CSEE 4840 – Embedded System Design
Scorched Earth XESS

Project Team Members:
Dennis Chua (dc2036@columbia.edu)
Mike Sumulong (mbs2114@columbia.edu)
Jeremy Chou (jc2465@columbia.edu)
Date: 3/29/2005

Introduction

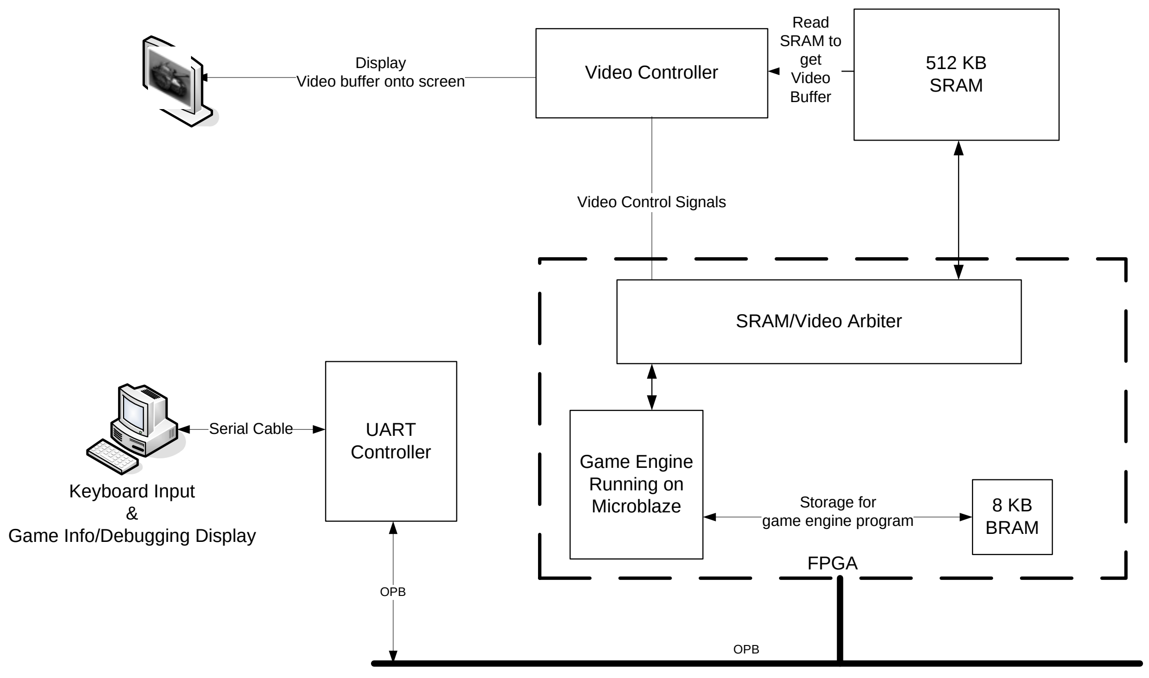
Scorched Earth is a DOS console game belonging to the artillery game genre. It is a turn-based game for two players in which each controls a tank and attempts to destroy the other. When firing a shot, the player adjusts the angle of the ballistic, together with its initial velocity. Both gravity and weather affect the trajectory of the shell. The game provides an assortment of weapons and defenses to aid each tank. This combination of offense and defense weapons, together with simple yet entertaining explosion in VGA, makes for several hours of fun playing Scorched Earth.

In our project, we will take what we have learned in class to implement Scorched Earth on the XESS XSB-300E board. We will be using the FPGA and its components along with integrating VHDL code with C code to recreate this game. The game will take inputs from the keyboard, RAM, and output the graphics onto the video display.

Problem Decomposition

Modules and Task Assignment

We have organized our project into the following three modules: (a) Game Engine & UART I/O; (b) Video Display; (c) Video Buffer I/O.



The *game engine* is the software component of the project. It is coded in C for the XESS MicroBlaze CPU. It handles the artillery calculations, the game mechanics, the graphic drawing and the interaction with the user. The other two modules manage the XESS peripherals.

The *video buffer I/O* stores and retrieves the pixel map of the game's graphics in RAM. Designing this module not only involves dealing with hardware memory, but also devising the geometric layout of the display field together with exposing easy-to-use routines to the C program for pixel manipulation.

