system.

Lessons Learned & Advice for Future Students

Jeremy Chou

From the start of this project, each one of us stayed in the lab working on our project. We have spent countless hours writing and modifying the project. Initially, we were focused on getting the tanks, trajectory and terrain to draw on the VGA monitor. We were able to accomplish this in software and demo it at the 75% deadline. The software prototype gave us a lot of insight into what we needed to implement to transition the graphics from software to hardware only.

The transition was difficult because we basically had to start from scratch. Professor Edwards suggested a few tips to us on the approach to take; which was to move off the video frame buffer and to use sprites for a smoother animation. For this project, I was responsible for generating the sprites. I ran into a lot of problems relating to timing issues, latching sequential signals, and reading incorrect data from the BRAM.

Fortunately, I had great teammates who helped me along the way. Our initial plan of splitting up the project into three components, dedicating a group member to each part, but also be involved in the other member's part turned out surprisingly well. We were involved in every aspect of the project, understanding how each component functions and knowing the exact status of each member. In the end, I believe all the hardworking paid off. I had lots of fun and a great time working with my teammates. It was a wonderful experience. Advice for future students: Find the right partners to work with and start early.

Michael Sumulong

This project required alot of time in the lab but was definitely a worthwhile experience. There are a few things that I would advise future students of Embedded Systems Design to follow.

When it comes to standard protocols, such as the OPB, make sure you are following it exactly. Study it in detail and make sure your signals are asserted or deasserted when they are supposed to be. Spending hours trying to find out that a bug was caused by not setting a bus back to zeroes is not fun.

Another thing that I should've done was to resist the urge to just change something in VHDL then compile it and see if it works. This can be a tedious effert when everytime you compile, 15 to 20 minutes go by.

Make an effort to really understand what you are doing in hardware or software. Also, look at the previous years' projects and their code, it can be enlightening to see how others have already implemented a feature that your project might need.

Take advantage of the TA's and Professor's advice and knowledge, they can save you alot of time and effort in the long run. But if the TA is telling you to do one thing and the Professor is telling you another, it might be best to listen to your instincts. Lastly, when debugging your code, examine as much of the behavior as possible and think of all possible causes. What you think the software/hardware is doing could be completely off and doing something else.

Dennis M. Chua

- Time Management Is Critical

  Our group was made up of three MS students who were all enrolled full time. Looking back, we were able to match the demands of this course by alloting regular working hours. On the average, we spent three hours per day, for four days per week working on the lab exercises and on the project. The TA, Marcio, made the observation that students who put in regular hours are able to meet the project goals on time. His observation matches our experience.

- Division of Labor Works

  Each member had his strengths and weaknesses. We tried to divide the project tasks not only along these lines, but also according to personal preference.

  As in any project where prior experience is absent, achieving a clear-cut and well-executed division of labor leading to a smooth final integration is difficult. What helped was that we were willing to cross the bounds of responsibilities and help each other out whenever possible. Not only did this help us get over obstacles, it also allowed us to understand how the project merged as a whole. Next to the TA's guidance and the Professor's lecture notes, we found adequate help among ourselves to make this project a success.

- Borrow and Adapt

Having a pre-existing Scorched “Earf” game freely available on the Internet gave us clear functional goals. There was never any major disagreement in specifications because we had a model at hand.

But implementing the game in XESS proved to be a challenge. There was no exact precedence from prior W4840 projects for us to adapt. Nonetheless, we benefited from other people's code.

In one instance, we could not correctly implement the retrieval of BRAM data from the C code. The bug in our work lay in the faulty FSM for the OPB data transfer. Our "EUREKA" moment came when we stumbled on the TAMF project from 2004 and studied their code. Here's a transcript of that discovery:

MIKE: "Oh my God, this is crazy!"
DENNIS: "I don't get it, Mike. You look like you've just seen a drop-dead gorgeous girl. All I see is VEE-H-DEE-L!"
MIKE: "This is it. Look at how they did their code."
DENNIS: "Hey, you're right. That's a Marcio-FSM they have there."
MIKE: "We can do this! We can do this!"

As a Japanese saying goes, "To invent is admirable, but to copy and improve on, that's innovation." We had our bug corrected in less than fifteen minutes after co-opting what we needed.

