MAZE

A SIMPLE GPU WITH EMBEDDED VIDEO GAME

Designed by:	
Joseph Liang	cjl2104@columbia.edu
Joe Zhang	xz2025@columbia.edu
Wei-Chung Hsu	wh2138@cs.columbia.edu
David Lau	dsl2012@columbia.edu

Table of Contents

1. INTRODUCTION

We implemented a random maze generator. The program generates a random 10x10 maze. The way this was done is to start with all blocks in the 10x10 matrix in separate disjoint sets, then, randomly remove walls until all the blocks are in the same joint set. Each maze will have only one path from the starting block to the ending block.

The user starts at the starting point, and then he can use the arrows to control the direction in which he moves. A square represents the user. When the square reaches a pre-defined block, the program will detect a successful path out of the maze and it terminates.

We designed a simple GPU processor that contains a line drawing. The user would be able to control a graphical object to traverse through a generated maze using the keyboard.

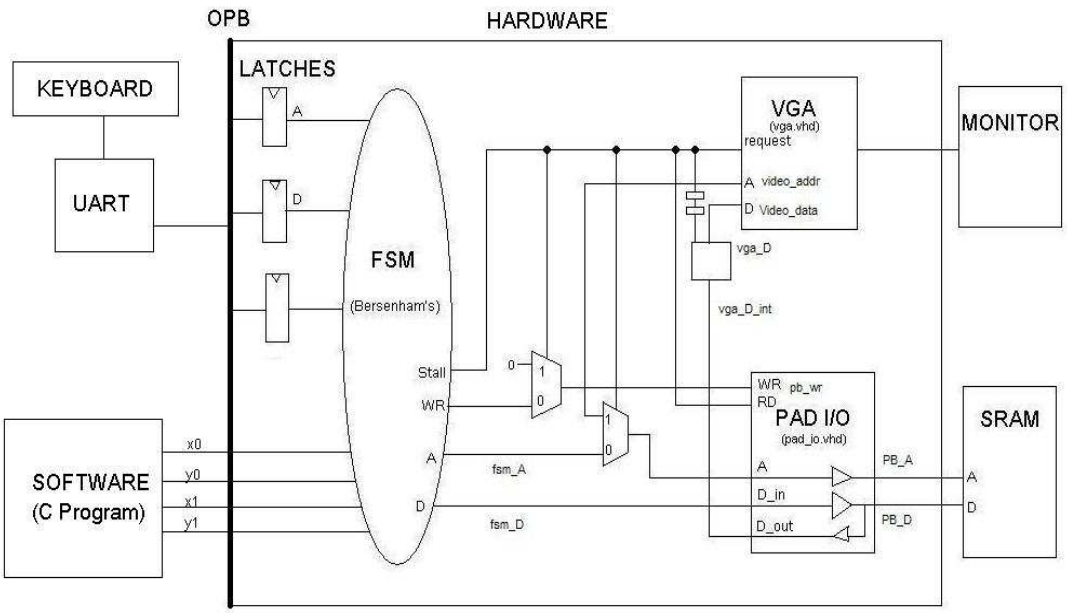
There are 3 parts to our project, (1) hardware functions, (2) firmware/driver/API functions, and (3) software game logic. For our hardware, we utilized UAT to establish a connection between the keyboard and the FPGA. Within the FPGA we implemented the Bernsenham's line drawing algorithm in VHDL, which was translated to an API function and called by our software. We then coded game engine with, which produced signals to communicate with the external Toshiba 512KB SRAM and the Texas Instrument Video D/A converter to display graphics.

2. DESIGN

2.1 Hardware components and Connections

We implemented our video memory into Toshiba 512KB SRAM. For each color frame with 640x480 resolutions, it took 300KB, and this would limit us to implement a double buffer swapping function. The data buffer has both a read and a write feature. Therefore, it could load the game graphics data in the game loading time. We directly imported a lot of the Lab 5 source code in Spring 2004 [1] as it was similar in structure. We need two cycles delay between PAD I/O and VGA, because the VGA comes two cycles later.