The *video display* deals with the video subsystem of the XESS board. Our goal is to display the video buffer contents by means of colored raster scanning. Conforming to the operation of the video buffer I/O and dealing with XESS video hardware will be the main challenge in developing this module.

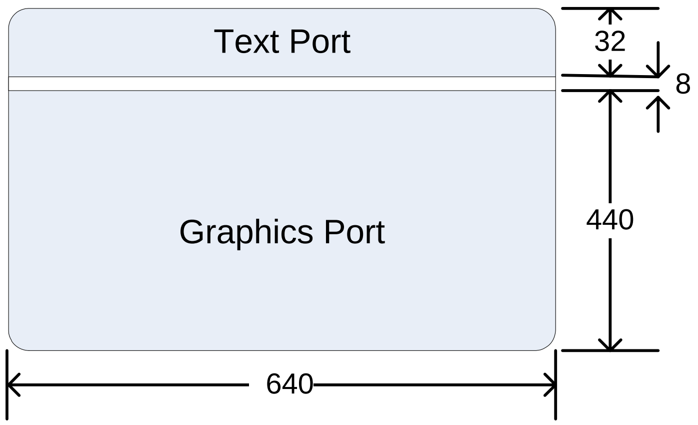
Given this all comes up to a nice number, we've divided the modules among ourselves. Each member of the team will be a "lead programmer" in one of these modules, with the other two providing design and technical support, together with help in troubleshooting. This way, we all have our individual responsibilities; yet every member knows the overall shape of the project and is intimately involved in its development.

Module	**Lead Programmer**
Game Engine & UART	Dennis Chua
Video Display	Jeremy Chou
Video Buffer I/O	Mike Sumulong

Design Constraints

<u>Video Display</u>

The display field will be divided into two regions. At the upper section is a text port where textual messages (such as score) are displayed. Below this is the larger graphics port. The graphics and animations that are central to the game are rendered in this area.



One of our initial concerns was with the amount of memory required by the video buffer. Given that the buffer models the pixel-by-pixel appearance of the graphics display, we identified *display size* and *color depth* as the two overriding factors. Using these, the overall memory size for the video buffer is determined by the following:

Size of Video Buffer = (color depth) x (vertical and horizontal display size)

We considered a few combinations and settled on an **8-bits/pixel color depth for RGB** together with **640 x 480 display size**. This comes up to **a 300 kilo-byte video buffer**, which fits nicely into the off-chip SRAM.

Fonts and Static Graphics

Our next design issue dealt with bitmaps for static graphics such as the tank icons. We chose to represent it by means of an 8x16 graphic. In order to animate the turret movement, we'll use ten of these templates. By loading them into the SRAM, we can display them by means of a look-up operation instead of composing them on the fly. All in all we calculate these **pixel maps to take up 1,280 bits of SRAM**.

We also limited the keyboard input to up-and-down ('i' and 'k' keys) and left-and-right ('j' and 'l' keys) commands. This also cuts back on the textual elements displayed to the video.

We've decided to restrict the range of text displayed to the set of numeric characters. In fact, we've identified twenty-three alphanumeric characters for text display (ex. "0" … "9", "weapon", "score"). Taking the 8x16 fonts for these from Lab #2, **we expect these fonts to take up 2,944 bits of SRAM**. Overall we're guided by the need to conserve space allotted in the SRAM for these static graphics.