Relevant Source Code

1. opb_xsb300.vhd
2. main.c
3. system.mhs
4. opb_xsb_v2_0_0.mpd

opb_xsb300.vhd

```
-----------------------------------------------------------------------
--
-- OPB bus bridge for the XESS XSB-300E board
--
-- Includes a memory controller, a VGA framebuffer, and glue for the SRAM
--
--
-- Cristian Soviani, Dennis Lim, and Stephen A. Edwards
--
-- Modified for XESS Scorched Earch by:
-- Dennis Chua, Jeremy Chou, and Michael Sumulong
--
-----------------------------------------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity opb_xsb300 is
  generic (
    C_OPB_AWIDTH : integer          := 32;
    C_OPB_DWIDTH : integer          := 32;
    C_BASEADDR   : std_logic_vector := X"2000_0000";
    C_HIGHADDR   : std_logic_vector := X"2000_00FF");

  port (
    OPB_Clk        : in    std_logic;
    OPB_Rst        : in    std_logic;
    OPB_ABus       : in    std_logic_vector (31 downto 0);
    OPB_BE         : in    std_logic_vector (3 downto 0);
    OPB_DBus       : in    std_logic_vector (31 downto 0);
    OPB_RNW        : in    std_logic;
    OPB_select     : in    std_logic;
    OPB_seqAddr    : in    std_logic;
    pixel_clock    : in    std_logic;
    UIO_DBus       : out   std_logic_vector (31 downto 0);
    UIO_errAck     : out   std_logic;
    UIO_retry      : out   std_logic;
    UIO_toutSup    : out   std_logic;
    UIO_xferAck    : out   std_logic;
    PB_A           : out   std_logic_vector (19 downto 0);
    PB_UB_N        : out   std_logic;
    PB_LB_N        : out   std_logic;
    PB_WE_N        : out   std_logic;
    PB_OE_N        : out   std_logic;
    RAM_CE_N       : out   std_logic;
    VIDOUT_CLK     : out   std_logic;
    VIDOUT_RED     : out   std_logic_vector (9 downto 0);
    VIDOUT_GREEN   : out   std_logic_vector (9 downto 0);
    VIDOUT_BLUE    : out   std_logic_vector (9 downto 0);
    VIDOUT_BLANK_N : out   std_logic;
    VIDOUT_HSYNC_N : out   std_logic;
    VIDOUT_VSYNC_N : out   std_logic;
    PB_D           : inout std_logic_vector (15 downto 0));
end opb_xsb300;

architecture Behavioral of opb_xsb300 is

  ---------------------------------------------------------------------
  -- Constant Parameters
  ---------------------------------------------------------------------

  -- Video parameters
  constant HTOTAL       : integer := 800;
  constant HSYNC        : integer := 96;
  constant HBACK_PORCH  : integer := 48;
  constant HACTIVE      : integer := 640;
  constant HFRONT_PORCH : integer := 16;

  constant VTOTAL       : integer := 525;
  constant VSYNC        : integer := 2;
  constant VBACK_PORCH  : integer := 33;
  constant VACTIVE      : integer := 480;
  constant VFRONT_PORCH : integer := 10;

  -- Sprite parameters
  constant sprite_dimX : integer := 17;
  constant sprite_dimY : integer := 16;


  -------------------------------------------------------------------
  -- Component Declarations
  -------------------------------------------------------------------

  -- 512 X 8 dual-ported Xilinx block RAM
  component RAMB4_S8_S8
    port (
      DOA   : out std_logic_vector (7 downto 0);
      ADDRA : in  std_logic_vector (8 downto 0);
      CLKA  : in  std_logic;
      DIA   : in  std_logic_vector (7 downto 0);
      ENA   : in  std_logic;
      RSTA  : in  std_logic;
      WEA   : in  std_logic;
      DOB   : out std_logic_vector (7 downto 0);
      ADDRB : in  std_logic_vector (8 downto 0);
      CLKB  : in  std_logic;
      DIB   : in  std_logic_vector (7 downto 0);
      ENB   : in  std_logic;
      RSTB  : in  std_logic;
      WEB   : in  std_logic);
  end component;

  -- 256 X 16 dual-ported Xilinx block RAM
  -- Port A is 8 bit wide, Port B is 16 bit wide
  component RAMB4_S8_S16
    port (
      DOA   : out std_logic_vector (7 downto 0);
      ADDRA : in  std_logic_vector (8 downto 0);
      CLKA  : in  std_logic;
      DIA   : in  std_logic_vector (7 downto 0);
      ENA   : in  std_logic;
      RSTA  : in  std_logic;
      WEA   : in  std_logic;
      DOB   : out std_logic_vector (15 downto 0);
      ADDRB : in  std_logic_vector (7 downto 0);
      CLKB  : in  std_logic;
      DIB   : in  std_logic_vector (15 downto 0);
      ENB   : in  std_logic;
      RSTB  : in  std_logic;
      WEB   : in  std_logic);
  end component;

  component memoryctrl
    port (
      rst        : in  std_logic;
      clk        : in  std_logic;
      cs         : in  std_logic;
      select0    : in  std_logic;
      RNW        : in  std_logic;
      vreq       : in  std_logic;
      onecycle   : in  std_logic;
      videocycle : out std_logic;       -- try to discontinue use
      hihalf     : out std_logic;
      pb_wr      : out std_logic;
      pb_rd      : out std_logic;
      xfer       : out std_logic;
      ce0        : out std_logic;
      ce1        : out std_logic;
      rres       : out std_logic;
      video_ce   : out std_logic);      -- try to discontinue use
  end component;

  component pad_io
    port (
      clk       : in    std_logic;
      rst       : in    std_logic;
      PB_A      : out   std_logic_vector(19 downto 0);
```

```
      PB_UB_N   : out   std_logic;
      PB_LB_N   : out   std_logic;
      PB_WE_N   : out   std_logic;
      PB_OE_N   : out   std_logic;
      RAM_CE_N  : out   std_logic;
      PB_D      : inout std_logic_vector(15 downto 0);
      pb_addr   : in    std_logic_vector(19 downto 0);
      pb_ub     : in    std_logic;
      pb_lb     : in    std_logic;
      pb_wr     : in    std_logic;
      pb_rd     : in    std_logic;
      ram_ce    : in    std_logic;
      pb_dread  : out   std_logic_vector(15 downto 0);
      pb_dwrite : in    std_logic_vector(15 downto 0));
  end component;

  -- Fast low-voltage TTL-level I/O pad with 12 mA drive
  component OBUF_F_12
    port (
      O : out std_ulogic;
      I : in  std_ulogic);
  end component;

  -- Basic edge-sensitive flip-flop
  component FD
    port (
      C : in  std_logic;
      D : in  std_logic;
      Q : out std_logic);
  end component;

  -- Force instances of FD into pads for speed
  attribute iob       : string;
  attribute iob of FD : component is "true";


  ---------------------------------------------------------------------
  -- Signal Declarations
  ---------------------------------------------------------------------

  -- BRAM Signals
  signal RamPageAddress                 : std_logic_vector(6 downto 0);
  signal RamSelect                      : std_logic_vector (7 downto 0);
  signal DOUT0, DOUT5, DOUT6, DOUT7     : std_logic_vector(7 downto 0);
  signal DOUTB0, DOUTB5, DOUTB6, DOUTB7 : std_logic_vector(7 downto 0);
  signal RST, WE                        : std_logic_vector (7 downto 0);

  -- Signals for the video controller
  signal Hcount                : std_logic_vector(9 downto 0);  -- Horizontal position
(0-800)
  signal Vcount                : std_logic_vector(9 downto 0);  -- Vertical position (0-
524)
  signal Xcoord, Ycoord        : std_logic_vector(9 downto 0);
  signal HBLANK_N, VBLANK_N    : std_logic;  -- Blanking signals
  signal EndOfLine, EndOfField : std_logic;
  signal r                     : std_logic_vector (9 downto 0);
  signal g                     : std_logic_vector (9 downto 0);
  signal b                     : std_logic_vector (9 downto 0);

  -- Signals for the character generator
  signal FontLoad, LoadChar : std_logic;  -- Font/Character RAM read triggers
  signal FontAddr           : std_logic_vector(10 downto 0);
  signal CharRamPage        : std_logic_vector(2 downto 0);
  signal CharRamSelect_N    : std_logic_vector(4 downto 0);
  signal FontRamPage        : std_logic_vector(1 downto 0);
  signal FontRamSelect_N    : std_logic_vector(2 downto 0);
  signal CharAddr           : std_logic_vector(11 downto 0);
  signal CharColumn         : std_logic_vector(9 downto 0);
  signal CharRow            : std_logic_vector(9 downto 0);
  signal Column             : std_logic_vector(6 downto 0);  -- 0-79
  signal Row                : std_logic;  -- 0-1

  signal LoadNShift : std_logic;        -- Shift register control
  signal FontData   : std_logic_vector(7 downto 0);  -- Input to shift register
  signal ShiftData  : std_logic_vector(7 downto 0);  -- Shift register data
  signal TextData   : std_logic;        -- Serial out ANDed with blanking
  signal textColor : std_logic_vector (29 downto 0);

  -- Signals for the gradient background
  signal gradientStartR, gradientStartG, gradientStartB : std_logic_vector(9 downto 0);
  signal gradientIncR, gradientIncG, gradientIncB       : std_logic_vector(10 downto 0);
  signal backgroundR, backgroundG, backgroundB          : std_logic_vector(9 downto 0);
  signal gradientCtrR, gradientCtrG, gradientCtrB       : std_logic_vector(9 downto 0);

  -- Signals for the SRAM
  signal SRAM_addr_mux : std_logic_vector(19 downto 0);
  signal video_addr    : std_logic_vector (19 downto 0);  -- NOT USED ANYMORE
  signal video_data    : std_logic_vector (15 downto 0);  -- NOT USED ANYMORE
  signal video_req     : std_logic;     -- try to discontinue use
  signal video_ce      : std_logic;     -- try to discontinue use
  signal cs            : std_logic;

  signal onecycle           : std_logic;
  signal videocycle         : std_logic;  -- try to discontinue use
  signal amuxsel            : std_logic;  -- try to dicontinue use
  signal hihalf             : std_logic;
  signal rce0, rce1, rreset : std_logic;
  signal xfer               : std_logic;

  signal RNW : std_logic;

  signal ABus      : std_logic_vector (31 downto 0);
  signal DBus      : std_logic_vector (31 downto 0);
  signal wdata_mux : std_logic_vector (15 downto 0);
  signal BE        : std_logic_vector (3 downto 0);

  signal pb_bytesel   : std_logic_vector (1 downto 0);
  signal SRAM_rdata   : std_logic_vector (15 downto 0);
  signal SRAM_CE      : std_logic;
  signal pb_wr, pb_rd : std_logic;

  -- Terrain related signals
  signal terrainHeight                : std_logic_vector(9 downto 0);
  signal TERRAIN_DOUTA, TERRAIN_DOUTB : std_logic_vector(7 downto 0);
  signal drawTerrainFlag              : std_logic;
  signal terrainColor                 : std_logic_vector(29 downto 0);


  -- Sprite related signals
  signal SPRITE_DOUTA                             : std_logic_vector(7 downto 0);  --
left hanging
  signal SPRITEAddr                               : std_logic_vector(7 downto 0);  --
composed from WhichSprite & SpriteFontRow
  signal LeftTankFontRow, RightTankFontRow        : std_logic_vector(3 downto 0);  --
gets row 0 to 15 of sprite font
  signal CannonballFontRow, ExplosionFontRow      : std_logic_vector(3 downto 0);  --
gets row 0 to 15 of sprite font
  signal WhichSpriteLeftTank                      : std_logic_vector(3 downto 0);  --
gets the different sprite bitmaps
  signal WhichSpriteRightTank                     : std_logic_vector(3 downto 0);  --
gets the different sprite bitmaps
  signal SPRITE_DOUTB                             : std_logic_vector(15 downto 0);  -- 16
bit read from bram
  signal LeftTankLoadNShift, RightTankLoadNShift  : std_logic;  -- 16 bit Shift register
control
  signal CannonballLoadNShift, ExplosionLoadNShift: std_logic;  -- 16 bit Shift register
control
  signal ShiftLeftTankData                        : std_logic_vector(15 downto 0);  -- 16
bit Shift register data for Left Tank
  signal ShiftRightTankData                            : std_logic_vector(15 downto 0);
-- 16 bit Shift register data for Right Tank
  signal ShiftCannonballData                           : std_logic_vector(15 downto 0);
-- 16 bit Shift register data for Cannonball
  signal ShiftExplosionData                            : std_logic_vector(15 downto 0);
-- 16 bit Shift register data for Explosion
  signal VideoTankDataLeft                             : std_logic;  -- each pixel
shifted out from shift register to vga
  signal VideoTankDataRight                            : std_logic;  -- each pixel
shifted out from shift register to vga
  signal VideoExplosionData                            : std_logic;  -- each pixel
shifted out from shift register to vga
  signal VideoCannonballData                           : std_logic;  -- each pixel
shifted out from shift register to vga
```

```
  signal LeftTankSprite_x, LeftTankSprite_y           : std_logic_vector(9 downto 0);  -
- x and y coordinates of the sprite
  signal RightTankSprite_x, RightTankSprite_y         : std_logic_vector(9 downto 0);  -
- x and y coordinates of the sprite
  signal CannonballSprite_x, CannonballSprite_y       : std_logic_vector(9 downto 0);  -
- x and y coordinates of the sprite
  signal ExplosionSprite_x, ExplosionSprite_y         : std_logic_vector(9 downto 0);  -
- x and y coordinates of the sprite
  signal LeftTank_line_toggle, RightTank_line_toggle   : std_logic;  -- equals 1 when we
are in range of sprite
  signal Cannonball_line_toggle, Explosion_line_toggle : std_logic;  -- equals 1 when we
are in range of sprite
  signal InLeftTankRange, InRightTankRange            : std_logic;
  signal InCannonballRange, InExplosionRange          : std_logic;
  signal InLeftTankRangeAddr, InRightTankRangeAddr    : std_logic;
  signal InCannonballRangeAddr, InExplosionRangeAddr   : std_logic;



---- TAMF EXPERIMENT!!

  signal data_bus, data_bus_ce, data_bus_rce : std_logic_vector(31 downto 0);
  signal cs2, q2, q1, q0, ce, xfer2 : std_logic;



  ---------------------------------------------------------------------------
  -- BRAM Initialization
  ---------------------------------------------------------------------------

  attribute INIT_00 : string;
  attribute INIT_01 : string;
  attribute INIT_02 : string;
  attribute INIT_03 : string;
  attribute INIT_04 : string;
  attribute INIT_05 : string;
  attribute INIT_06 : string;
  attribute INIT_07 : string;
  attribute INIT_08 : string;
  attribute INIT_09 : string;
  attribute INIT_0a : string;
  attribute INIT_0b : string;
  attribute INIT_0c : string;
  attribute INIT_0d : string;
  attribute INIT_0e : string;
  attribute INIT_0f : string;

  -- Terrain altitude values (0-512)
  attribute INIT_00 of RAMB4_TERRAIN : label is
    "b4afaba7a39f9c9997949391908e8e8e8e8e9091939597999ca0a3a7abb0b4";
  attribute INIT_01 of RAMB4_TERRAIN : label is
    "b1a396918c91969a9da3a7a9abb3b4b7b6b4b3a7a4a4a4a4aaadb1b3b4b7b9";
  attribute INIT_02 of RAMB4_TERRAIN : label is
    "07080b0d1013161a1e22272c31373d434950585f62666e767b8994a0b2bcbab6";
  attribute INIT_03 of RAMB4_TERRAIN : label is
    "7870686058514a433d37312c27221e1a1613100d0b09070504030303030405";
  attribute INIT_04 of RAMB4_TERRAIN : label is
    "8283848586878a888785858484848485868787898b8e90939697989b9997948a81";
  attribute INIT_05 of RAMB4_TERRAIN : label is
    "0001020304050607090a141e28323c46505a6e736e7378787a7b7c7d7e7f8081";
  attribute INIT_06 of RAMB4_TERRAIN : label is
    "6a676767676767675f564e453e362f28211b150f0a05040302010102020100000";
  attribute INIT_07 of RAMB4_TERRAIN : label is
    "98969493929190909091929394969898b9da19d9a9694928b88807b736d64676a";
  attribute INIT_08 of RAMB4_TERRAIN : label is
    "b9babdc0c3c5c7c8c9cacbcccdcecfd0d1d2d2d0cac4bfb9b4b0aba7a4a09d9a";
  attribute INIT_09 of RAMB4_TERRAIN : label is
    "b4afaba7a39f9c9997949391908f8e8e8e8e8f9091939597999ca0a3a7abb0b4";
  attribute INIT_0a of RAMB4_TERRAIN : label is
    "77797b7e8084878b8f93989da2a8aeb4b7b9bac8c8c7c6c5c4c0c0c3c4c5bfb9";
  attribute INIT_0b of RAMB4_TERRAIN : label is
    "d6d6d6cec7c0b9b2aca6a09b96928d8986827f7d7a787775747373737373747576";
  attribute INIT_0c of RAMB4_TERRAIN : label is
    "1b1d2024272b3034393f444a50575e656d757d858e97a1aad6d6d6d6d6d6d6d6";
  attribute INIT_0d of RAMB4_TERRAIN : label is
    "62615f58514b453f3a35302c2824211e1b19161513121111111111111213141618";
  attribute INIT_0e of RAMB4_TERRAIN : label is
```

```
    "9597999ca0a3a7abb0b4b7b8bcbdbec3c9c8c5c1c1bbb3ada59f998f875e5d62";
  attribute INIT_0f of RAMB4_TERRAIN : label is
    "b4b4b4b4b4afafa7a8a7a39f9797949391908f8e8e8e8e8e8e8e8e8e8f909193";


  -- Sprite bitmap values for 16 bit reads
  attribute INIT_00 of RAMB4_SPRITE : label is -- Tank Angle 0 bottomhalf/tophalf
    "ffffffff7ffe3ffc3ffc1f001f0007ff00000000000000000000000000000000";
  attribute INIT_01 of RAMB4_SPRITE : label is -- Tank Angle 10 bottomhalf/tophalf
    "ffffffff7ffe3ffc3ffc1f001f00070000f0000f000000000000000000000000";
  attribute INIT_02 of RAMB4_SPRITE : label is -- Tank Angle 20 bottomhalf/tophalf
    "ffffffff7ffe3ffc3ffc1f001f0006000180006001c000300000000000000000";
  attribute INIT_03 of RAMB4_SPRITE : label is -- Tank Angle 30 bottomhalf/tophalf
    "ffffffff7ffe3ffc3ffc1f001f000400030000c0002000180006000100000000";
  attribute INIT_04 of RAMB4_SPRITE : label is -- Tank Angle 40 bottomhalf/tophalf
    "ffffffff7ffe3ffc3ffc1f001f00040003000080004000200018000400020001";
  attribute INIT_05 of RAMB4_SPRITE : label is -- Tank Angle 50 bottomhalf/tophalf
    "ffffffff7ffe3ffc3ffc1f001f00040002000100008000400020001000080004";
  attribute INIT_06 of RAMB4_SPRITE : label is -- Tank Angle 60 bottomhalf/tophalf
    "ffffffff7ffe3ffc3ffc1f001f00040002000100008000800040002000100010";
  attribute INIT_07 of RAMB4_SPRITE : label is -- Tank Angle 70 bottomhalf/tophalf
    "ffffffff7ffe3ffc3ffc1f001f00040002000200010001000080008000400040";
  attribute INIT_08 of RAMB4_SPRITE : label is -- Tank Angle 80 bottomhalf/tophalf
    "ffffffff7ffe3ffc3ffc1f001f00040004000200020002000100010001000100";
  attribute INIT_09 of RAMB4_SPRITE : label is -- Tank Angle 90 bottomhalf/tophalf
    "ffffffff7ffe3ffc3ffc1f001f00040004000400040004000400040004000400";
  attribute INIT_0a of RAMB4_SPRITE : label is -- Cannonball
    "000000000000000000000000000000003c007e00ff00ff00ff00ff007e003c00";
  attribute INIT_0b of RAMB4_SPRITE : label is -- Explosion
    "07e01ff83ffc7ffe7ffeffffffffffffffffffffffffffff7ffe7ffe3ffc1ff807e0";
  attribute INIT_0c of RAMB4_SPRITE : label is -- Square block
    "ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff";
  attribute INIT_0d of RAMB4_SPRITE : label is -- All black
    "0000000000000000000000000000000000000000000000000000000000000000";


  -- Standard 8x16 font taken from the Linux console font file "lat0-16.psfu"
  attribute INIT_00 of RAMB4_S8_S8_5 : label is
    "000000001818001818183c3c3c18000000000000000000000000000000000000";
  attribute INIT_01 of RAMB4_S8_S8_5 : label is
    "000000006c6cfe6c6c6cfe6c6c00000000000000000000000000002466666600";
  attribute INIT_02 of RAMB4_S8_S8_5 : label is
    "0000000086c66030180cc6c200000000000010107cd616167cd0d0d67c101000";
  attribute INIT_03 of RAMB4_S8_S8_5 : label is
    "000000000000000000000030181818000000000076ccccccdc76386c6c380000";
  attribute INIT_04 of RAMB4_S8_S8_5 : label is
    "0000000030180c0c0c0c0c0c18300000000000000c18303030303030180c0000";
  attribute INIT_05 of RAMB4_S8_S8_5 : label is
    "00000000000018187e18180000000000000000000000663cff3c660000000000";
  attribute INIT_06 of RAMB4_S8_S8_5 : label is
    "0000000000000000fe0000000000000000000030181818000000000000000000";
  attribute INIT_07 of RAMB4_S8_S8_5 : label is
    "0000000000c06030180c06000000000000000000181800000000000000000000";
  attribute INIT_08 of RAMB4_S8_S8_5 : label is
    "000000007e181818181818783818000000000000007cc6e6e6d6d6cecec67c0000";
  attribute INIT_09 of RAMB4_S8_S8_5 : label is
    "000000007cc60606063c0606c67c000000000000fec6c06030180c06c67c0000";
  attribute INIT_0a of RAMB4_S8_S8_5 : label is
    "000000007cc6060606fcc0c0c0fe0000000000001e0c0c0cfecc6c3c1c0c0000";
  attribute INIT_0b of RAMB4_S8_S8_5 : label is
    "0000000030303030180c0606c6fe0000000000007cc6c6c6c6fcc0c060380000";
  attribute INIT_0c of RAMB4_S8_S8_5 : label is