The following is a diagram of our data flow:



The FSM in the above diagram is an implementation of bersenham's algorithm. Basically, bersenham's algorithm takes two points , (x0, y0) and (x1, y1) and plots pixels between these points to approximate a line drawn between these two points. Our FSM is an implementation of Bersenham's algorithm, which takes 4 integers from the OPB_bus which represent x0, y0, x1 and y1. In each clock cycle, it produces a new Xout and Yout, which represent the pixel that needs to be drawn in this clock cycle. When it finishes the line, it waits for the next set of points from the OPB bus.

We also implemented a VGA block which interfaces between the monitor and the rest of the blocks. We programmed our FSM to have priority over the VGA block, so we have the stall signal connected to the VGA block. When the stall signal is on, it indicates that the FSM is writing to the SRAM, so the VGA should not be reading from the SRAM at this time! If ths stall signal is not on, then the VGA sends a read request to the PAD I/O block with the address of the data it wishes to access. The PAD I/O block takes the data from this address in the SRAM, and sends it to the VGA. The VGA then writes this data to the screen.

The PAD I/O block services read/writes to the SRAM via the VGA and FSM blocks. The FSM writes data to the SRAM whereas the VGA reads from the SRAM; thus the data line from the PAD I/O block to the SRAM must be a tri-state buffer. The pad I/O has as its inputs an address line and a read/write signal to let it know whether it is writing to the SRAM or reading from it,.

2.2 Maze Generation

Generating a random maze involves manipulating disjoint sets. Disjoint sets, in computer science, are collections of objects that have no common members. They can be represented any basic data structure. The basic idea of random maze generation is to start with as many disjoint sets as there are blocks in the maze, then randomly remove a boundary between two sets and combine the neighboring contents into one set. This process stops until all blocks are determined to be in the same set. Here is the pseudo code for the maze generation algorithm:

```
While (not all blocks are in the same set) {
 Wall = randomly chosen boundary between two sets
If (the neighboring elements are not in the same set)
  Remove wall
     Change their set numbers to the same number
}
```

We used a matrix to represent the maze. Each element of the matrix represents a block in the matrix, and it's a struct that has three members: set number, right wall, bottom wall. The set number denotes the set that the current block is in. The right and left walls denote the boundaries with its right and bottom neighbors. This is an efficient way of representing the information about the maze because it facilitates the process of analyzing and printing the maze on screen.

2.3 Bernsenham Line Drawing Function

We needed to print the maze using hardware. The maze consisted of only straight lines, and therefore we decided to implement a famous hardware line-drawing function called the “Bersenham Line Algorithm”. The algorithm takes the coordinates of two pixels on screen, and would draw a straight line connecting the two pixels.

The following is the pseudo code of Bersenham’s Line Algorithm, taken from wikipedia [2] (http://en.wikipedia.org/wiki/Bresenham's_line_algorithm).

```
function line(x0, x1, y0, y1)
 boolean steep := abs(y1 - y0) - abs(x1 - x0)
 if steep then
        swap(x0, y0)
        swap(x1, y1)
 if x0 > x1 then
        swap(x0, x1)
        swap(y0, y1)
 int deltax := x1 - x0
 int deltay := abs(y1 - y0)
int error := 0
 int y := y0
if y0 < y1 then ystep := 1 else ystep := -1
 for x from x0 to x1
        if steep then plot(y,x) else plot(x,y)
        error := error + deltay
        if 2xerror ≥ deltax
                y := y + ystep
                error := error - deltax
```

The line is drawn between two points (x0, y0) and (x1, y1). It starts from (x0,y0) and starts to go in the down and right directions. It adjusts its “movements” when necessary, by finding an x value between x0 and x0, such that there’s a y value at column x that’s closest to the line. It plots each (x,y) value it finds and this process continues until it reaches (x1,y1).