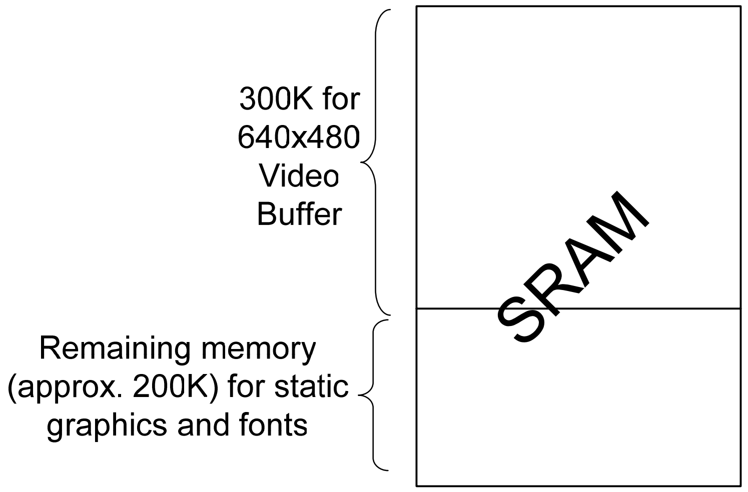
C Program Calculations

The responsiveness of the system was not a major issue. For one, our game is not in real-time; it is turn-based, thus time-critical response is not required. For the same reasons, we're not too concerned with complex trigonometric and exponential calculations and the delay they may introduce. Below is part of the ballistic calculation we expect to perform:

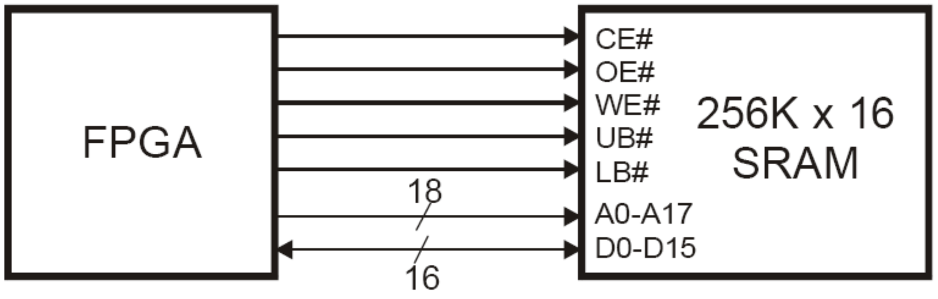
```
pi = 3.14159  -- define pi

angleDegrees = 40  -- shooting angle, in degrees
angleRadians = (angleDegrees/360) * 2 * pi

velocity = 20  -- define the velocity
vx0 = velocity * cos(angleRadians)         -- x-axis velocity component
vy0 = velocity * sin(angleRadians) -- y-axis velocity component

xball = x0 + vx0*t  -- x-coordinate of ballistic at time t
yball = y0 + vy0*t + (1/2)*g*t^2  -- y-coordinate
```

From: http://babek.info/libertybasicfiles/lbnews/nl126/parabolicnally.htm

Video Buffer

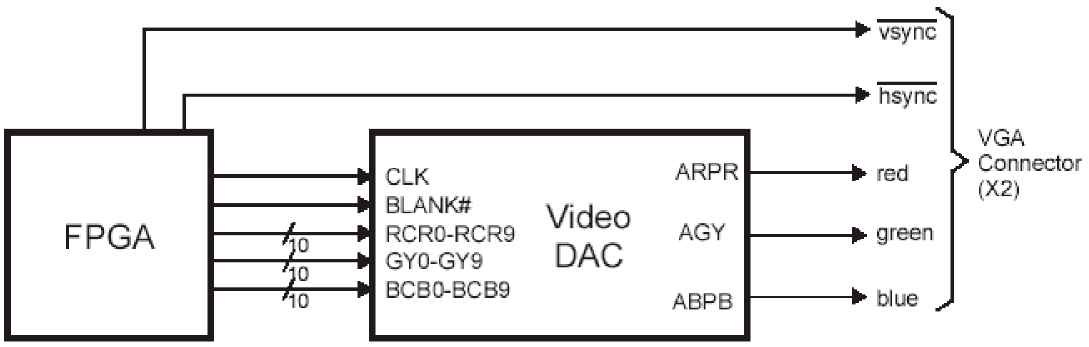


The figure above shows the memory map for the video buffer within the SRAM.



Refer to the Toshiba TC55V16256j SRAM documentation for complete details:
http://www1.cs.columbia.edu/~sedwards/classes/2005/4840/55v16256.pdf

Video Display



Mode	***Resolution***	***Vertical***	***Horizontal***	***Pixel Clock***
VGA	640 x 480	59.94 Hz	31.469 kHz	25.175 MHz

Peripherals Manifest

Below is a list of the XESS components used in this project:

- (XCS300E-6PQ208C) Xilinx Spartan IIE 300-Kgate FPGA.
- (THS8133B) Texas Instruments video D/A converter.
- (TC55V16256j) Toshiba 256K x 16 bit SRAM.