    "00000000780c0606067ec6c6c67c0000000000007cc6c6c6c67cc6c6c67c0000";
  attribute INIT_0d of RAMB4_S8_S8_5 : label is
    "0000003018180000001818000000000000000000001818000000181800000000";
  attribute INIT_0e of RAMB4_S8_S8_5 : label is
    "000000000000fe0000fe00000000000000000000060c18306030180c06000000";
  attribute INIT_0f of RAMB4_S8_S8_5 : label is
    "000000001818001818180cc6c67c0000000000006030180c060c183060000000";
  attribute INIT_00 of RAMB4_S8_S8_6 : label is
    "00000000c6c6c6c6fec6c66c38100000000000007cc0dcdededec6c6c67c0000";
  attribute INIT_01 of RAMB4_S8_S8_6 : label is
    "000000003c66c2c0c0c0c0c2663c000000000000fc666666667c666666fc0000";
  attribute INIT_02 of RAMB4_S8_S8_6 : label is
    "00000000fe6662606878686266fe000000000000f86c6666666666666cf80000";
  attribute INIT_03 of RAMB4_S8_S8_6 : label is
    "000000003a66c6c6dec0c0c2663c000000000000f06060606878686266fe0000";
  attribute INIT_04 of RAMB4_S8_S8_6 : label is
    "000000003c18181818181818183c000000000000c6c6c6c6c6fec6c6c6c60000";
  attribute INIT_05 of RAMB4_S8_S8_6 : label is
    "00000000e666666c78786c6666e600000000000078cccccc0c0c0c0c0c1e0000";
  attribute INIT_06 of RAMB4_S8_S8_6 : label is
    "00000000c6c6c6c6c6d6fefeeec6000000000000fe66626060606060 60f00000";
  attribute INIT_07 of RAMB4_S8_S8_6 : label is
    "000000007cc6c6c6c6c6c6c6c67c000000000000c6c6c6c6cedefef6e6c60000";
  attribute INIT_08 of RAMB4_S8_S8_6 : label is
    "00000e0c7cded6c6c6c6c6c6c67c000000000000f0606060607c666666fc0000";
  attribute INIT_09 of RAMB4_S8_S8_6 : label is
    "000000007cc6c6060c3860c6c67c000000000000e66666666c7c666666fc0000";
  attribute INIT_0a of RAMB4_S8_S8_6 : label is
    "000000007cc6c6c6c6c6c6c6c6c6000000000000003c1818181818185a7e7e0000";
  attribute INIT_0b of RAMB4_S8_S8_6 : label is
    "000000006ceefed6d6d6c6c6c6c600000000000010386cc6c6c6c6c6c6c60000";
  attribute INIT_0c of RAMB4_S8_S8_6 : label is
    "000000003c181818183c666666660000000000000c6c66c7c38387c6cc6c60000";
  attribute INIT_0d of RAMB4_S8_S8_6 : label is
    "000000003c30303030303030303c000000000000fec6c26030180c86c6fe0000";
  attribute INIT_0e of RAMB4_S8_S8_6 : label is
    "000000003c0c0c0c0c0c0c0c0c3c00000000000000000000060c183060c000000000";
  attribute INIT_0f of RAMB4_S8_S8_6 : label is
    "00ff000000000000000000000000000000000000000000000000000000c66c3810";
  attribute INIT_00 of RAMB4_S8_S8_7 : label is
    "0000000076cccccc7c0c7800000000000000000000000000000000001830303000";
  attribute INIT_01 of RAMB4_S8_S8_7 : label is
    "000000007cc6c0c0c0c67c0000000000000000007c666666666c786060e00000";
  attribute INIT_02 of RAMB4_S8_S8_7 : label is
    "000000007cc6c0c0fec67c00000000000000000076cccccccc6c3c0c0c1c0000";
  attribute INIT_03 of RAMB4_S8_S8_7 : label is
    "0078cc0c7ccccccccccc7600000000000000000000f060606060f060646c380000";
  attribute INIT_04 of RAMB4_S8_S8_7 : label is
    "000000003c18181818183800181800000000000000e666666666766c6060e00000";
  attribute INIT_05 of RAMB4_S8_S8_7 : label is
    "00000000e6666c78786c666060e00000003c66660606060606060e0006060000";
  attribute INIT_06 of RAMB4_S8_S8_7 : label is
    "00000000c6d6d6d6d6feec00000000000000000018343030303030303070 0000";
  attribute INIT_07 of RAMB4_S8_S8_7 : label is
    "000000007cc6c6c6c6c67c000000000000000000666666666666dc0000000000";
  attribute INIT_08 of RAMB4_S8_S8_7 : label is
    "001e0c0c7ccccccccccc76000000000000f060607c6666666666dc0000000000";
  attribute INIT_09 of RAMB4_S8_S8_7 : label is
    "000000007cc60c3860c67c000000000000000000f06060606676dc0000000000";
  attribute INIT_0a of RAMB4_S8_S8_7 : label is
    "0000000076cccccccccccc0000000000000000001c3630303030fc3030100000";
  attribute INIT_0b of RAMB4_S8_S8_7 : label is
    "000000006cfed6d6d6c6c6000000000000000000183c66666666660000000000";
  attribute INIT_0c of RAMB4_S8_S8_7 : label is
    "00f80c067ec6c6c6c6c6c6000000000000000000c66c3838386cc60000000000";
  attribute INIT_0d of RAMB4_S8_S8_7 : label is
    "000000000e18181818701818180e000000000000fec6603018ccfe0000000000";
  attribute INIT_0e of RAMB4_S8_S8_7 : label is
    "0000000070181818180e181818700000000000001818181818181818181 80000";
  attribute INIT_0f of RAMB4_S8_S8_7 : label is
    "000000003c1818183c66666666006600000000000000000000000000dc760000";


begin

  --------------------------------------------------------------------
  --
  -- Instances of the block RAMs
  --
  --------------------------------------------------------------------

  -- Port A is used for communication with the OPB
  -- Port B is for video
  RAMB4_S8_S8_0 : RAMB4_S8_S8
    port map (
      DOA   => DOUT0,
      ADDRA => ABus(8 downto 0),
      CLKA  => OPB_Clk,
      DIA   => DBus(7 downto 0),
      ENA   => '1',
      RSTA  => RST(0),
      WEA   => WE(0),
      DOB   => DOUTB0,
      ADDRB => CharAddr(8 downto 0),
      CLKB  => Pixel_Clock,
      DIB   => X"00",
      ENB   => '1',
      RSTB  => CharRamSelect_N(0),
      WEB   => '0');

  -- Port A: C program I/O
  -- Port B: VHDL video subsystem I/O
  RAMB4_TERRAIN : RAMB4_S8_S8
    port map (
      DOA   => TERRAIN_DOUTA,
      ADDRA => ABus(8 downto 0),
      CLKA  => OPB_Clk,
      DIA   => DBus(7 downto 0),
      ENA   => '1',
      RSTA  => RST(1),
      WEA   => WE(1),
      DOB   => TERRAIN_DOUTB,
      ADDRB => Xcoord (9 downto 1),
      CLKB  => Pixel_Clock,
      DIB   => X"00",
      ENB   => '1',
      RSTB  => '0',
      WEB   => '0');


  RAMB4_SPRITE : RAMB4_S8_S16
    port map (
      DOA   => SPRITE_DOUTA,
      ADDRA => ABus(8 downto 0),
      CLKA  => OPB_Clk,
      DIA   => DBus(7 downto 0),
      ENA   => '1',
      RSTA  => '0',
      WEA   => '0',
      DOB   => SPRITE_DOUTB,            --16 bit data out
      ADDRB => SPRITEAddr,              --8 bit address
      CLKB  => Pixel_Clock,
      DIB   => X"0000",
      ENB   => '1',
      RSTB  => '0',
      WEB   => '0');

  RAMB4_S8_S8_5 : RAMB4_S8_S8
    port map (
      DOA   => DOUT5,
      ADDRA => ABus(8 downto 0),
      CLKA  => OPB_Clk,
      DIA   => DBus(7 downto 0),
      ENA   => '1',
      RSTA  => RST(5),
      WEA   => WE(5),
      DOB   => DOUTB5,
      ADDRB => FontAddr(8 downto 0),
      CLKB  => pixel_clock,
      DIB   => X"00",
      ENB   => '1',
      RSTB  => FontRamSelect_N(0),
      WEB   => '0');

  RAMB4_S8_S8_6 : RAMB4_S8_S8
    port map (
      DOA   => DOUT6,
      ADDRA => ABus(8 downto 0),
      CLKA  => OPB_Clk,
      DIA   => DBus(7 downto 0),
      ENA   => '1',
      RSTA  => RST(6),
      WEA   => WE(6),
      DOB   => DOUTB6,
      ADDRB => FontAddr(8 downto 0),
      CLKB  => pixel_clock,
      DIB   => X"00",
      ENB   => '1',
      RSTB  => FontRamSelect_N(1),
      WEB   => '0');

  RAMB4_S8_S8_7 : RAMB4_S8_S8
    port map (
      DOA   => DOUT7,
      ADDRA => ABus(8 downto 0),
      CLKA  => OPB_Clk,
      DIA   => DBus(7 downto 0),
      ENA   => '1',
      RSTA  => RST(7),
      WEA   => WE(7),
      DOB   => DOUTB7,
      ADDRB => FontAddr(8 downto 0),
      CLKB  => pixel_clock,
      DIB   => X"00",
      ENB   => '1',
      RSTB  => FontRamSelect_N(2),
      WEB   => '0');

-- Memory control/arbitration state machine

  memoryctrl1 : memoryctrl port map (
    rst        => OPB_Rst,
    clk        => OPB_Clk,
    cs         => cs,
    select0    => OPB_select,
    RNW        => RNW,
    vreq       => video_req,            -- try to discontinue use
    onecycle   => onecycle,
    videocycle => videocycle,           -- try to discontinue use
    hihalf     => hihalf,
    pb_wr      => pb_wr,
    pb_rd      => pb_rd,
    xfer       => xfer,
    ce0        => rce0,
    ce1        => rce1,
    rres       => rreset,
    video_ce   => video_ce);            -- try to dicontinue use

-- I/O pads

  pad_io1 : pad_io port map (
    clk       => OPB_Clk,
    rst       => OPB_Rst,
    PB_A      => PB_A,
    PB_UB_N   => PB_UB_N,
    PB_LB_N   => PB_LB_N,
    PB_WE_N   => PB_WE_N,
    PB_OE_N   => PB_OE_N,
    RAM_CE_N  => RAM_CE_N,
    PB_D      => PB_D,
    pb_addr   => SRAM_addr_mux,
    pb_rd     => pb_rd,
    pb_wr     => pb_wr,
    pb_ub     => pb_bytesel(1),
    pb_lb     => pb_bytesel(0),
    ram_ce    => SRAM_CE,
    pb_dread  => SRAM_rdata,
    pb_dwrite => wdata_mux);


  ---------------------------------------------------------------------
  --
  -- SRAM Memory / OPB Stuff
  --
  ---------------------------------------------------------------------

  LatchOPBSignals : process (OPB_Clk)
  begin
    if OPB_Clk'event and OPB_Clk = '1' then
      if OPB_RST = '1' then
        ABus <= ( others => '0' );
        DBus <= ( others => '0' );
        RNW  <= '1';
        BE   <= ( others => '0' );
      else
        ABus <= OPB_ABus;
        DBus <= OPB_DBus;
        RNW  <= OPB_RNW;
        BE   <= OPB_BE;
      end if;
    end if;
  end process;


  SRAM_CE <= pb_rd or pb_wr;

  amuxsel <= videocycle;                -- try to disconinue use

  SRAM_addr_mux <= video_addr when (amuxsel = '1')  -- try to discontinue use
                   else (ABus(20 downto 2) & (ABus(1) or hihalf));

  onecycle <= (not BE(3)) or (not BE(2)) or (not BE(1)) or (not BE(0));

  wdata_mux <= DBus(15 downto 0) when ((ABus(1) or hihalf) = '1')
               else DBus(31 downto 16);

  process(videocycle, BE, ABus(1), hihalf, pb_rd, pb_wr)
  begin
    if videocycle = '1' then            -- try to discontinue use
      pb_bytesel   <= "11";
    elsif pb_rd = '1' or pb_wr = '1' then
      if ABus(1) = '1' or hihalf = '1' then
        pb_bytesel <= BE(1 downto 0);
      else
        pb_bytesel <= BE(3 downto 2);
      end if;
    else
      pb_bytesel   <= "00";
    end if;
  end process;

  cs <= OPB_select when ABus(31 downto 20) = X"008" else '0';


  -- NOT USED ANYMORE
  process (OPB_Clk)
  begin
    if OPB_Clk'event and OPB_Clk = '1' then
      if video_ce = '1' then            -- try to discontinue use
        video_data <= SRAM_rdata;
      end if;
    end if;
  end process;

  -- Write the low two bytes if rce0 or rce1 (so that all 32 bits are on the data bus) is
enabled
  process (OPB_Clk, OPB_Rst)
  begin
    if OPB_Rst = '1' then
      data_bus_rce(15 downto 0)   <= X"0000";
    elsif OPB_Clk'event and OPB_Clk = '1' then
      if rreset = '1' then
        data_bus_rce(15 downto 0) <= X"0000";
      elsif (rce1 or rce0) = '1' then
        data_bus_rce(15 downto 0) <= SRAM_rdata(15 downto 0);
      end if;
    end if;
  end process;

  -- Write the high two bytes if rce0 is enabled
  process (OPB_Clk, OPB_Rst)
  begin
    if OPB_Rst = '1' then
      data_bus_rce(31 downto 16)   <= X"0000";
    elsif OPB_Clk'event and OPB_Clk = '1' then
      if rreset = '1' then
        data_bus_rce(31 downto 16) <= X"0000";
      elsif rce0 = '1' then
        data_bus_rce(31 downto 16) <= SRAM_rdata(15 downto 0);
      end if;
    end if;
  end process;
  -------------------------------------------------------------------
  --
  -- Data Registers latched from the OPB data bus
  --
  -------------------------------------------------------------------

----- TAMF EXPERIMENT ----

  cs2 <= OPB_select when OPB_ABus(31 downto 16) = X"0180" else '0';
  ce <= (q2 and not q1) or (q0);

  process (OPB_Clk)
  begin
    if OPB_Clk'event and OPB_Clk='1' then
      q2 <= (not q2 and q1) or (q2 and not q1);
      q1 <= (cs2 and not q2 and not q1) or (q2 and not q1);
      q0 <= q2 and not q1;
    end if;
  end process;

  ReadDataFromMemory : process (OPB_Clk, OPB_Rst)
  begin
    if OPB_Rst = '1' then
      data_bus_ce <= X"0000_0000";
    elsif OPB_Clk'event and OPB_Clk = '1' then
      if ce='1' and RNW='1' then
        if RST(0) = '0' then
         data_bus_ce <= DOUT0 & DOUT0 & DOUT0 & DOUT0;
        elsif RST(1) = '0' then
          data_bus_ce <= TERRAIN_DOUTA & TERRAIN_DOUTA & TERRAIN_DOUTA & TERRAIN_DOUTA;
        else
         case ABus(15 downto 0) is
            when X"FFA1"          => data_bus_ce <= "00" & textColor;
            when X"FFA2"          => data_bus_ce <= "00" & X"00000" & gradientStartR;
            when X"FFA3"          => data_bus_ce <= "00" & X"00000" & gradientStartG;
            when X"FFA4"          => data_bus_ce <= "00" & X"00000" & gradientStartB;
            when X"FFA5"          => data_bus_ce <= "0" & X"00000" & gradientIncR;
            when X"FFA6"          => data_bus_ce <= "0" & X"00000" & gradientIncG;
            when X"FFA7"          => data_bus_ce <= "0" & X"00000" & gradientIncB;
            when X"FFB1"          => data_bus_ce <= "00" & X"00000" & LeftTankSprite_x;
            when X"FFB2"          => data_bus_ce <= "00" & X"00000" & LeftTankSprite_y;
            when X"FFB3"          => data_bus_ce <= "00" & X"00000" & RightTankSprite_x;
            when X"FFB4"          => data_bus_ce <= "00" & X"00000" & RightTankSprite_y;
            when X"FFB5"          => data_bus_ce <= "00" & X"00000" & CannonballSprite_x;
            when X"FFB6"          => data_bus_ce <= "00" & X"00000" & CannonballSprite_y;
            when X"FFB7"          => data_bus_ce <= "00" & X"00000" & ExplosionSprite_x;
            when X"FFB8"          => data_bus_ce <= "00" & X"00000" & ExplosionSprite_y;
            when X"FFB9"          => data_bus_ce <= X"0000_000" & WhichSpriteLeftTank;
            when X"FFBA"          => data_bus_ce <= X"0000_000" & WhichSpriteRightTank;
           when X"FFBB"            => data_bus_ce <= "00" & terrainColor;
            when others           => data_bus_ce <= X"0000_0000";
          end case;
       end if;
      else
        data_bus_ce <= X"0000_0000";
      end if;
    end if;
  end process;


  WriteDataToMemory : process (OPB_Clk, OPB_Rst)
  begin
    if OPB_Rst='1' then
      textColor                                                                   <=
(others => '1');
      gradientStartR                                                              <=
(others => '0');
      gradientStartG                                                              <=
(others => '0');
      gradientStartB                    <= (others => '0');
      gradientIncR                      <= (others => '0');
      gradientIncG                      <= (others => '0');
      gradientIncB                      <= (others => '0');
      LeftTankSprite_x                  <= (others => '0');
      LeftTankSprite_y                  <= (others => '0');
      RightTankSprite_x                 <= (others => '0');
      RightTankSprite_y                   <= (others => '0');
      CannonballSprite_x                  <= (others => '0');
      CannonballSprite_y                  <= (others => '0');
      ExplosionSprite_x                   <= (others => '0');
      ExplosionSprite_y                   <= (others => '0');
      WhichSpriteLeftTank                 <= (others => '0');
      WhichSpriteRightTank                <= (others => '0');
      terrainColor                        <= (others => '0');
    elsif OPB_Clk'event and OPB_Clk='1' then
      if ce='1' and rnw='0' then
        case ABus(15 downto 0) is
          when X"FFA1"                     => textColor            <= DBus(29 downto 0);
          when X"FFA2"                     => gradientStartR       <= DBus(9 downto 0);
          when X"FFA3"                     => gradientStartG       <= DBus(9 downto 0);
          when X"FFA4"                     => gradientStartB       <= DBus(9 downto 0);
          when X"FFA5"                     => gradientIncR         <= DBus(10 downto 0);
          when X"FFA6"                     => gradientIncG         <= DBus(10 downto 0);
          when X"FFA7"                     => gradientIncB         <= DBus(10 downto 0);
          when X"FFB1"                     => LeftTankSprite_x     <= DBus(9 downto 0);
          when X"FFB2"                     => LeftTankSprite_y     <= DBus(9 downto 0);
          when X"FFB3"                     => RightTankSprite_x    <= DBus(9 downto 0);
          when X"FFB4"                     => RightTankSprite_y    <= DBus(9 downto 0);
          when X"FFB5"                     => CannonballSprite_x   <= DBus(9 downto 0);
          when X"FFB6"                     => CannonballSprite_y   <= DBus(9 downto 0);
          when X"FFB7"                     => ExplosionSprite_x    <= DBus(9 downto 0);
          when X"FFB8"                     => ExplosionSprite_y    <= DBus(9 downto 0);
          when X"FFB9"                     => WhichSpriteLeftTank  <= DBus(3 downto 0);
          when X"FFBA"                     => WhichSpriteRightTank <= DBus(3 downto 0);
          when X"FFBB"                     => terrainColor         <= DBus(29 downto 0);
          when others                      =>
        end case;
      end if;
    end if;
  end process;

  xfer2 <= q0;
  data_bus <= data_bus_ce when ce='1' else data_bus_rce;
  UIO_DBus <= data_bus;
  UIO_xferAck <= xfer when ce='0' else xfer2;


  -- Unused Slave OPB outputs
  UIO_errAck  <= '0';
  UIO_retry   <= '0';
  UIO_toutSup <= '0';


----- TAMF EXPERIMENT ----

  RamPageAddress <= ABus(15 downto 9);

  RamSelect <=
    "00000001" when RamPageAddress = "0000000" else
    "00000010" when RamPageAddress = "0000001" else
    --"00000100" when RamPageAddress = "010" else
    --"00001000" when RamPageAddress = "011" else
    --"00010000" when RamPageAddress = "100" else
    --"00100000" when RamPageAddress = "101" else
    --"01000000" when RamPageAddress = "110" else
    --"10000000" when RamPageAddress = "111" else
    "00000000";

  WE <=
    RamSelect when cs2 = '1' and RNW = '0' and OPB_Rst = '0' else
    "00000000";

  RST <=
    not RamSelect when cs2 = '1' and RNW = '1' and OPB_Rst = '0' else
    "11111111";

  --------------------------------------------------------------------
  --
  -- Background Gradient Processes
  --
  --------------------------------------------------------------------

  BackgroundGradient : process (pixel_clock, OPB_Rst)
  begin
    if OPB_Rst = '1' then
      backgroundR           <= (others => '0');
      backgroundG           <= (others => '0');
      backgroundB           <= (others => '0');
    elsif pixel_clock'event and pixel_clock = '1' then
      if EndOfLine = '1' then
        if EndOfField = '1' then
          backgroundR       <= gradientStartR;
          backgroundG       <= gradientStartG;
          backgroundB       <= gradientStartB;
        else
          -- RED
          if gradientIncR(10) = '1' then  -- Inc/Dec each line by x
            if gradientIncR(9) = '1' then
              if backgroundR > (backgroundR - not(gradientIncR(9 downto 0) - 1)) then
                backgroundR <= backgroundR - not(gradientIncR(9 downto 0) - 1);
              end if;
            else
              if backgroundR < backgroundR + gradientIncR(9 downto 0) then
                backgroundR <= backgroundR + gradientIncR(9 downto 0);
              end if;
            end if;
          else
            if gradientIncR(9) = '1' then
              if ( ( (not(gradientIncR(9 downto 0) - 1) + 1) = gradientCtrR(9 downto 0) )
and ( (backgroundR-1) < backgroundR ) ) then
                backgroundR <= backgroundR - 1;
              end if;
            elsif ( (gradientIncR(9 downto 0) /= ("00" & X"00")) and ( (backgroundR+1) >
backgroundR ) ) then
              backgroundR   <= backgroundR + 1;
            end if;
          end if;

          -- GREEN
          if gradientIncG(10) = '1' then  -- Inc/Dec each line by x
            if gradientIncG(9) = '1' then
              if backgroundG > (backgroundG - not(gradientIncG(9 downto 0) - 1)) then
                backgroundG <= backgroundG - not(gradientIncG(9 downto 0) - 1);
              end if;
            else