3. IMPLEMENTATION

3.1 Maze Generation and UART

The maze generator itself is written in C. When the maze is successfully generated, a maze printing function is called, in which it scans each block of the maze and decides which of the boundaries are still existent. If a boundary is found to be existent, it'll be printed on screen by calling the Bresenham function, supplying with it the coordinates of the two pixels.

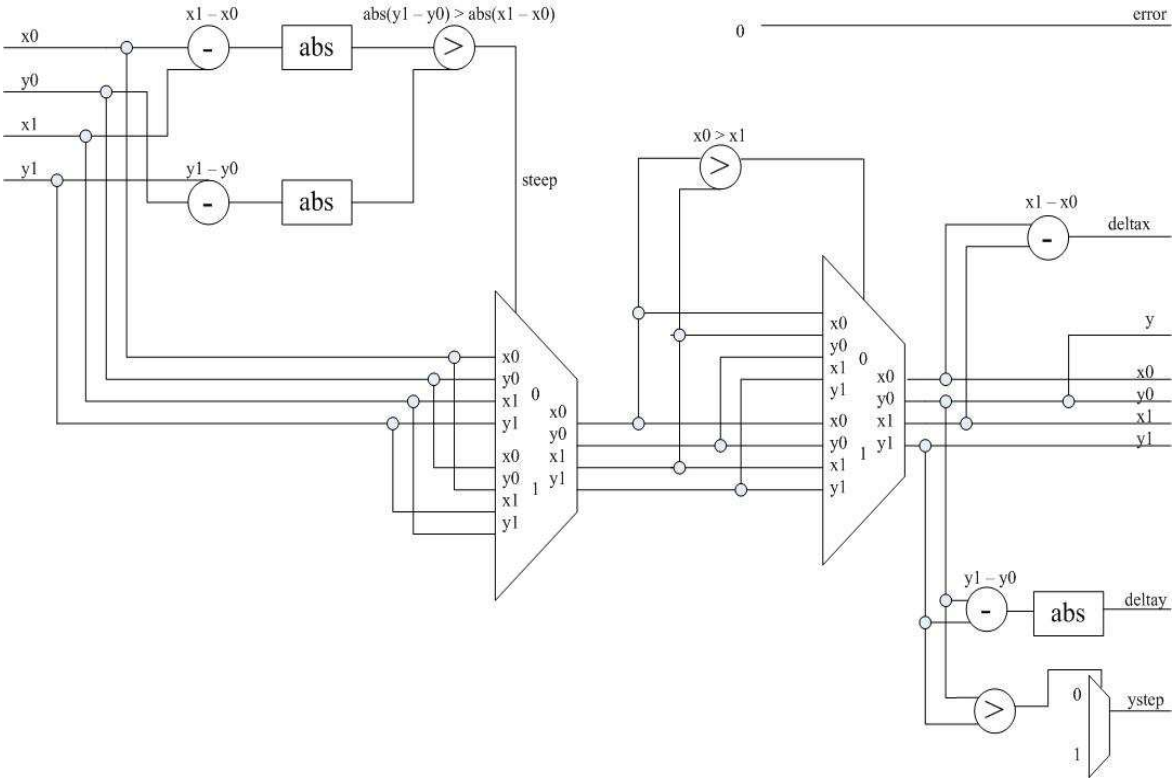
After the maze is printed, the user would be allowed to move a small circle around the maze in an attempt to solve it. The small circle is drawn with lines, each line being generated with the Bresenham function. User input is implemented with UART, which is a type of hardware that translates parallel data and serial data. Using UART, the program can receive a user input from minicom and sends it to the OPB bus, which can then be read by the FPFA. Possible inputs are up, down, left and right. Our C program decides whether a move at a given block is legal, given the boundary information of the adjacent blocks.