              if backgroundG < backgroundG + gradientIncG(9 downto 0) then
                backgroundG <= backgroundG + gradientIncG(9 downto 0);
              end if;
            end if;
          else
            if gradientIncG(9) = '1' then
              if ( ( (not(gradientIncG(9 downto 0) - 1) + 1) = gradientCtrG(9 downto 0) )
and ( (backgroundG-1) < backgroundG ) ) then
                backgroundG <= backgroundG - 1;
              end if;
            elsif ( (gradientIncG(9 downto 0) /= ("00" & X"00")) and ( (backgroundG+1) >
backgroundG ) ) then
              backgroundG   <= backgroundG + 1;
            end if;
          end if;

          -- BLUE
          if gradientIncB(10) = '1' then  -- Inc/Dec each line by x
            if gradientIncB(9) = '1' then
              if backgroundB > (backgroundB - not(gradientIncB(9 downto 0) - 1)) then
                backgroundB <= backgroundB - not(gradientIncB(9 downto 0) - 1);
              end if;
            else
              if backgroundB < backgroundB + gradientIncB(9 downto 0) then
                backgroundB <= backgroundB + gradientIncB(9 downto 0);
              end if;
            end if;
          else
            if gradientIncB(9) = '1' then
              if ( ( (not(gradientIncB(9 downto 0) - 1) + 1) = gradientCtrB(9 downto 0) )
and ( (backgroundB-1) < backgroundB ) ) then
                backgroundB <= backgroundB - 1;
              end if;
            elsif ( (gradientIncB(9 downto 0) /= ("00" & X"00")) and ( (backgroundB+1) >
backgroundB ) ) then
```

```
            backgroundB   <= backgroundB + 1;
          end if;
        end if;

      end if;
    end if;
  end if;
end process;

BackgroundGradient2 : process (pixel_clock, OPB_Rst)
begin
  if OPB_Rst = '1' then
    gradientCtrR           <= (others => '0');
    gradientCtrG           <= (others => '0');
    gradientCtrB           <= (others => '0');
  elsif pixel_clock'event and pixel_clock = '1' then
    if EndOfLine = '1' then
      if EndOfField = '1' then
        gradientCtrR       <= (others => '0');
        gradientCtrG       <= (others => '0');
        gradientCtrB       <= (others => '0');
      else
        -- RED
        if gradientIncR(10) = '0' then  -- Inc/Dec by 1 when line = x
          if gradientIncR(9) = '1' then
            -- Do subtraction
            -- Get the unsigned version of gradient (which is in 2's compl) by
            -- subtracting 1 then complementing the result
            if ( not(gradientIncR(9 downto 0) - 1) = gradientCtrR(9 downto 0) ) then
              gradientCtrR <= "00" & X"00";
            else
              gradientCtrR <= gradientCtrR + 1;
            end if;
          elsif gradientIncR(9 downto 0) /= ("00" & X"00") then
            -- Do addition
            if ( gradientIncR(9 downto 0) = gradientCtrR(9 downto 0) ) then
              gradientCtrR <= "00" & X"00";
            else
              gradientCtrR <= gradientCtrR + 1;
            end if;
          else
            gradientCtrR   <= gradientCtrR + 1;
          end if;
        end if;

        -- GREEN
        if gradientIncG(10) = '0' then  -- Inc/Dec by 1 when line = x
          if gradientIncG(9) = '1' then
            -- Do subtraction
            -- Get the unsigned version of gradient (which is in 2's compl) by
            -- subtracting 1 then complementing the result
            if ( not(gradientIncG(9 downto 0) - 1) = gradientCtrG(9 downto 0) ) then
              gradientCtrG <= "00" & X"00";
            else
              gradientCtrG <= gradientCtrG + 1;
            end if;
          elsif gradientIncG(9 downto 0) /= ("00" & X"00") then
            -- Do addition
            if ( gradientIncG(9 downto 0) = gradientCtrG(9 downto 0) ) then
              gradientCtrG <= "00" & X"00";
            else
              gradientCtrG <= gradientCtrG + 1;
            end if;
          else
            gradientCtrG   <= gradientCtrG + 1;
          end if;
        end if;

        -- BLUE
        if gradientIncB(10) = '0' then  -- Inc/Dec by 1 when line = x
          if gradientIncB(9) = '1' then
            -- Do subtraction
            -- Get the unsigned version of gradient (which is in 2's compl) by
            -- subtracting 1 then complementing the result
            if ( not(gradientIncB(9 downto 0) - 1) = gradientCtrB(9 downto 0) ) then
              gradientCtrB <= "00" & X"00";
            else
```

```vhdl
              gradientCtrB <= gradientCtrB + 1;
            end if;
          elsif gradientIncB(9 downto 0) /= ("00" & X"00") then
            -- Do addition
            if ( gradientIncB(9 downto 0) = gradientCtrB(9 downto 0) ) then
              gradientCtrB <= "00" & X"00";
            else
              gradientCtrB <= gradientCtrB + 1;
            end if;
          else
            gradientCtrB   <= gradientCtrB + 1;
          end if;
        end if;

      end if;
    end if;
  end if;
end process;



---------------------------------------------------------------------
--
-- Video controller
--
---------------------------------------------------------------------

-- Horizontal and vertical counters

HCounter : process (pixel_clock, OPB_Rst)
begin
  if OPB_Rst = '1' then
    Hcount   <= (others => '0');
  elsif pixel_clock'event and pixel_clock = '1' then
    if EndOfLine = '1' then
      Hcount <= (others => '0');
    else
      Hcount <= Hcount + 1;
    end if;
  end if;
end process HCounter;

EndOfLine <= '1' when Hcount = HTOTAL - 1 else '0';

VCounter : process (pixel_clock, OPB_Rst)
begin
  if OPB_Rst = '1' then
    Vcount     <= (others => '0');
  elsif pixel_clock'event and pixel_clock = '1' then
    if EndOfLine = '1' then
      if EndOfField = '1' then
        Vcount <= (others => '0');
      else
        Vcount <= Vcount + 1;
      end if;
    end if;
  end if;
end process VCounter;

EndOfField <= '1' when Vcount = VTOTAL - 1 else '0';

-- State machines to generate HSYNC, VSYNC, HBLANK, and VBLANK

HSyncGen : process (pixel_clock, OPB_Rst)
begin
  if OPB_Rst = '1' then
    VIDOUT_HSYNC_N   <= '0';
  elsif pixel_clock'event and pixel_clock = '1' then
    if EndOfLine = '1' then
      VIDOUT_HSYNC_N <= '0';
    elsif Hcount = HSYNC - 1 then
      VIDOUT_HSYNC_N <= '1';
    end if;
  end if;
end process HSyncGen;
```

```
  -- The -1 correction doesn't appear here to correct for the
  -- registered video signal outputs.

  HBlankGen : process (pixel_clock, OPB_Rst)
  begin
    if OPB_Rst = '1' then
      HBLANK_N   <= '0';
    elsif pixel_clock'event and pixel_clock = '1' then
      if Hcount = HSYNC + HBACK_PORCH then
        HBLANK_N <= '1';
      elsif Hcount = HSYNC + HBACK_PORCH + HACTIVE then
        HBLANK_N <= '0';
      end if;
    end if;
  end process HBlankGen;

  VSyncGen : process (pixel_clock, OPB_Rst)
  begin
    if OPB_Rst = '1' then
      VIDOUT_VSYNC_N     <= '0';
    elsif pixel_clock'event and pixel_clock = '1' then
      if EndOfLine = '1' then
        if EndOfField = '1' then
          VIDOUT_VSYNC_N <= '0';
        elsif VCount = VSYNC - 1 then
          VIDOUT_VSYNC_N <= '1';
        end if;
      end if;
    end if;
  end process VSyncGen;

  VBlankGen : process (pixel_clock, OPB_Rst)
  begin
    if OPB_Rst = '1' then
      VBLANK_N     <= '0';
    elsif pixel_clock'event and pixel_clock = '1' then
      if EndOfLine = '1' then
        if (Vcount = VSYNC + VBACK_PORCH - 1) then
          VBLANK_N <= '1';
        elsif VCount = VSYNC + VBACK_PORCH + VACTIVE - 1 then
          VBLANK_N <= '0';
        end if;
      end if;
    end if;
  end process VBlankGen;

  Xcoord <= Hcount - HSYNC - HBACK_PORCH;
  Ycoord <= Vcount - VSYNC - VBACK_PORCH;



  ---------------------------------------------------------------------
  --
  -- Text Generation from the Character Buffer and Font Bitmaps
  --
  ---------------------------------------------------------------------

  -- RAM read triggers and shift register control

  LoadChar   <= '1' when Hcount(2 downto 0) = X"5" else '0';
  FontLoad   <= '1' when Hcount(2 downto 0) = X"6" else '0';
  LoadNShift <= '1' when Hcount(2 downto 0) = X"7" else '0';

  -- Correction of 4 needed to calculate the character address before the
  -- character is displayed
  CharColumn <= Hcount - HSYNC - HBACK_PORCH + 4;
  Column     <= CharColumn(9 downto 3);  -- /8
  CharRow    <= Vcount - VSYNC - VBACK_PORCH;
  Row        <= CharRow(4);              -- / 16

  -- Column + Row * 80
  CharAddr <= Column + ("00000" & Row & "000000") + ("0000000" & Row & "0000") when (Row
<= '1') else "000000000000";

  CharRamPage     <= CharAddr(11 downto 9);
  CharRamSelect_N <= "11110" when CharRamPage = "000" else "11111";
  -- Most significant bit of character ignored
  FontAddr(10 downto 4) <= DOUTB0(6 downto 0);
  FontAddr(3 downto 0)  <= CharRow(3 downto 0);

  -- Unusual addressing: font only holds 96 of 128 possible characters
  -- First 32 characters appear twice
  FontRamPage     <= FontAddr(10 downto 9);
  FontRamSelect_N <=
    "110" when FontRamPage = "00" else
    "110" when FontRamPage = "01" else
    "101" when FontRamPage = "10" else
    "011" when FontRamPage = "11" else
    "111";

  FontData <= DOUTB5 or DOUTB6 or DOUTB7;

  -- Shift register
  ShiftRegister : process (pixel_clock, OPB_Rst)
  begin
    if OPB_Rst = '1' then
      ShiftData   <= X"AB";
    elsif pixel_clock'event and pixel_clock = '1' then
      if LoadNShift = '1' then
        ShiftData <= FontData;
      else
        ShiftData <= ShiftData(6 downto 0) & ShiftData(7);
      end if;
    end if;
  end process ShiftRegister;

  TextData <= ShiftData(7);


  ---------------------------------------------------------------------
  --
  -- Terrain Generation
  --
  ---------------------------------------------------------------------

  -- Latch the terrain y-coordiate data from RAMB4_TERRAIN
  terrainHeight <=  "00" & TERRAIN_DOUTB;

  drawTerrainFlag <= '1' when ( (Xcoord >= 0) and (Xcoord < 640) ) and ( Ycoord >= 
(terrainHeight + 250) )
                      else '0';


  ---------------------------------------------------------------------
  --
  -- Sprite Generation
  --
  ---------------------------------------------------------------------

  -- 16 bit shift register control
  LeftTankLoadNShift <= '1' when ((Xcoord = (LeftTankSprite_x)) and
                                  ((Ycoord >= LeftTankSprite_y) and
                                   (Ycoord < (LeftTankSprite_y+sprite_dimY))))
                        else '0';

  RightTankLoadNShift <= '1' when ((Xcoord = (RightTankSprite_x)) and
                                   ((Ycoord >= RightTankSprite_y) and
                                    (Ycoord < (RightTankSprite_y+sprite_dimY))))
                         else '0';

  CannonballLoadNShift <= '1' when ((Xcoord = (CannonballSprite_x)) and
                                    ((Ycoord >= CannonballSprite_y) and
                                     (Ycoord < (CannonballSprite_y+sprite_dimY))))
                          else '0';

  ExplosionLoadNShift <= '1' when ((Xcoord = (ExplosionSprite_x)) and
                                   ((Ycoord >= ExplosionSprite_y) and
                                    (Ycoord < (ExplosionSprite_y+sprite_dimY))))
                         else '0';


  LeftTank_line_toggle <= '1' when ((Ycoord >= LeftTankSprite_y) and
                                    (Ycoord < (LeftTankSprite_y+sprite_dimY)))
                          else '0';

  RightTank_line_toggle <= '1' when ((Ycoord >= RightTankSprite_y) and
                                     (Ycoord < (RightTankSprite_y+sprite_dimY)))
                           else '0';

  Cannonball_line_toggle <= '1' when ((Ycoord >= CannonballSprite_y) and
                                      (Ycoord < (CannonballSprite_y+sprite_dimY)))
                            else '0';

  Explosion_line_toggle <= '1' when ((Ycoord >= ExplosionSprite_y) and
                                     (Ycoord < (ExplosionSprite_y+sprite_dimY)))
                           else '0';


  InLeftTankRange <= '1' when (((Xcoord >= (LeftTankSprite_x)) and
                                (Xcoord < (LeftTankSprite_x+sprite_dimX))) and
                               ((Ycoord >= LeftTankSprite_y) and
                                (Ycoord < (LeftTankSprite_y+sprite_dimY))))
                     else '0';

  InRightTankRange <= '1' when (((Xcoord >= (RightTankSprite_x)) and
                                 (Xcoord < (RightTankSprite_x+sprite_dimX))) and
                                ((Ycoord >= RightTankSprite_y) and
                                 (Ycoord < (RightTankSprite_y+sprite_dimY))))
                      else '0';

  InCannonballRange <= '1' when (((Xcoord >= (CannonballSprite_x)) and
                                  (Xcoord < (CannonballSprite_x+sprite_dimX))) and
                                 ((Ycoord >= CannonballSprite_y) and
                                  (Ycoord < (CannonballSprite_y+sprite_dimY))))
                       else '0';

  InExplosionRange <= '1' when (((Xcoord >= (ExplosionSprite_x)) and
                                 (Xcoord < (ExplosionSprite_x+sprite_dimX))) and
                                ((Ycoord >= ExplosionSprite_y) and
                                 (Ycoord < (ExplosionSprite_y+sprite_dimY))))
                      else '0';

  InLeftTankRangeAddr <= '1' when (((Xcoord >= (LeftTankSprite_x-1)) and
                                    (Xcoord < (LeftTankSprite_x+sprite_dimX))) and
                                   ((Ycoord >= LeftTankSprite_y) and
                                    (Ycoord < (LeftTankSprite_y+sprite_dimY))))
                         else '0';

  InRightTankRangeAddr <= '1' when (((Xcoord >= (RightTankSprite_x-1)) and
                                     (Xcoord < (RightTankSprite_x+sprite_dimX))) and
                                    ((Ycoord >= RightTankSprite_y) and
                                     (Ycoord < (RightTankSprite_y+sprite_dimY))))
                          else '0';

  InCannonballRangeAddr <= '1' when (((Xcoord >= (CannonballSprite_x-1)) and
                                      (Xcoord < (CannonballSprite_x+sprite_dimX))) and
                                     ((Ycoord >= CannonballSprite_y) and
                                      (Ycoord < (CannonballSprite_y+sprite_dimY))))
                           else '0';

  InExplosionRangeAddr <= '1' when (((Xcoord >= (ExplosionSprite_x-1)) and
                                     (Xcoord < (ExplosionSprite_x+sprite_dimX))) and
                                    ((Ycoord >= ExplosionSprite_y) and
                                     (Ycoord < (ExplosionSprite_y+sprite_dimY))))
                          else '0';


  SpriteIncrementer : process (OPB_Rst, Pixel_Clock)
  begin
    if (OPB_Rst = '1') then
      LeftTankFontRow     <= X"0";
      RightTankFontRow    <= X"0";
      CannonballFontRow   <= X"0";
      ExplosionFontRow    <= X"0";
    elsif Pixel_Clock'event and Pixel_Clock = '1' then
      if (LeftTank_line_toggle = '1') and (EndOfLine = '1') then
        LeftTankFontRow   <= LeftTankFontRow + 1;
      elsif (LeftTank_line_toggle = '0') then
        LeftTankFontRow   <= X"0";
      end if;
      if (RightTank_line_toggle = '1') and (EndOfLine = '1') then
        RightTankFontRow  <= RightTankFontRow + 1;
      elsif (RightTank_line_toggle = '0') then
        RightTankFontRow  <= X"0";
      end if;
      if (Cannonball_line_toggle = '1') and (EndOfLine = '1') then
        CannonballFontRow <= CannonballFontRow + 1;
      elsif (Cannonball_line_toggle = '0') then
        CannonballFontRow <= X"0";
      end if;
      if (Explosion_line_toggle = '1') and (EndOfLine = '1') then
        ExplosionFontRow  <= ExplosionFontRow + 1;
      elsif (Explosion_line_toggle = '0') then
        ExplosionFontRow  <= X"0";
      end if;
    end if;
  end process;

  -- Picks correct sprite to display
  SPRITEAddr <= X"B" & ExplosionFontRow when InExplosionRangeAddr = '1'  else
    X"A" & CannonballFontRow when InCannonballRangeAddr = '1' else
    WhichSpriteRightTank & RightTankFontRow when InRightTankRangeAddr = '1'  else
    WhichSpriteLeftTank & LeftTankFontRow   when InLeftTankRangeAddr = '1'   else
    X"C0";


  -- Shift register 16 bit for Sprites
  ShiftRegister16 : process (Pixel_Clock, OPB_Rst)
  begin
    if OPB_Rst = '1' then
      ShiftLeftTankData     <= X"0000";  -- X"AB";
      ShiftRightTankData    <= X"0000";  -- X"AB";
      ShiftCannonballData   <= X"0000";
      ShiftExplosionData    <= X"0000";
    elsif Pixel_Clock'event and Pixel_Clock = '1' then
      if LeftTankLoadNShift = '1' then
        ShiftLeftTankData   <= SPRITE_DOUTB;
      else
        ShiftLeftTankData   <= ShiftLeftTankData(14 downto 0) & "0";
      end if;
      if RightTankLoadNShift = '1' then
        ShiftRightTankData  <= SPRITE_DOUTB;
      else
        ShiftRightTankData  <= "0" & ShiftRightTankData(15 downto 1);
      end if;
      if CannonballLoadNShift = '1' then
        ShiftCannonballData <= SPRITE_DOUTB;
      else
        ShiftCannonballData <= ShiftCannonballData(14 downto 0) & "0";
      end if;
      if ExplosionLoadNShift = '1' then
        ShiftExplosionData  <= SPRITE_DOUTB;
      else
        ShiftExplosionData  <= ShiftExplosionData(14 downto 0) & "0";
      end if;
    end if;
  end process ShiftRegister16;

  VideoTankDataLeft   <= ShiftLeftTankData(15);
  VideoTankDataRight  <= ShiftRightTankData(0);
  VideoCannonballData <= ShiftCannonballData(15);
  VideoExplosionData  <= ShiftExplosionData(15);



  ---------------------------------------------------------------------
  --
  -- Video Output
  --
  ---------------------------------------------------------------------

  -- Registered video signals going to the video DAC
  VideoOut : process (pixel_clock, OPB_Rst)
  begin
    if OPB_Rst = '1' then
      VIDOUT_BLANK_N <= '0';
      r              <= (others => '0');
      g              <= (others => '0');
      b              <= (others => '0');
    elsif pixel_clock'event and pixel_clock = '1' then
      VIDOUT_BLANK_N <= VBLANK_N and HBLANK_N;

      if( (InExplosionRange = '1') and (VideoExplosionData = '1') ) then
        r <= "1111111111";
        g <= "0000000000";
        b <= "1111111111";
      elsif( (InCannonballRange = '1') and (VideoCannonballData = '1') and
             (InExplosionRange = '0') ) then
        r <= "1111111111";
        g <= "1111111111";
        b <= "0000000000";
      elsif( (InLeftTankRange = '1') and (VideoTankDataLeft = '1') and
             (InExplosionRange = '0') and (InCannonballRange = '0') ) then
        r <= "1111111111";
        g <= "0000000000";
        b <= "0000000000";
      elsif( (InRightTankRange = '1') and (VideoTankDataRight = '1') and
             (InExplosionRange = '0') and (InCannonballRange = '0') ) then
        r <= "0000000000";
        g <= "0000000000";
        b <= "1111111111";
      elsif ( TextData = '1' ) and ( Ycoord < 32 ) then
        -- Text
        r <= textColor(29 downto 20);
        g <= textColor(19 downto 10);
        b <= textColor(9 downto 0);
      elsif ( drawTerrainFlag = '1' ) then
        -- Terrain
        r <= terrainColor(29 downto 20);
        g <= terrainColor(19 downto 10);
        b <= terrainColor(9 downto 0);
      else
        -- Gradient background
        r <= backgroundR;
        g <= backgroundG;
        b <= backgroundB;
      end if;
    end if;
  end process VideoOut;


  VIDOUT_RED   <= r;
  VIDOUT_GREEN <= g;
  VIDOUT_BLUE  <= b;

  -- Video clock I/O pad to the DAC
  vidclk : OBUF_F_12 port map (
    O => VIDOUT_CLK,
    I => pixel_clock);

end Behavioral;
```

main.c

```c
#include "xbasic_types.h"
#include "xio.h"

#include "xintc_l.h"
#include "xuartlite_l.h"

#define MEM_BASE_ADDR 0x01800000

#define BRAM_ADDR__CHAR_BUF        (MEM_BASE_ADDR + 0x0000)
#define BRAM_ADDR__TERRAIN_DATA    (MEM_BASE_ADDR + 0x0200)

#define REG_ADDR__TEXT_COLOR       (MEM_BASE_ADDR + 0xFFA1)
#define REG_ADDR__GRADIENT_STARTR  (MEM_BASE_ADDR + 0xFFA2)
#define REG_ADDR__GRADIENT_STARTG  (MEM_BASE_ADDR + 0xFFA3)
#define REG_ADDR__GRADIENT_STARTB  (MEM_BASE_ADDR + 0xFFA4)
#define REG_ADDR__GRADIENT_INCR    (MEM_BASE_ADDR + 0xFFA5)
#define REG_ADDR__GRADIENT_INCG    (MEM_BASE_ADDR + 0xFFA6)
#define REG_ADDR__GRADIENT_INCB    (MEM_BASE_ADDR + 0xFFA7)
#define REG_ADDR__LEFT_TANKX       (MEM_BASE_ADDR + 0xFFB1)
#define REG_ADDR__LEFT_TANKY       (MEM_BASE_ADDR + 0xFFB2)
#define REG_ADDR__RIGHT_TANKX      (MEM_BASE_ADDR + 0xFFB3)
#define REG_ADDR__RIGHT_TANKY      (MEM_BASE_ADDR + 0xFFB4)
#define REG_ADDR__CANNONBALLX      (MEM_BASE_ADDR + 0xFFB5)
#define REG_ADDR__CANNONBALLY      (MEM_BASE_ADDR + 0xFFB6)
#define REG_ADDR__EXPLOSIONX       (MEM_BASE_ADDR + 0xFFB7)
#define REG_ADDR__EXPLOSIONY       (MEM_BASE_ADDR + 0xFFB8)
#define REG_ADDR__WHICH_LEFT_TANK  (MEM_BASE_ADDR + 0xFFB9)
#define REG_ADDR__WHICH_RIGHT_TANK (MEM_BASE_ADDR + 0xFFBA)
#define REG_ADDR__TERRAIN_COLOR    (MEM_BASE_ADDR + 0xFFBB)

/* Address of a particular character on the screen (rows are 80) */
#define CHAR(r,c) \
   (((unsigned char *)(BRAM_ADDR__CHAR_BUF))[(c) + ((r) << 6) + ((r) << 4)])

#define W 640
#define H 480
#define INIT_TERRAIN_COLOR 0x42 << 22 | 0x42 << 12 | 0x42 << 2
#define INIT_TEXT_COLOR 0xff << 22 | 0xfa << 12 | 0xfa << 2
#define WHITE_COLOR 0xff << 22 | 0xfa << 12 | 0xfa << 2

#define INIT_BLUE_X 600
#define INIT_BLUE_Y 102
#define INIT_RED_X 23
#define INIT_RED_Y 103

#define OFF_SCREENX 660
#define OFF_SCREENY 0

#define BLUE_TANK   1
#define RED_TANK    0
#define MAX_TANKS   2
#define INIT_LIFE   1
#define INIT_TURRET_ANGLE 0
#define INIT_VELOCITY 30
#define MIN_VELOCITY 0
#define MAX_VELOCITY 1000
#define LEFTWARD 0
#define RIGHTWARD 1
#define INC_ANGLE 10
#define INC_VELOCITY 5
#define DEATH -1

#define UPI 0x49
#define LEFTJ 0x4A
#define DOWNK 0x4B
#define RIGHTL 0x4C
#define UPi 0x69
#define LEFTj 0x6A
#define DOWNk 0x6B
#define RIGHTl 0x6C
#define SPACE 0x20

#define UPW 0x57
#define LEFTA 0x41
```

```
#define DOWNS 0x53
#define RIGHTD 0x44
#define UPw 0x77
#define LEFTa 0x61
#define DOWNs 0x73
#define RIGHTd 0x64

#define TOGGLE_WIND 0x09

typedef struct tank_struct {
    // Top left point of tank sprite.
    short orig_x;
    short orig_y;

    // Angle of tank's gun
    short turret_angle;
    int velocity;
    int wind;

    char direction;
    char life;
} tank;

tank tanks[MAX_TANKS];
char active = RED_TANK;

short sin_table[92] = {
        0,
        174,
        348,
        523,
        697,
        871,
        1045,
        1218,
        1391,
        1564,
        1736,
        1908,
        2079,
        2249,
        2419,
        2588,
        2756,
        2923,
        3090,
        3255,
        3420,
        3583,
        3746,
        3907,
        4067,
        4226,
        4383,
        4539,
        4694,
        4848,
        4999,
        5150,
        5299,
        5446,
        5591,
        5735,
        5877,
        6018,
        6156,
        6293,
        6427,
        6560,
        6691,
        6819,
        6946,
        7071,
        7193,
        7313,
        7431,
        7547,
        7660,
        7771,
        7880,
        7986,
        8090,
        8191,
        8290,
        8386,
        8480,
        8571,
        8660,
        8746,
        8829,
        8910,
        8987,
        9063,
        9135,
        9205,
        9271,
        9335,
        9396,
        9455,
        9510,
        9563,
        9612,
        9659,
        9702,
        9743,
        9781,
        9816,
        9848,
        9876,
        9902,
        9925,
        9945,
        9961,
        9975,
        9986,
        9993,
        9998,
        10000,
        10000
};


/* Write a text string on the display */
void put_string(char *string, int row, int column)
{
  char *p = &(CHAR(row, column));
  while (*p++ = *string++)
    ;
}

// defined in isr.c
extern void uart_handler(void *callback);
extern int uart_interrupt_count;
extern char uart_character;

/*
 * setup_interrupts: Initialize the interrupt sources and handlers
 *
 * Should be called once when the system starts
 *
 * The main _interrupt_handler() function from Xilinx
 *
 * Saves and restores CPU context, etc.
 *
 * Sees which interrupts are pending, and for each it
 *     acknowledges the interrupt and
 *     calls a user-defined interrupt handler in Xintc_InterruptVectorTable
 *
 * Place interrupt service routines in isr.c to ensure they are placed in
 * the proper memory segment.
 */
void setup_interrupts()
{
  /*
```

```
   * Reset the interrupt controller peripheral
   */

  /* Disable the interrupt signal */
  XIntc_mMasterDisable(XPAR_INTC_SINGLE_BASEADDR);

  /* Disable all interrupt sources */
  XIntc_mEnableIntr(XPAR_INTC_SINGLE_BASEADDR,0);

  /* Acknowledge all possible interrupt sources
     to make sure none are pending */
  XIntc_mAckIntr(XPAR_INTC_SINGLE_BASEADDR, 0xffffffff);

  /*
   * Install the UART interrupt handler
   */

  XIntc_InterruptVectorTable[XPAR_INTC_MYUART_INTERRUPT_INTR].Handler =
    uart_handler;

  /*
   * Enable interrupt sources
   */

  /* Enable CPU interrupts */
  microblaze_enable_interrupts();

  /* Enable interrupts from the interrupt controller */
  XIntc_mMasterEnable(XPAR_INTC_SINGLE_BASEADDR);

  /* Tell the interrupt controller to accept interrupts from the UART */
  XIntc_mEnableIntr(XPAR_INTC_SINGLE_BASEADDR, XPAR_MYUART_INTERRUPT_MASK);

  /* Enable UART interrupt generation */
  XUartLite_mEnableIntr(XPAR_MYUART_BASEADDR);
}

void drawBackground(short R, short G, short B)
{
//************************************************************************
// This function draws the background gradient. It draws the gradient
// starting with black and ending with whatever RGB value is passed to it.
// It then writes the starting RGB as well as an increment value to the
// hardware.
//************************************************************************
  Xuint16 startR, startG, startB;
  Xuint16 endR, endG, endB;

  startR = 0x00 << 2;
  startG = 0x00 << 2;
  startB = 0X00 << 2;

  endR = R << 2;
  endG = G << 2;
  endB = B << 2;

  XIo_Out16(REG_ADDR__GRADIENT_STARTR, startR);
  XIo_Out16(REG_ADDR__GRADIENT_STARTG, startG);
  XIo_Out16(REG_ADDR__GRADIENT_STARTB, startB);


  // RED
  //inc/dec by x each line:
  if ( ( (endR - startR) / 480  >= 1 ) || ( (endR - startR) / 480 <= -1  ) ) {
  // format is: [bit 10: set to 1 if in inc/dec mode by # of lines][9 thru 0: 2's compl
number]
  XIo_Out32(REG_ADDR__GRADIENT_INCR, ( 0x000003FF & (Xint16)((endR - startR) / 480) ) |
0x00000400 );
  }
  //inc/dec by 1 when line = x:
  else {
    XIo_Out32(REG_ADDR__GRADIENT_INCR, 0x000003FF & (Xint16)( 480 / (endR - startR) ) );
  }

  // GREEN
  //inc/dec by x each line:
  if ( ( (endG - startG) / 480  >= 1 ) || ( (endG - startG) / 480 <= -1  ) ) {
    XIo_Out32(REG_ADDR__GRADIENT_INCG, ( 0x000003FF & (Xint16)((endG - startG) / 480) ) |
0x00000400 );
  }
  //inc/dec by 1 when line = x:
  else {
    XIo_Out32(REG_ADDR__GRADIENT_INCG, 0x000003FF & (Xint16)( 480 / (endG - startG) ) );

  }

  // BLUE
  //inc/dec by x each line:
  if ( ( (endB - startB) / 480  >= 1 ) || ( (endB - startB) / 480 <= -1  ) ) {
    XIo_Out32(REG_ADDR__GRADIENT_INCB, ( 0x000003FF & (Xint16)((endB - startB) / 480) ) |
0x00000400 );
  }
  //inc/dec by 1 when line = x:
  else {
    XIo_Out32(REG_ADDR__GRADIENT_INCB, 0x000003FF & (Xint16)( 480 / (endB - startB) ) );
  }

  return;
}


float calc_amplitude(int angle)
{
//*************************************************************************
// This function calculates the amplitude of the terrain bounce.
//*************************************************************************
  float amplitude;
  int real_ang = angle % 360;

  if (real_ang >= 0 && real_ang <= 90) {
    amplitude = sin_table[ real_ang ];
  }
  else if (real_ang > 90 && real_ang <= 180) {
    amplitude = sin_table[ 180 - real_ang ];
  }
  else if (real_ang > 180 && real_ang <= 270) {
    amplitude = -sin_table[ real_ang - 180];
  }
  else if (real_ang > 270 && real_ang <= 360) {
    amplitude = -sin_table[ 360 - real_ang ];
  }

  amplitude /= 10000.00;

  return amplitude;
}


void bounce_terrain(int num_bounce, float height)
{
//*************************************************************************
// This function calculates the height changes of the terrain
// and draws it.  The effect is the terrain bouncing up and down.
//*************************************************************************
  Xuint8 buffer1[320];
  Xuint8 jitter, tmpL, tmpR;
  short x, i;
  int y;

  for (x = 0; x < 320; x++){
      buffer1[x] = XIo_In8(BRAM_ADDR__TERRAIN_DATA + x);
  }

  while(num_bounce > 0){
      for (x = 0; x < 320; x++) {
          y = (int) ((float)buffer1[x] + (calc_amplitude(jitter) * height) );
          if (y < 1) y = 1;
          XIo_Out8(BRAM_ADDR__TERRAIN_DATA + x, y);
          jitter += 1;
      }

      tmpR = XIo_In8(BRAM_ADDR__TERRAIN_DATA + 597/2);
      tmpL = XIo_In8(BRAM_ADDR__TERRAIN_DATA + 26/2);
      XIo_Out16(REG_ADDR__RIGHT_TANKY, tmpR + 250 - 23 );
      XIo_Out16(REG_ADDR__LEFT_TANKY, tmpL + 250 - 20 );
      num_bounce--;

      for(i=0; i < 10000; i++); //delay the bouncing
  }

  for (x = 0; x < 320; x++){
      XIo_Out8(BRAM_ADDR__TERRAIN_DATA + x, buffer1[x]);
  }

  XIo_Out16(REG_ADDR__RIGHT_TANKY, H-INIT_BLUE_Y );
  XIo_Out16(REG_ADDR__LEFT_TANKY, H-INIT_RED_Y );

}

void draw_explosion(short x, short y)
{
//***********************************************************************
// This function draws the explosion sprite at the specified
// (x, y) coordinates.  The explosion effect is performed by
// rapidly moving the explosion on and off the screen.
// At the same time, this function calls the bounce_terrain function
// to bounce the terrain.
//***********************************************************************
  int i, j;

  XIo_Out16(REG_ADDR__CANNONBALLX, OFF_SCREENX);
  XIo_Out16(REG_ADDR__CANNONBALLY, 20);

  for(i=0; i<2000; i++){
      XIo_Out16(REG_ADDR__EXPLOSIONX, x);
      XIo_Out16(REG_ADDR__EXPLOSIONY, H-y);
      for(j=0; j<1000; j++); // delay the explosion
      XIo_Out16(REG_ADDR__EXPLOSIONX, OFF_SCREENX);
      XIo_Out16(REG_ADDR__EXPLOSIONY, OFF_SCREENY);
      XIo_Out32(REG_ADDR__TERRAIN_COLOR, (WHITE_COLOR) + i);

  }
  XIo_Out32(REG_ADDR__TERRAIN_COLOR, INIT_TERRAIN_COLOR);
  bounce_terrain(10, 5.0);
}

void assess_damage(short exp_x, short exp_y)
{
//***********************************************************************
// This function tests to see if the explosion is within the range of
// the tank.  If it is, then set the tank dead and draw explosion.
// The test is true when the x, y coordinates of the cannonball is
// within the x and y coordinates of either tank.
//***********************************************************************
    if(active == RED_TANK) {
        if( ( ( (exp_x) <= tanks[BLUE_TANK].orig_x ) &&
            ( (exp_x) >= tanks[BLUE_TANK].orig_x - 16 ) ) ||
            ( ( ( (exp_x + 8) <= tanks[BLUE_TANK].orig_x ) ) &&
            ( (exp_x + 8) >= tanks[BLUE_TANK].orig_x - 16 ) ) ) {
            if( ( (exp_y ) > tanks[BLUE_TANK].orig_y - 16) &&
                ( (exp_y ) < tanks[BLUE_TANK].orig_y ) ||
               ( ( (exp_y + 8) > tanks[BLUE_TANK].orig_y - 16 ) &&
               ( (exp_y + 8) < tanks[BLUE_TANK].orig_y ) ) ) {
                draw_explosion(exp_x, exp_y);
               tanks[BLUE_TANK].life = DEATH;
                print("BLUE_TANK ****************************** DEATH
*************************\r\n");
            }
        }
       else if( ( ( (exp_x) <= tanks[RED_TANK].orig_x + 8 ) &&
            ( (exp_x) >= tanks[RED_TANK].orig_x ) ) ||
            ( ( ( (exp_x + 8) <= tanks[RED_TANK].orig_x + 16 ) ) &&
            ( (exp_x + 8) >= tanks[RED_TANK].orig_x ) ) ) {
            if( ( (exp_y ) > tanks[RED_TANK].orig_y - 16) &&
                ( (exp_y ) < tanks[RED_TANK].orig_y ) ||
               ( ( (exp_y + 8) > tanks[RED_TANK].orig_y - 16 ) &&
               ( (exp_y + 8) < tanks[RED_TANK].orig_y ) ) ) {
               draw_explosion(exp_x, exp_y);
                tanks[RED_TANK].life = DEATH;
                print("RED_TANK ****************************** SUICIDE!!
*************************\r\n");
           }
        }

    }
    else { // BLUE_TANK is active
       if( ( ( (exp_x) <= tanks[RED_TANK].orig_x + 8 ) &&
           ( (exp_x) >= tanks[RED_TANK].orig_x ) ) ||
           ( ( ( (exp_x + 8) <= tanks[RED_TANK].orig_x + 16 ) ) &&
           ( (exp_x + 8) >= tanks[RED_TANK].orig_x ) ) ) {
           if( ( (exp_y ) > tanks[RED_TANK].orig_y - 16) &&
                ( (exp_y ) < tanks[RED_TANK].orig_y ) ||
               ( ( (exp_y + 8) > tanks[RED_TANK].orig_y - 16 ) &&
               ( (exp_y + 8) < tanks[RED_TANK].orig_y ) ) ) {
               draw_explosion(exp_x, exp_y);
                tanks[RED_TANK].life = DEATH;
                print("RED_TANK ****************************** DEATH
*************************\r\n");
           }
        }
       else if( ( ( (exp_x) <= tanks[BLUE_TANK].orig_x + 10) &&
           ( (exp_x) >= tanks[BLUE_TANK].orig_x ) ) ||
           ( ( ( (exp_x + 8) <= tanks[BLUE_TANK].orig_x ) ) &&
           ( (exp_x + 8) >= tanks[BLUE_TANK].orig_x + 10 ) ) ) {
           if( ( (exp_y ) > tanks[BLUE_TANK].orig_y ) &&
                ( (exp_y ) < tanks[BLUE_TANK].orig_y ) ||
               ( ( (exp_y + 8) > tanks[BLUE_TANK].orig_y - 16 ) &&
               ( (exp_y + 8) < tanks[BLUE_TANK].orig_y ) ) ) {
               draw_explosion(exp_x, exp_y);
                tanks[BLUE_TANK].life = DEATH;
                print("BLUE_TANK ****************************** SUICIDE!!
*************************\r\n");
           }
        }

    }
}

void draw_parabola(short velocity, short angleDegrees, short x_coor, short y_coor, char
direction, int wind)
{
//***********************************************************************
// This function calculates the parabolic path of the cannonball.
// At every calculation of x and y, there is a test to see if the
// cannonball collides with the terrain or a tank.  It calls the
// function assess_damage to test for tank collision.  To test
// for terrain collision, the height of the terrain is read from
// bram and compared to the height of the cannonball.
//***********************************************************************
    float vx0, vy0, t;
    short x, y, readval;

    vx0 = (velocity * (float) (sin_table[90-angleDegrees]/10000.00) ) + wind;
    vy0 = velocity * (float) (sin_table[angleDegrees]/10000.00);

    if(direction == LEFTWARD){
        vx0 *= -1;
        x_coor = x_coor - 17;
       x = x_coor;
       y = y_coor;
    }
    else{
        x_coor = x_coor + 17;
       x = x_coor;
       y = y_coor;
    }

    while(x>0 && x<(W-1) && y>0){
      x = x_coor + vx0*t;
      y = y_coor + vy0*t + (0.5)*(-32.2)*t*t;
      if(y < H) {
          readval = (XIo_In8(BRAM_ADDR__TERRAIN_DATA + (x/2))) + 250;
          if((H-y) >= readval){
              draw_explosion(x, y);
              y = 0;
         }
```

```
          else{
              XIo_Out16(REG_ADDR__CANNONBALLX, x);
              XIo_Out16(REG_ADDR__CANNONBALLY, H-y);
              assess_damage(x, y);
              if(tanks[BLUE_TANK].life == DEATH || tanks[RED_TANK].life == DEATH) y = 0;
          }
      }
      else{
          XIo_Out16(REG_ADDR__CANNONBALLX, OFF_SCREENX);
          XIo_Out16(REG_ADDR__CANNONBALLY, OFF_SCREENY+20);
      }
      t += 0.0005;
    }

    XIo_Out16(REG_ADDR__CANNONBALLX, OFF_SCREENX);
    XIo_Out16(REG_ADDR__CANNONBALLY, OFF_SCREENY+20);

}

void text_display(char player, char angle, short power, int text_color, int wind_value)
{
//*************************************************************************
// This function displays the text at the top of the screen.
// Player 1 is displayed on the left and Player 2 on the right.
// When the control keys are pressed, the values are updated accordingly.
//*************************************************************************
    short hundred, ten, one;
    char number[11] = { '0', '1', '2', '3', '4', '5', '6', '7', '8', '9', '\0' };
    char anglebuf[3];
    char powerbuf[4];
    char windbuf[4];

    anglebuf[2] = '\0';
    powerbuf[3] = '\0';
    windbuf[3] = '\0';

    XIo_Out32(REG_ADDR__TEXT_COLOR, text_color);

    if(player == 0) print("Player 1\r");
    else if(player == 1) print("Player 2\r");

    if(player == 0) {
        put_string(" Player 1", 0, 0);
    //*************************************************************************
    // Player 1's display.
    // Angle is 2 characters max. Power is 3 characters max.
    // Convert's Player 1's integer angle, power to characters to display
    //*************************************************************************
        put_string("Angle = ", 1, 1);
        if(angle > 9) { // display 2 digits
            anglebuf[0] = number[(angle/10)];
           anglebuf[1] = '0';
            put_string(anglebuf, 1, 9);
        }
        else{ // display 1 digit
            anglebuf[0] = number[(angle/10)];
           anglebuf[1] = ' ';
            put_string(anglebuf, 1, 9);
        }

        put_string("Power = ", 1, 12);
        if(power > 99){ // display 3 digits
            hundred = power/100;
           ten = (power - (100 * hundred)) / 10;
           one = (power - (100 * hundred) - (10 * ten));
            powerbuf[0] = number[hundred];
           powerbuf[1] = number[ten];
           powerbuf[2] = number[one];
           put_string(powerbuf, 1, 20);
        }
        else if(power > 9){ // display 2 digits
            ten = power/10;
           one = (power - (10 * ten));
            powerbuf[0] = number[ten];
           powerbuf[1] = number[one];
           powerbuf[2] = ' ';
            put_string(powerbuf, 1, 20);
        }
        else{ // display 1 digit
            powerbuf[0] = number[power];
            powerbuf[1] = ' ';
            powerbuf[2] = ' ';
            put_string(powerbuf, 1, 20);
        }

    }// end if for player 1
    //*************************************************************************
    // Player 2's display.
    // Angle is 2 characters max. Power is 3 characters max.
    // Convert's Player 2's integer angle, power to characters to display
    //*************************************************************************
    else{
        put_string(" Player 2", 0, 70);
        put_string("Angle = ", 1, 58);
        if(angle > 9) { // display 2 digits
            anglebuf[0] = number[(angle/10)];
            anglebuf[1] = '0';
            put_string(anglebuf, 1, 66);
        }
        else{ // display 1 digit
            anglebuf[0] = number[(angle/10)];
            anglebuf[1] = ' ';
            put_string(anglebuf, 1, 66);
        }

        put_string("Power = ", 1, 69);
        if(power > 99){ // display 3 digits
            hundred = power/100;
            ten = (power - (100 * hundred)) / 10;
            one = (power - (100 * hundred) - (10 * ten));
            powerbuf[0] = number[hundred];
            powerbuf[1] = number[ten];
            powerbuf[2] = number[one];
            put_string(powerbuf, 1, 77);
        }
        else if(power > 9){ // display 2 digits