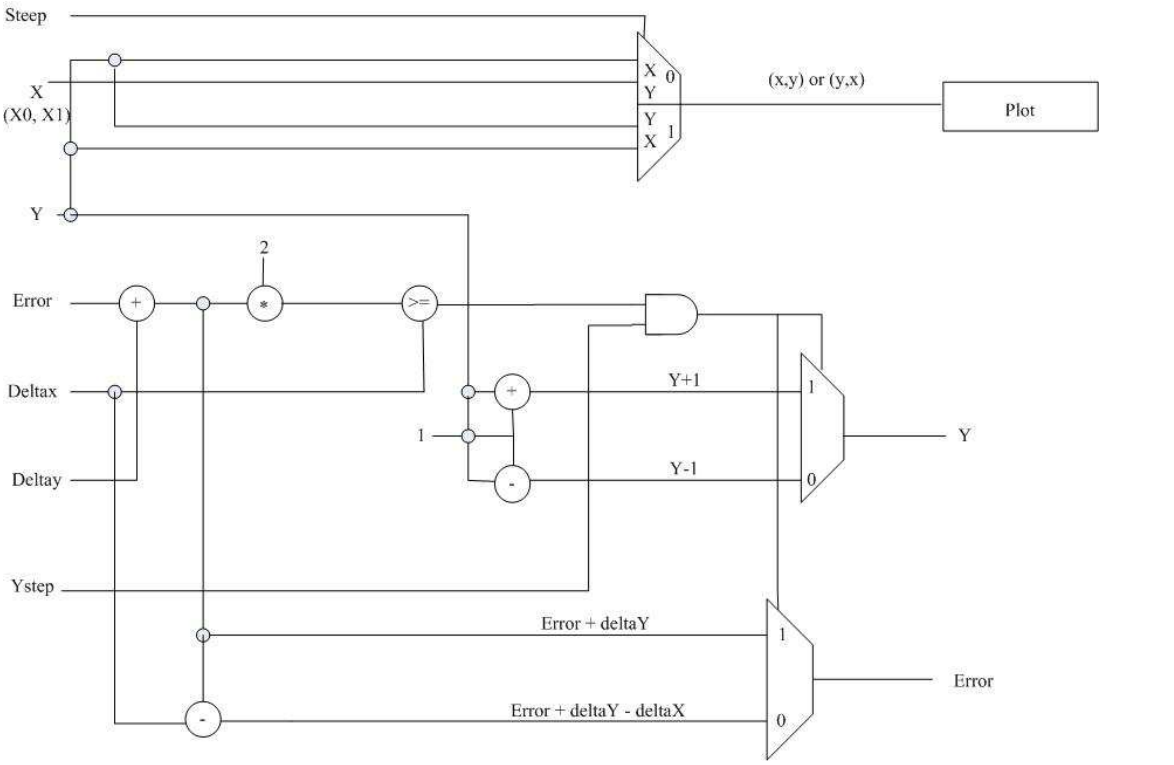
3.2 Hardware Implementation

In order to fit the Bersenham's Line Algorithm into the hardware implementation to overcome the concurrency and negative value, we change the algorithm to the following (the blue code):

```
function line(x0, x1, y0, y1)
 boolean steep := abs(y1 - y0) - abs(x1 - x0)
 if steep then
        swap(x0, y0)
        swap(x1, y1)
 if x0 > x1 then
        swap(x0, x1)
        swap(y0, y1)
 int deltax := x1 - x0
 int deltay := abs(y1 - y0)
 int error := 0
 int y := y0
 if y0 < y1 then ystep := 1 else ystep := 0
 boolean borrow_bit := false
 for x from x0 to x1
        if steep then plot(y,x) else plot(x,y)
        error := error + deltay
        if borrow_bit = true and error > deltax then
                error = error - deltax
                borrow_bit = false
        if 2xerror ≥ deltax
                if ystep then y = y + 1 else y = y - 1
                if error > deltax then error := error - deltax
                else borrow_bit = true
```

The key point in this algorithm is that we use borrow bit method to avoid the negative situation in error value. Then we implement this algorithm in to VHDL code into 9 state/clock-cycle state-machine. The following is our block diagram and VHDL code:


x0
y0
x1
y1
x1 – x0
abs
y1 – y0
abs
abs(y1 – y0) > abs(x1 – x0)
steep
0
error
x0 > x1
x1 – x0
deltax
y
x0
y0
x1
y1
y1 – y0
abs
deltay
ystep