            ten = power/10;
            one = (power - (10 * ten));
            powerbuf[0] = number[ten];
            powerbuf[1] = number[one];
            powerbuf[2] = ' ';
            put_string(powerbuf, 1, 77);
        }
        else{ // display 1 digit
            powerbuf[0] = number[power];
            powerbuf[1] = ' ';
            powerbuf[2] = ' ';
            put_string(powerbuf, 1, 77);
        }
    }// end else for player 2

    //*************************************************************************
    // Display Wind Facter.
    // Wind Facter is 3 characters max.
    //*************************************************************************
    put_string("Wind Factor = ", 1, 30);
    if(wind_value > 99){ // display 3 digits
        hundred = wind_value/100;
        ten = (wind_value - (100 * hundred)) / 10;
        one = (wind_value - (100 * hundred) - (10 * ten));
        windbuf[0] = number[hundred];
        windbuf[1] = number[ten];
        windbuf[2] = number[one];
        put_string(windbuf, 1, 44);
    }
    else if(wind_value > 9){ // display 2 digits
        ten = wind_value/10;
        one = (wind_value - (10 * ten));
        windbuf[0] = number[ten];
        windbuf[1] = number[one];
        windbuf[2] = ' ';
        put_string(windbuf, 1, 44);
    }
    else{ // display 1 digit
```

```
        windbuf[0] = number[wind_value];
       windbuf[1] = ' ';
       windbuf[2] = ' ';
        put_string(windbuf, 1, 44);
    }
}


void end_game()
{
//************************************************************************
// This function is called when someone has won.
// The text is cleared and a winning message is displayed.
// The background is redrawn with the winning tank's color.
// Losing tank disappears.
//************************************************************************
  int x, i;

  // Clear the text on screen
  for(x=0; x<2; x++){
      for(i=0; i<80; i++){
          put_string(" ", x, i);
      }
  }
  // Change text color and draw background color of winning tank
  XIo_Out32(REG_ADDR__TEXT_COLOR, 0x03ff3fff);
  if(tanks[RED_TANK].life == DEATH) {
      XIo_Out16(REG_ADDR__LEFT_TANKX, OFF_SCREENX);
      XIo_Out16(REG_ADDR__LEFT_TANKY, 40);
      for(x=0 ; x<1024 ; x++){
          drawBackground(0xff, 0xff, x);
         for(i=0; i<1000; i++);
      }
      for(i=0; i<2; i++){
          put_string("YOU LOSE SUCKA!! BLUE WINS", i, 28);
      }
  }
  else {
      XIo_Out16(REG_ADDR__RIGHT_TANKX, OFF_SCREENX);
      XIo_Out16(REG_ADDR__RIGHT_TANKY, 60);
      for(x=0 ; x<1024 ; x++){
         drawBackground(x, 0xff, 0xff);
         for(i=0; i<1000; i++);
      }
      for(i=0; i<2; i++){
          put_string("YOU LOSE SUCKA!! RED WINS", i, 28);
      }
  }

}

void init_sprites()
{
//************************************************************************
// This function initializes the parameters for each tank
// Also hides the cannonball and explosion sprites
//************************************************************************
    tanks[RED_TANK].orig_x = INIT_RED_X;
    tanks[RED_TANK].orig_y = INIT_RED_Y;
    tanks[RED_TANK].life = INIT_LIFE;
    tanks[RED_TANK].turret_angle = INIT_TURRET_ANGLE;
    tanks[RED_TANK].direction = RIGHTWARD;
    tanks[RED_TANK].velocity = INIT_VELOCITY;
    tanks[RED_TANK].wind = 0;

    tanks[BLUE_TANK].orig_x = INIT_BLUE_X;
    tanks[BLUE_TANK].orig_y = INIT_BLUE_Y;
    tanks[BLUE_TANK].life = INIT_LIFE;
    tanks[BLUE_TANK].turret_angle = INIT_TURRET_ANGLE;
    tanks[BLUE_TANK].direction = LEFTWARD;
    tanks[BLUE_TANK].velocity = INIT_VELOCITY;
    tanks[BLUE_TANK].wind = 0;

    XIo_Out16(REG_ADDR__LEFT_TANKX, 0);
    XIo_Out16(REG_ADDR__LEFT_TANKY, 100);
    XIo_Out16(REG_ADDR__RIGHT_TANKX, 0);
    XIo_Out16(REG_ADDR__RIGHT_TANKY, 120);
    XIo_Out16(REG_ADDR__CANNONBALLX, OFF_SCREENX);
    XIo_Out16(REG_ADDR__CANNONBALLY, OFF_SCREENY+20);
    XIo_Out16(REG_ADDR__EXPLOSIONX, OFF_SCREENX);
    XIo_Out16(REG_ADDR__EXPLOSIONY, OFF_SCREENY);

}

void draw_redTank(short x_coor, short y_coor, char angle)
{
//***********************************************************************
// This function draws the left tank at (x,y) with the specified angle
//***********************************************************************
  XIo_Out16(REG_ADDR__LEFT_TANKX, x_coor);
  XIo_Out16(REG_ADDR__LEFT_TANKY, (H - y_coor) );
  XIo_Out8(REG_ADDR__WHICH_LEFT_TANK, angle);

}

void draw_blueTank(short x_coor, short y_coor, char angle)
{
//***********************************************************************
// This function draws the right tank at (x,y) with the specified angle
//***********************************************************************
  XIo_Out16(REG_ADDR__RIGHT_TANKX, x_coor);
  XIo_Out16(REG_ADDR__RIGHT_TANKY, (H - y_coor) );
  XIo_Out8(REG_ADDR__WHICH_RIGHT_TANK, angle);

}

void draw_Tank()
{
//*****************************************************************************
// This function tests which for which tank to draw and calculates angle index
//*****************************************************************************
  if(active == BLUE_TANK) {
      draw_blueTank(tanks[active].orig_x, tanks[active].orig_y,
((tanks[active].turret_angle/10) + 48));
  }
  else {
      draw_redTank(tanks[active].orig_x, tanks[active].orig_y,
((tanks[active].turret_angle/10) + 48));
  }
}


int main()
{
  int wind_inc = 0; // random wind counter
  char wind_on = 1;

  // Enable the instruction cache: makes the code run 6 times faster
  //microblaze_enable_icache();
  microblaze_disable_icache();
  setup_interrupts();

  print("\r\nScorched “EARF” XESS!\r\n");
  put_string("SCORCHED “EARF” XESS!!", 0, 28);

  // Set Terrain Color
  XIo_Out32(REG_ADDR__TERRAIN_COLOR, INIT_TERRAIN_COLOR);


//***********************************************************************
// Initialize and draw tanks onto screen
//***********************************************************************
  init_sprites();
  draw_redTank(tanks[RED_TANK].orig_x, tanks[RED_TANK].orig_y,
((tanks[active].turret_angle/10) + 48));
  draw_blueTank(tanks[BLUE_TANK].orig_x, tanks[BLUE_TANK].orig_y,
((tanks[active].turret_angle/10) + 48));


//***********************************************************************
// Display the initial value of each tank
//***********************************************************************
```

```
  text_display(RED_TANK, tanks[RED_TANK].turret_angle, tanks[RED_TANK].velocity,
INIT_TEXT_COLOR, tanks[RED_TANK].wind);
  text_display(BLUE_TANK, tanks[BLUE_TANK].turret_angle, tanks[BLUE_TANK].velocity,
INIT_TEXT_COLOR, tanks[BLUE_TANK].wind);


//***********************************************************************
// GAME STARTS HERE!!!
//***********************************************************************
  for (;;) {

    drawBackground(0xff, 0x85, 0x00); // Set Orange colored background
    XIo_Out32(REG_ADDR__TERRAIN_COLOR, INIT_TERRAIN_COLOR); // Write terrain color to
Register

    if(wind_on == 1){
        // Display text on top of screen for active tank
        text_display(active, tanks[active].turret_angle, tanks[active].velocity,
INIT_TEXT_COLOR, tanks[active].wind);
    }
    else{
        text_display(active, tanks[active].turret_angle, tanks[active].velocity,
INIT_TEXT_COLOR, 0);
    }


//***********************************************************************
// RED Tank's Turn
//***********************************************************************
    if(active == RED_TANK){
      switch(uart_character){
      // MOVE TANK'S ANGLE
      case LEFTA :
      case LEFTa :
       if(active == RED_TANK) {
         tanks[active].turret_angle += INC_ANGLE;
         if(tanks[active].turret_angle > 90) {
           tanks[active].turret_angle -= INC_ANGLE;
         }
       }
       else {
         tanks[active].turret_angle -= INC_ANGLE;
         if(tanks[active].turret_angle < 0) {
           tanks[active].turret_angle += INC_ANGLE;
         }
       }
       draw_Tank();
       if(wind_on == 1)
           text_display(active, tanks[active].turret_angle, tanks[active].velocity,
INIT_TEXT_COLOR, tanks[active].wind);
       else
           text_display(active, tanks[active].turret_angle, tanks[active].velocity,
INIT_TEXT_COLOR, 0);
       uart_character = 0;
      break;

      // MOVE TANK'S ANGLE
      case RIGHTD :
      case RIGHTd :
       if(active == RED_TANK) {
         tanks[active].turret_angle -= INC_ANGLE;
         if(tanks[active].turret_angle < 0) {
           tanks[active].turret_angle += INC_ANGLE;
         }
       }
       else {
         tanks[active].turret_angle += INC_ANGLE;
         if(tanks[active].turret_angle > 90) {
           tanks[active].turret_angle -= INC_ANGLE;
         }
       }
       draw_Tank();
       if(wind_on == 1)
           text_display(active, tanks[active].turret_angle, tanks[active].velocity,
INIT_TEXT_COLOR, tanks[active].wind);
       else
```

```
            text_display(active, tanks[active].turret_angle, tanks[active].velocity,
INIT_TEXT_COLOR, 0);
        uart_character = 0;
      break;

      // INCREMENT POWER
      case UPW :
      case UPw :
        tanks[active].velocity += INC_VELOCITY;
        if(tanks[active].velocity > MAX_VELOCITY) {
          tanks[active].velocity -= INC_VELOCITY;
        }
        if(wind_on == 1)
            text_display(active, tanks[active].turret_angle, tanks[active].velocity,
INIT_TEXT_COLOR, tanks[active].wind);
        else
            text_display(active, tanks[active].turret_angle, tanks[active].velocity,
INIT_TEXT_COLOR, 0);
        uart_character = 0;
      break;

      // DECREMENT POWER
      case DOWNS :
      case DOWNs :
        tanks[active].velocity -= INC_VELOCITY;
        if(tanks[active].velocity < MIN_VELOCITY) {
          tanks[active].velocity += INC_VELOCITY;
        }
        if(wind_on == 1)
            text_display(active, tanks[active].turret_angle, tanks[active].velocity,
INIT_TEXT_COLOR, tanks[active].wind);
        else
            text_display(active, tanks[active].turret_angle, tanks[active].velocity,
INIT_TEXT_COLOR, 0);
        uart_character = 0;
      break;

      // FIRE CANNONBALL
      case SPACE :
         if(wind_on == 1){
            draw_parabola(tanks[active].velocity, tanks[active].turret_angle,
tanks[active].orig_x,
                         tanks[active].orig_y, tanks[active].direction,
tanks[active].wind);
        }
        else{
            draw_parabola(tanks[active].velocity, tanks[active].turret_angle,
tanks[active].orig_x,
                         tanks[active].orig_y, tanks[active].direction, 0);
        }
        uart_character = 0;

        if(active == BLUE_TANK) {
          if(tanks[active].life == DEATH) goto FINISH;
          active = RED_TANK;
          tanks[active].wind = wind_inc;
          if(tanks[active].life == DEATH) goto FINISH;
        }
        else {
          if(tanks[active].life == DEATH) goto FINISH;
          active = BLUE_TANK;
          tanks[active].wind = wind_inc;
          if(tanks[active].life == DEATH) goto FINISH;
        }
      break;

      // Toggle Wind Facter ON/OFF
      case TOGGLE_WIND :
         if(wind_on == 1) wind_on = 0;
        else wind_on = 1;
        uart_character = 0;
      break;

      default :
      break;

      }// end switch
```

```
    }// end if

//*************************************************************************
// BLUE Tank's Turn
//*************************************************************************
    else{
      switch(uart_character){
      // MOVE TANK'S ANGLE
      case LEFTJ :
      case LEFTj :
        if(active == RED_TANK) {
          tanks[active].turret_angle += INC_ANGLE;
          if(tanks[active].turret_angle > 90) {
            tanks[active].turret_angle -= INC_ANGLE;
          }
        }
        else {
          tanks[active].turret_angle -= INC_ANGLE;
          if(tanks[active].turret_angle < 0) {
            tanks[active].turret_angle += INC_ANGLE;
          }
        }
        draw_Tank();
        if(wind_on == 1)
            text_display(active, tanks[active].turret_angle, tanks[active].velocity,
INIT_TEXT_COLOR, tanks[active].wind);
        else
            text_display(active, tanks[active].turret_angle, tanks[active].velocity,
INIT_TEXT_COLOR, 0);
        uart_character = 0;
      break;

      // MOVE TANK'S ANGLE
      case RIGHTL :
      case RIGHTl :
        if(active == RED_TANK) {
          tanks[active].turret_angle -= INC_ANGLE;
          if(tanks[active].turret_angle < 0) {
            tanks[active].turret_angle += INC_ANGLE;
          }
        }
        else {
          tanks[active].turret_angle += INC_ANGLE;
          if(tanks[active].turret_angle > 90) {
            tanks[active].turret_angle -= INC_ANGLE;
          }
        }
        draw_Tank();
        if(wind_on == 1)
            text_display(active, tanks[active].turret_angle, tanks[active].velocity,
INIT_TEXT_COLOR, tanks[active].wind);
        else
            text_display(active, tanks[active].turret_angle, tanks[active].velocity,
INIT_TEXT_COLOR, 0);
        uart_character = 0;
      break;

      // INCREMENT POWER
      case UPI :
      case UPi :
        tanks[active].velocity += INC_VELOCITY;
        if(tanks[active].velocity > MAX_VELOCITY) {
          tanks[active].velocity -= INC_VELOCITY;
        }
        if(wind_on == 1)
            text_display(active, tanks[active].turret_angle, tanks[active].velocity,
INIT_TEXT_COLOR, tanks[active].wind);
        else
            text_display(active, tanks[active].turret_angle, tanks[active].velocity,
INIT_TEXT_COLOR, 0);
        uart_character = 0;
      break;

      // DECREMENT POWER
      case DOWNK :
```

```
      case DOWNk :
       tanks[active].velocity -= INC_VELOCITY;
       if(tanks[active].velocity < MIN_VELOCITY) {
         tanks[active].velocity += INC_VELOCITY;
       }
       if(wind_on == 1)
           text_display(active, tanks[active].turret_angle, tanks[active].velocity,
INIT_TEXT_COLOR, tanks[active].wind);
       else
           text_display(active, tanks[active].turret_angle, tanks[active].velocity,
INIT_TEXT_COLOR, 0);
       uart_character = 0;
      break;

      // FIRE CANNONBALL
      case SPACE :
       if(wind_on == 1){
           draw_parabola(tanks[active].velocity, tanks[active].turret_angle,
tanks[active].orig_x,
                         tanks[active].orig_y, tanks[active].direction,
tanks[active].wind);
       }
       else{
           draw_parabola(tanks[active].velocity, tanks[active].turret_angle,
tanks[active].orig_x,
                         tanks[active].orig_y, tanks[active].direction, 0);
       }
       uart_character = 0;

       if(active == BLUE_TANK) {
         if(tanks[active].life == DEATH) goto FINISH;
         active = RED_TANK;
         tanks[active].wind = wind_inc;
         if(tanks[active].life == DEATH) goto FINISH;

       }
       else {
         if(tanks[active].life == DEATH) goto FINISH;
         active = BLUE_TANK;
         tanks[active].wind = wind_inc;
         if(tanks[active].life == DEATH) goto FINISH;
       }
      break;

      // Toggle Wind Facter ON/OFF
      case TOGGLE_WIND :
        if(wind_on == 1) wind_on = 0;
       else wind_on = 1;
       uart_character = 0;
      break;

      default :
       break;

      }// end switch
    }// end else

    wind_inc++;
    if(wind_inc == 100) wind_inc = 0; // reset wind counter

  }// end infinite for loop


//************************************************************************
// GAME OVER!!
//************************************************************************
  FINISH:
      end_game();
      for (;;) {
          XIo_Out32(REG_ADDR__TERRAIN_COLOR, 0);
          bounce_terrain(10, 10.0);
      }

  return 0;
}
```

```
# Parameters
PARAMETER VERSION = 2.0.0

# Global Ports

PORT PB_A = PB_A, DIR = OUT, VEC = [19:0]
PORT PB_D = PB_D, DIR = INOUT, VEC = [15:0]
PORT PB_LB_N = PB_LB_N, DIR = OUT
PORT PB_UB_N = PB_UB_N, DIR = OUT
PORT PB_WE_N = PB_WE_N, DIR = OUT
PORT PB_OE_N = PB_OE_N, DIR = OUT
PORT RAM_CE_N = RAM_CE_N, DIR = OUT
PORT VIDOUT_CLK = VIDOUT_CLK, DIR = OUT
PORT VIDOUT_HSYNC_N = VIDOUT_HSYNC_N, DIR = OUT
PORT VIDOUT_VSYNC_N = VIDOUT_VSYNC_N, DIR = OUT
PORT VIDOUT_BLANK_N = VIDOUT_BLANK_N, DIR = OUT
PORT VIDOUT_RED = VIDOUT_RED, DIR = OUT, VEC = [9:0]
PORT VIDOUT_GREEN = VIDOUT_GREEN, DIR = OUT, VEC = [9:0]
PORT VIDOUT_BLUE = VIDOUT_BLUE, DIR = OUT, VEC = [9:0]
PORT FPGA_CLK1 = FPGA_CLK1, DIR = IN
PORT RS232_TD = RS232_TD, DIR=OUT
PORT RS232_RD = RS232_RD, DIR=IN
PORT AU_CSN_N =  AU_CSN_N, DIR=OUT
PORT AU_BCLK =  AU_BCLK, DIR=OUT
PORT AU_MCLK = AU_MCLK, DIR=OUT
PORT AU_LRCK = AU_LRCK, DIR=OUT
PORT AU_SDTI = AU_SDTI, DIR=OUT
PORT AU_SDTO0 = AU_SDTO0, DIR=IN

# Sub Components


BEGIN microblaze
 PARAMETER INSTANCE = mymicroblaze
 PARAMETER HW_VER = 2.00.a
 PARAMETER C_USE_BARREL = 1
 PARAMETER C_USE_ICACHE = 0
 PARAMETER C_ADDR_TAG_BITS = 6
 PARAMETER C_CACHE_BYTE_SIZE = 2048
 #PARAMETER C_ICACHE_BASEADDR = 0x00860000
 #PARAMETER C_ICACHE_HIGHADDR = 0x0087FFFF
 PORT Clk = sys_clk
 PORT Reset = fpga_reset
 PORT Interrupt = intr
 BUS_INTERFACE DLMB = d_lmb
 BUS_INTERFACE ILMB = i_lmb
 BUS_INTERFACE DOPB = myopb_bus
 BUS_INTERFACE IOPB = myopb_bus
END

BEGIN opb_intc
 PARAMETER INSTANCE = intc
 PARAMETER HW_VER = 1.00.c
 PARAMETER C_BASEADDR = 0xFFFF0000
 PARAMETER C_HIGHADDR = 0xFFFF00FF
 PORT OPB_Clk = sys_clk
 PORT Intr =  uart_intr
 PORT Irq =   intr
 BUS_INTERFACE SOPB = myopb_bus
END

BEGIN bram_block
 PARAMETER INSTANCE = bram
 PARAMETER HW_VER = 1.00.a
 BUS_INTERFACE PORTA = conn_0
 BUS_INTERFACE PORTB = conn_1
END

BEGIN opb_xsb300
 PARAMETER INSTANCE = xsb300
 PARAMETER HW_VER = 1.00.a
 PARAMETER C_BASEADDR = 0x00800000
 PARAMETER C_HIGHADDR = 0x00FFFFFF
 PORT PB_A = PB_A
 PORT PB_D = PB_D
 PORT PB_LB_N = PB_LB_N
 PORT PB_UB_N = PB_UB_N
 PORT PB_WE_N = PB_WE_N
 PORT PB_OE_N = PB_OE_N
 PORT RAM_CE_N = RAM_CE_N
 PORT OPB_Clk = sys_clk
 PORT pixel_clock = pixel_clock
 PORT VIDOUT_CLK = VIDOUT_CLK
 PORT VIDOUT_HSYNC_N = VIDOUT_HSYNC_N
 PORT VIDOUT_VSYNC_N = VIDOUT_VSYNC_N
 PORT VIDOUT_BLANK_N = VIDOUT_BLANK_N
 PORT VIDOUT_RED = VIDOUT_RED
 PORT VIDOUT_GREEN = VIDOUT_GREEN
 PORT VIDOUT_BLUE = VIDOUT_BLUE
 BUS_INTERFACE SOPB = myopb_bus
END

BEGIN clkgen
 PARAMETER INSTANCE = clkgen_0
 PARAMETER HW_VER = 1.00.a
 PORT FPGA_CLK1 = FPGA_CLK1
 PORT sys_clk = sys_clk
 PORT pixel_clock = pixel_clock
 PORT fpga_reset = fpga_reset
END

BEGIN lmb_lmb_bram_if_cntlr
 PARAMETER INSTANCE = lmb_lmb_bram_if_cntlr_0
 PARAMETER HW_VER = 1.00.a
 PARAMETER C_BASEADDR = 0x00000000
 PARAMETER C_HIGHADDR = 0x00000FFF
 BUS_INTERFACE DLMB = d_lmb
 BUS_INTERFACE ILMB = i_lmb
 BUS_INTERFACE PORTA = conn_0
 BUS_INTERFACE PORTB = conn_1
END

BEGIN opb_uartlite
 PARAMETER INSTANCE = myuart
 PARAMETER HW_VER = 1.00.b
 PARAMETER C_CLK_FREQ = 50_000_000
 PARAMETER C_USE_PARITY = 0
 PARAMETER C_BASEADDR = 0xFEFF0100
 PARAMETER C_HIGHADDR = 0xFEFF01FF
 PORT OPB_Clk = sys_clk
 PORT Interrupt = uart_intr
 BUS_INTERFACE SOPB = myopb_bus
 PORT RX=RS232_RD
 PORT TX=RS232_TD
END

BEGIN opb_v20
 PARAMETER INSTANCE = myopb_bus
 PARAMETER HW_VER = 1.10.a
 PARAMETER C_DYNAM_PRIORITY = 0
 PARAMETER C_REG_GRANTS = 0
 PARAMETER C_PARK = 0
 PARAMETER C_PROC_INTRFCE = 0
 PARAMETER C_DEV_BLK_ID = 0
 PARAMETER C_DEV_MIR_ENABLE = 0
 PARAMETER C_BASEADDR = 0x0fff1000
 PARAMETER C_HIGHADDR = 0x0fff10ff
 PORT SYS_Rst = fpga_reset
 PORT OPB_Clk = sys_clk
END

BEGIN lmb_v10
 PARAMETER INSTANCE = d_lmb
 PARAMETER HW_VER = 1.00.a
 PORT LMB_Clk = sys_clk
 PORT SYS_Rst = fpga_reset
END

BEGIN lmb_v10
 PARAMETER INSTANCE = i_lmb
 PARAMETER HW_VER = 1.00.a
 PORT LMB_Clk = sys_clk
 PORT SYS_Rst = fpga_reset
END
```

opb_xsb_v2_0_0.mpd

```
################################################################################
##
## Copyright (c) 1995-2002 Xilinx, Inc.  All rights reserved.
##
## opb_emc_v2_0_0.mpd
##
## Microprocessor Peripheral Definition
##
################################################################################

PARAMETER VERSION = 2.0.0

BEGIN opb_xsb300, IPTYPE = PERIPHERAL
OPTION IMP_NETLIST = TRUE
OPTION HDL = VHDL
#OPTION CORE_STATE = DEVELOPMENT

# Define bus interface
BUS_INTERFACE BUS=SOPB, BUS_STD=OPB, BUS_TYPE=SLAVE

# Generics for vhdl or parameters for verilog
PARAMETER C_OPB_AWIDTH = 32, DT=integer
PARAMETER C_OPB_DWIDTH = 32, DT=integer
PARAMETER C_BASEADDR = 0xFFFFFFFF, DT=std_logic_vector, MIN_SIZE=0x100, BUS=SOPB
PARAMETER C_HIGHADDR = 0x00000000, DT=std_logic_vector, BUS=SOPB


# Signals
PORT OPB_Clk = "", DIR=IN, BUS=SOPB, SIGIS=CLK
PORT OPB_Rst = OPB_Rst, DIR=IN, BUS=SOPB

# OPB slave signals
PORT OPB_ABus = OPB_ABus, DIR=IN, VEC=[0:C_OPB_AWIDTH-1], BUS=SOPB
PORT OPB_BE = OPB_BE, DIR=IN, VEC=[0:C_OPB_DWIDTH/8-1], BUS=SOPB
PORT OPB_DBus = OPB_DBus, DIR=IN, VEC=[0:C_OPB_DWIDTH-1], BUS=SOPB
PORT OPB_RNW = OPB_RNW, DIR=IN, BUS=SOPB
PORT OPB_select = OPB_select, DIR=IN, BUS=SOPB
PORT OPB_seqAddr = OPB_seqAddr, DIR=IN, BUS=SOPB
PORT UIO_DBus = Sl_DBus, DIR=OUT, VEC=[0:C_OPB_DWIDTH-1], BUS=SOPB
PORT UIO_errAck = Sl_errAck, DIR=OUT, BUS=SOPB
PORT UIO_retry = Sl_retry, DIR=OUT, BUS=SOPB
PORT UIO_toutSup = Sl_toutSup, DIR=OUT, BUS=SOPB
PORT UIO_xferAck = Sl_xferAck, DIR=OUT, BUS=SOPB


PORT PB_A = "", DIR=OUT, VEC=[19:0], IOB_STATE=BUF
PORT PB_LB_N = "", DIR=OUT, IOB_STATE=BUF
PORT PB_UB_N = "", DIR=OUT, IOB_STATE=BUF
PORT PB_D = "", DIR=INOUT, VEC=[15:0], 3STATE=FALSE, IOB_STATE=BUF
PORT PB_WE_N = "", DIR = OUT, IOB_STATE=BUF
PORT PB_OE_N = "", DIR = OUT, IOB_STATE=BUF
PORT RAM_CE_N = "", RAM_CE_N, DIR = OUT, IOB_STATE=BUF

PORT pixel_clock = "", DIR=IN

PORT VIDOUT_CLK = "", DIR=OUT, IOB_STATE=BUF
PORT VIDOUT_RED = "", DIR=OUT, VEC=[9:0]
PORT VIDOUT_GREEN  = "", DIR=OUT, VEC=[9:0]
PORT VIDOUT_BLUE = "", DIR=OUT, VEC=[9:0]
PORT VIDOUT_BLANK_N = "", DIR=OUT
PORT VIDOUT_HSYNC_N = "", DIR=OUT
PORT VIDOUT_VSYNC_N = "", DIR=OUT


END
```