Steep
X
(X0, X1)
Y
(x,y) or (y,x)
Plot
2
Error
Deltax
Deltay
Ystep
Y+1
Y-1
Y
Error + deltaY
Error + deltaY - deltaX
Error

```
if current_state = "0000" then
        if X0 > X1 then
                abs_x1_x0 <= X0 - X1;
        else
                abs_x1_x0 <= X1 - X0;
        end if;
        if Y0 > Y1 then
                abs_y1_y0 <= Y0 - Y1;
        else
                abs_y1_y0 <= Y1 - Y0;
        end if;
        current_state <= "0010";
elsif current_state = "0010" then
        if abs_y1_y0 > abs_x1_x0 then --swap
                steep <= '1';
                x0_prime1 <= Y0;
                y0_prime1 <= X0;
                x1_prime1 <= Y1;
                y1_prime1 <= X1;
                x0_prime2 <= abs_y1_y0;
                abs_y1_y0 <= abs_x1_x0;
                abs_x1_x0 <= x0_prime2;
        else
                steep <= '0';
                x0_prime1 <= X0;
                y0_prime1 <= Y0;
                x1_prime1 <= X1;
                y1_prime1 <= Y1;
```

```
            end if;
            current_state <= "0011";
elsif current_state = "0011" then
            if x0_prime1 > x1_prime1 then
                        x0_prime2 <= x1_prime1;
                        y0_prime2 <= y1_prime1;
                        x1_prime2 <= x0_prime1;
                        y1_prime2 <= y0_prime1;
            else
                        x0_prime2 <= x0_prime1;
                        y0_prime2 <= y0_prime1;
                        x1_prime2 <= x1_prime1;
                        y1_prime2 <= y1_prime1;
            end if;
            current_state <= "0100";
elsif current_state = "0100" then
            deltax <= x1_prime2 - x0_prime2;
            deltay <= abs_y1_y0;
            error_val <= ( others => '0' );
            temp_y <= y0_prime2;
            current_state <= "0101";
elsif current_state = "0101" then
            if y1_prime2 > y0_prime2 then
                        ystep <= '1';
            else
                        ystep <= '0';
            end if;
            temp_x <= x0_prime2;
```

```
            current_state <= "0110";
elsif current_state = "0110" then
            ready_sig <= '0';
            data <= Color;
            write_or_not <= '1';
            low_bit_sig <= '1';
            hi_bit_sig <= '1';
            -- draw pixel
            if steep = '1' then -- (tempy, tempx)
                        address <= temp_y(15 downto 1) + (temp_x(11 downto 0) & "00000000") + (temp_x(13
downto 0) & "000000");
            else -- (tempx, tempy)
                        address <= temp_x(15 downto 1) + (temp_y(11 downto 0) & "00000000") + (temp_y(13
downto 0) & "000000");
            end if;
            if temp_x = x1_prime2 then
                        current_state <= "1111";
            else
                        if borrow_bit = '1' then
                                    y0_prime1 <= error_val + deltay - deltax;
                        end if;
                        error_val <= error_val + deltay;
                        current_state <= "0111";
            end if;
elsif current_state = "0111" then
            if borrow_bit = '1' then
                        if error_val > deltax then -- return borrow
                                    x0_prime1 <= y0_prime1(14 downto 0) & '0';
```

```
                    error_val <= error_val - deltax;
                    borrow_bit <= '0';
                end if;
            else
                x0_prime1 <= error_val(14 downto 0) & '0';
            end if;
            current_state <= "1000";
        elsif current_state = "1000" then
            if x0_prime1 > deltax and borrow_bit = '0' then
                if ystep = '1' then
                    temp_y <= temp_y + 1;
                else
                    temp_y <= temp_y - 1;
                end if;
                if deltax > error_val then
                    borrow_bit <= '1';
                else
                    borrow_bit <= '0';
                    error_val <= error_val - deltax;
                end if;
            end if;
            temp_x <= temp_x +1;
            current_state <= "0110";
        elsif current_state = "1111" then --end state
            current_state <= "0000";
            ready_sig <= '1';
        end if;
```

3.3 Maze Rotation, Transformation and Scale

Originally, we want use 2D Matrix [3] to do the Maze rotation, transformation and scaling. However, it's difficult to directly use matrix operation because of the hardware limit in floating point operations. We need to avoid these operations by only using shift operator. Therefore, we choose 30, 45 degrees rotation in our implementation. For 45 degrees, we did in the following way before really draw the line:

```
x0'=(x0 >> 1) - (y0 >> 1) + shift_x;
y0'=(x0 >> 1) + (y0 >> 1) - shift_y;
x1'=(x0 >> 1) - (y1 >> 1) + shift_x;
y1'=(x0 >> 1) + (y1 >> 1) - shift_y;
draw_line(x0', x1', y0', y1');
```

For 30 degree, it has some tricky way that we need to do x>>1 + x>>2 + x>>3 to approach cos30 = .87 and it will not fit in to the screen for 30 rotation and we need to scale it by shifting 1 bit:

```
x0'=(((x0 >> 1) + (x0 >> 2) + (x0 >> 3) - (y0 >> 1))>>1) + shift_x;
y0'=(((x0 >> 1) + (y0 >> 1) + (y0 >> 2) + (y0 >> 3) )>>1) - shift_y;
x1'=(((x0 >> 1) + (x0 >> 2) + (x0 >> 3) - (y1 >> 1) )>>1) + shift_x;
y1'=(((x0 >> 1) + (y1 >> 1) + (y1 >> 2) + (y1 >> 3) )>>1) - shift_y;
draw_line(x0', x1', y0', y1');
```

3.4 Optimizations

Initially, our C code did not fit onto the 8k memory that was available to us, so we had to optimize our code. Optimization was not an easy task, as it involved figuring out which

parts of our code were taking up the most amount of memory, and what our usage of different memory areas (text, storage, etc) were. Our optimizations consisted of the following:

1) Using short and char instead of int, whenever possible
2) Putting many frequently used lines of code into functions

4. ADVICE FOR FUTURE STUDENTS

Go to all of the Professor and TA office hours- they are very helpful and speed up progress a lot. One piece of useful advice from the professor or TA would save you 2 hours of debugging.

Look at past projects, past labs, and current labs for ideas- it is pretty much impossible to just start from scratch. When we first started, we tried to build everything from scratch, and realized that it was not doable. By borrowing big chunks of code from other labs and projects, it is very easy to create a basic structure of the program.

Draw lots of diagrams and make sure those diagrams are correct, with the help of the TA or professor if needed. Do not start coding without knowing exactly what you want to accomplish.

Set up a regular meeting time, at least once a week. And set up weekly or bi-weekly deliverables, so that you'll be less likely to fall behind. Otherwise you'll find yourself not having finished anything substantial two nights before the due date.

To debug, use the cadence debug tool.

Look at the error dump for helpful debugging info.

5. RESPONSIBILITIES

The responsibilities of the team members were divided as follows:

Wei Chung – Main vhdl coder, , block diagram designer

Chiali Joseph Liang – Assistant to main vhdl coder, co-block diagram designer, translator for main vhdl coder

David Lau and Joe Zhang – software coders, hardware and software testing, documentation