Appendix

0 degrees:

					1	1	1	1	1	1	1	1	1	1	1
			1	1	1	1	1								
			1	1	1	1	1								
		1	1	1	1	1	1	1	1	1	1	1	1		
		1	1	1	1	1	1	1	1	1	1	1	1		
	1	1	1	1	1	1	1	1	1	1	1	1	1	1	
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

10 degrees:

												1	1	1	1
								1	1	1	1				
					1	1	1								
			1	1	1	1	1								
			1	1	1	1	1								
		1	1	1	1	1	1	1	1	1	1	1	1		
		1	1	1	1	1	1	1	1	1	1	1	1		
	1	1	1	1	1	1	1	1	1	1	1	1	1	1	
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

20 degrees:

														1	1
											1	1	1		
									1	1					
							1	1							
					1	1									
			1	1	1	1	1								
			1	1	1	1	1								
		1	1	1	1	1	1	1	1	1	1	1	1		
		1	1	1	1	1	1	1	1	1	1	1	1		
	1	1	1	1	1	1	1	1	1	1	1	1	1	1	
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

30 degrees:

															1
													1	1	
											1	1			
										1					
								1	1						
						1	1								
					1										
			1	1	1	1	1								
			1	1	1	1	1								
		1	1	1	1	1	1	1	1	1	1	1	1		
		1	1	1	1	1	1	1	1	1	1	1	1		
	1	1	1	1	1	1	1	1	1	1	1	1	1	1	
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

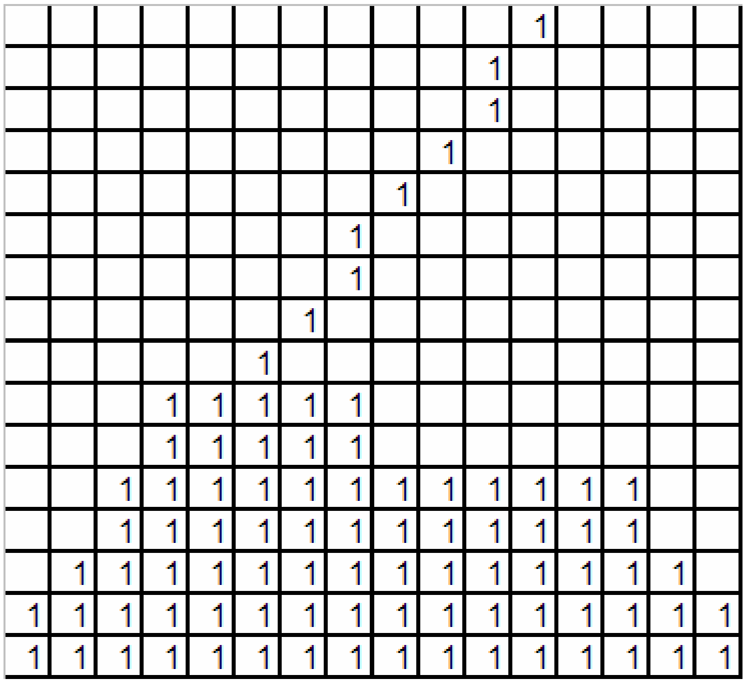
40 degrees:

															1
														1	
													1		
												1			
										1	1				
									1						
							1	1							
						1									
					1										
			1	1	1	1	1								
			1	1	1	1	1								
		1	1	1	1	1	1	1	1	1	1	1	1		
		1	1	1	1	1	1	1	1	1	1	1	1		
	1	1	1	1	1	1	1	1	1	1	1	1	1	1	
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

50 degrees:

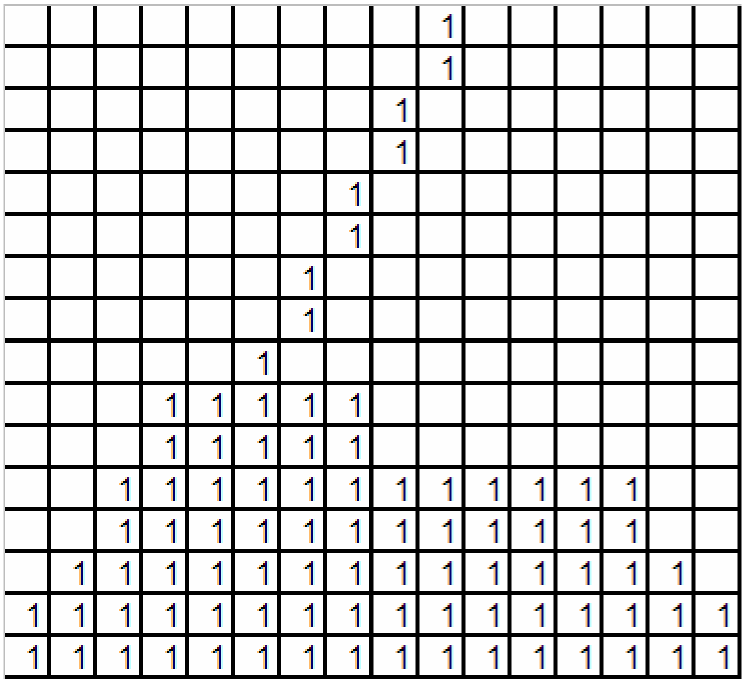
													1		
												1			
											1				
										1					
									1						
								1							
							1								
						1									
					1										
			1	1	1	1	1								
			1	1	1	1	1								
		1	1	1	1	1	1	1	1	1	1	1	1		
		1	1	1	1	1	1	1	1	1	1	1	1		
	1	1	1	1	1	1	1	1	1	1	1	1	1	1	
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

60 degrees:



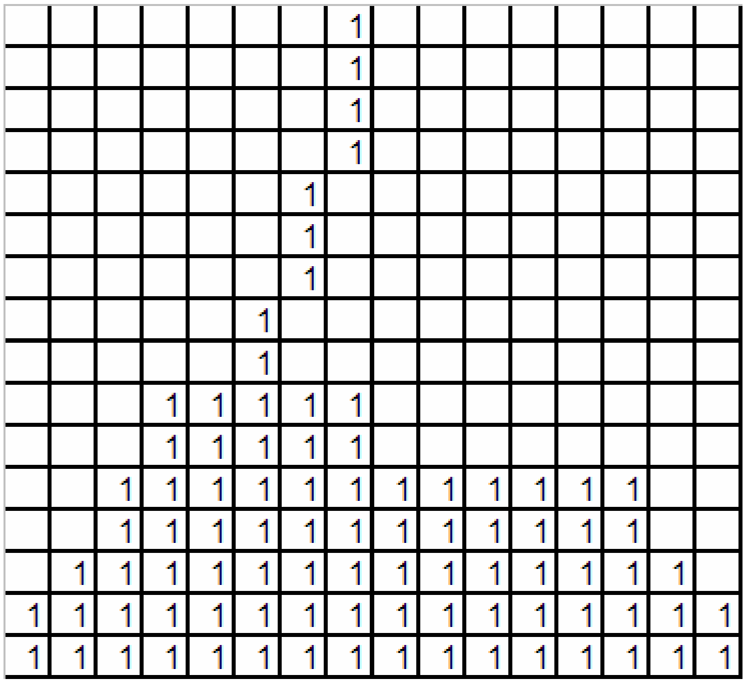
											1				
										1					
										1					
									1						
								1							
							1								
							1								
						1									
					1										
			1	1	1	1	1								
			1	1	1	1	1								
		1	1	1	1	1	1	1	1	1	1	1	1		
		1	1	1	1	1	1	1	1	1	1	1	1		
	1	1	1	1	1	1	1	1	1	1	1	1	1	1	
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

70 degrees:



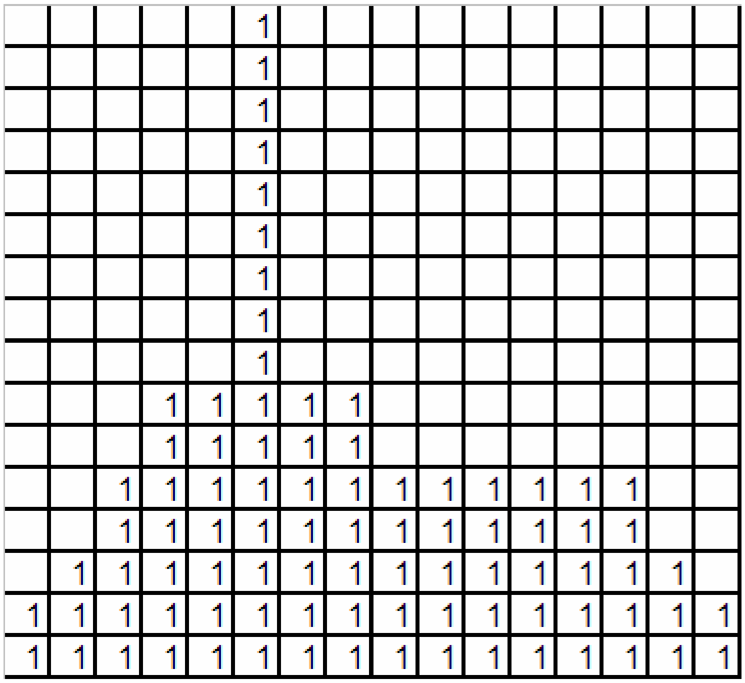
									1						
									1						
								1							
								1							
							1								
							1								
						1									
						1									
					1										
			1	1	1	1	1								
			1	1	1	1	1								
		1	1	1	1	1	1	1	1	1	1	1	1		
		1	1	1	1	1	1	1	1	1	1	1	1		
	1	1	1	1	1	1	1	1	1	1	1	1	1	1	
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

80 degrees:



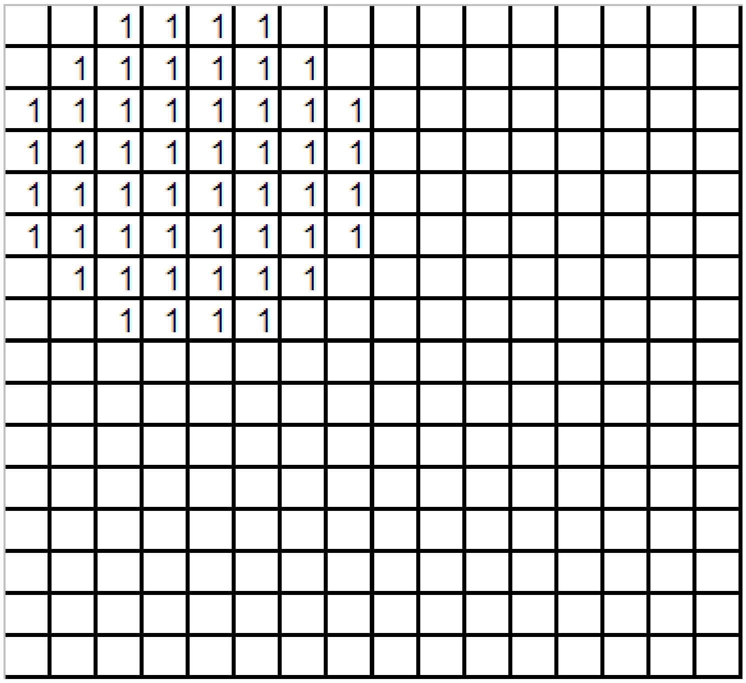
							1								
							1								
							1								
							1								
						1									
						1									
						1									
					1										
					1										
			1	1	1	1	1								
			1	1	1	1	1								
		1	1	1	1	1	1	1	1	1	1	1	1		
		1	1	1	1	1	1	1	1	1	1	1	1		
	1	1	1	1	1	1	1	1	1	1	1	1	1	1	
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

90 degrees:



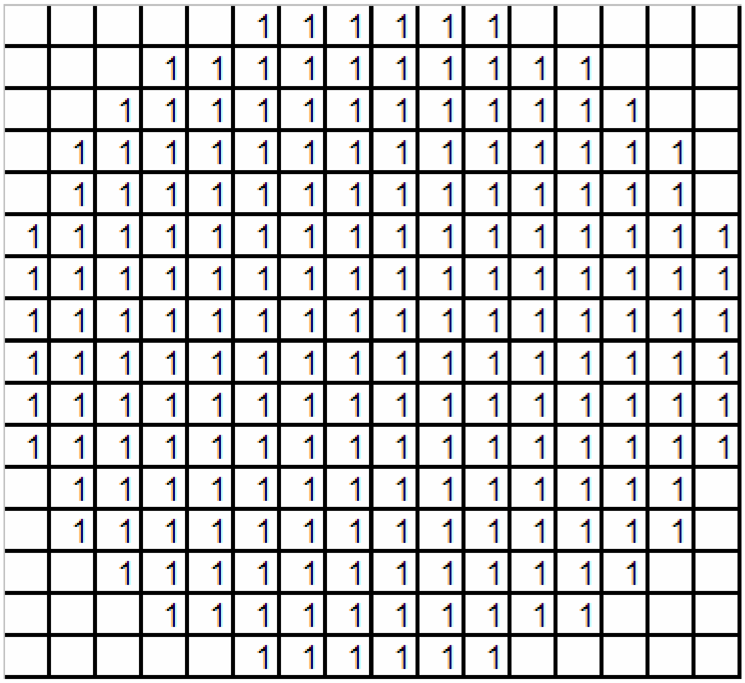
					1										
					1										
					1										
					1										
					1										
					1										
					1										
					1										
					1										
			1	1	1	1	1								
			1	1	1	1	1								
		1	1	1	1	1	1	1	1	1	1	1	1		
		1	1	1	1	1	1	1	1	1	1	1	1		
	1	1	1	1	1	1	1	1	1	1	1	1	1	1	
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Canonball:



		1	1	1	1										
	1	1	1	1	1	1									
1	1	1	1	1	1	1	1								
1	1	1	1	1	1	1	1								
1	1	1	1	1	1	1	1								
1	1	1	1	1	1	1	1								
	1	1	1	1	1	1									
		1	1	1	1										

Explosion:



					1	1	1	1	1	1					
			1	1	1	1	1	1	1	1	1	1			
		1	1	1	1	1	1	1	1	1	1	1	1		
	1	1	1	1	1	1	1	1	1	1	1	1	1	1	
	1	1	1	1	1	1	1	1	1	1	1	1	1	1	
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	1	1	1	1	1	1	1	1	1	1	1	1	1	1	
	1	1	1	1	1	1	1	1	1	1	1	1	1	1	
		1	1	1	1	1	1	1	1	1	1	1	1		
			1	1	1	1	1	1	1	1	1	1			
					1	1	1	1	1	1					