6. SOURCE CODE

data/system.ucf

```
net sys_clk period = 18.000;
net pixel_clock period = 36.000;

net FPGA_CLK1 loc="p77";

net RS232_TD loc="p71";
net RS232_RD loc="p73";

net PB_A<0> loc="p83"; #BAR1
net PB_A<1> loc="p84"; #BAR2
net PB_A<2> loc="p86"; #BAR3
net PB_A<3> loc="p87"; #BAR4
net PB_A<4> loc="p88"; #BAR5
net PB_A<5> loc="p89"; #BAR6
net PB_A<6> loc="p93"; #BAR7
net PB_A<7> loc="p94"; #BAR8
net PB_A<8> loc="p100";
net PB_A<9> loc="p101";
net PB_A<10> loc="p102";
net PB_A<11> loc="p109";
net PB_A<12> loc="p110";
net PB_A<13> loc="p111";
net PB_A<14> loc="p112";
net PB_A<15> loc="p113";
net PB_A<16> loc="p114";
net PB_A<17> loc="p115";
net PB_A<18> loc="p121";
net PB_A<19> loc="p122";

net PB_D<0> loc="p153"; #LEFT_A
net PB_D<1> loc="p145"; #LEFT_B
net PB_D<2> loc="p141"; #LEFT_C
net PB_D<3> loc="p135"; #LEFT_D
net PB_D<4> loc="p126"; #LEFT_E
net PB_D<5> loc="p120"; #LEFT_F
net PB_D<6> loc="p116"; #LEFT_G
net PB_D<7> loc="p108"; #LEFT_DP
net PB_D<8> loc="p127"; #RIGHT_A
net PB_D<9> loc="p129"; #RIGHT_B
net PB_D<10> loc="p132"; #RIGHT_C
net PB_D<11> loc="p133"; #RIGHT_D
net PB_D<12> loc="p134"; #RIGHT_E
net PB_D<13> loc="p136"; #RIGHT_F
net PB_D<14> loc="p138"; #RIGHT_G
net PB_D<15> loc="p139"; #RIGHT_DP


net PB_LB_N loc="p140";  #BAR9
net PB_UB_N loc="p146";  #BAR10
net PB_WE_N loc="p123";
net PB_OE_N loc="p125";
```

```
net RAM_CE_N loc="p147";

net VIDOUT_CLK loc="p23";
net VIDOUT_BLANK_N loc="p24";
net VIDOUT_HSYNC_N loc="p8";
net VIDOUT_VSYNC_N loc="p7";


net VIDOUT_RED<0> loc="p41";
net VIDOUT_RED<1> loc="p40";
net VIDOUT_RED<2> loc="p36";
net VIDOUT_RED<3> loc="p35";
net VIDOUT_RED<4> loc="p34";
net VIDOUT_RED<5> loc="p33";
net VIDOUT_RED<6> loc="p31";
net VIDOUT_RED<7> loc="p30";
net VIDOUT_RED<8> loc="p29";
net VIDOUT_RED<9> loc="p27";

net VIDOUT_GREEN<0> loc="p9" ;
net VIDOUT_GREEN<1> loc="p10";
net VIDOUT_GREEN<2> loc="p11";
net VIDOUT_GREEN<3> loc="p15";
net VIDOUT_GREEN<4> loc="p16";
net VIDOUT_GREEN<5> loc="p17";
net VIDOUT_GREEN<6> loc="p18";
net VIDOUT_GREEN<7> loc="p20";
net VIDOUT_GREEN<8> loc="p21";
net VIDOUT_GREEN<9> loc="p22";

net VIDOUT_BLUE<0> loc="p42";
net VIDOUT_BLUE<1> loc="p43";
net VIDOUT_BLUE<2> loc="p44";
net VIDOUT_BLUE<3> loc="p45";
net VIDOUT_BLUE<4> loc="p46";
net VIDOUT_BLUE<5> loc="p47";
net VIDOUT_BLUE<6> loc="p48";
net VIDOUT_BLUE<7> loc="p49";
net VIDOUT_BLUE<8> loc="p55";
net VIDOUT_BLUE<9> loc="p56";
```

pcores/opb_xsb300e_vga_v1_00_a/data/opb_xsb300e_vga_v2_1_0.mpd

```
BEGIN opb_xsb300e_vga

OPTION IPTYPE = PERIPHERAL
OPTION EDIF=TRUE
OPTION DEVELOPMENT=TRUE

BUS_INTERFACE BUS = SOPB, BUS_STD = OPB, BUS_TYPE = SLAVE

PARAMETER c_baseaddr     = 0xFFFFFFFF, DT = std_logic_vector, MIN_SIZE
= 0xFF
PARAMETER c_highaddr     = 0x00000000, DT = std_logic_vector
PARAMETER c_opb_awidth   = 32,         DT = integer
```

```
PARAMETER c_opb_dwidth   = 32,         DT = integer

## Ports
PORT opb_clk     = "",          DIR = IN,                      BUS =
SOPB
PORT opb_rst     = OPB_Rst,     DIR = IN,
BUS = SOPB
PORT opb_abus    = OPB_ABus,    DIR = IN, VEC = [31:0],     BUS = SOPB
PORT opb_be      = OPB_BE,      DIR = IN, VEC = [3:0], BUS = SOPB
PORT opb_dbus    = OPB_DBus,    DIR = IN, VEC = [31:0],     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 vga_dbus    = Sl_DBus,     DIR = OUT, VEC = [31:0],    BUS = SOPB
PORT vga_errack  = Sl_errAck,   DIR = OUT,
BUS = SOPB
PORT vga_retry   = Sl_retry,    DIR = OUT,
BUS = SOPB
PORT vga_toutsup = Sl_toutSup,  DIR = OUT,
BUS = SOPB
PORT vga_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=[0:9]
PORT VIDOUT_GREEN = "", DIR=OUT, VEC=[0:9]
PORT VIDOUT_BLUE = "", DIR=OUT, VEC=[0:9]
PORT VIDOUT_BLANK_N = "", DIR=OUT
PORT VIDOUT_HSYNC_N = "", DIR=OUT
PORT VIDOUT_VSYNC_N = "", DIR=OUT

END
```

system.mhs

```
# Parameters
 PARAMETER VERSION = 2.1.0


 PORT PB_A = PB_A, VEC = [19:0], DIR = OUT
 PORT PB_D = PB_D, VEC = [15:0], DIR = INOUT
 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 FPGA_CLK1 = FPGA_CLK1, DIR = IN
 PORT RS232_TD = RS232_TD, DIR = OUT
 PORT RS232_RD = RS232_RD, DIR = IN
 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, VEC = [9:0], DIR = OUT
 PORT VIDOUT_GREEN = VIDOUT_GREEN, VEC = [9:0], DIR = OUT
 PORT VIDOUT_BLUE = VIDOUT_BLUE, VEC = [9:0], DIR = OUT


# Video Controller
BEGIN opb_xsb300e_vga
 PARAMETER INSTANCE = vga
 PARAMETER HW_VER = 1.00.a
 PARAMETER C_BASEADDR = 0x00800000
 PARAMETER C_HIGHADDR = 0x00800fff
 BUS_INTERFACE SOPB = myopb_bus
 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_RED = VIDOUT_RED
 PORT VIDOUT_GREEN = VIDOUT_GREEN
 PORT VIDOUT_BLUE = VIDOUT_BLUE
 PORT VIDOUT_BLANK_N = VIDOUT_BLANK_N
 PORT VIDOUT_HSYNC_N = VIDOUT_HSYNC_N
 PORT VIDOUT_VSYNC_N = VIDOUT_VSYNC_N
END

# Interrupt controller for dealing with interrupts from the UART
BEGIN opb_intc
 PARAMETER INSTANCE = intc
 PARAMETER HW_VER = 1.00.c
 PARAMETER C_BASEADDR = 0xFFFF0000
 PARAMETER C_HIGHADDR = 0xFFFF00FF
 BUS_INTERFACE SOPB = myopb_bus
 PORT OPB_Clk = sys_clk
 PORT Intr = uart_intr
 PORT Irq = intr
END

# Block RAM for code and data is connected through two LMB busses
# to the Microblaze, which has two ports on it for just this reason.
# Data LMB bus
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_bram_if_cntlr
 PARAMETER INSTANCE = lmb_data_controller
 PARAMETER HW_VER = 1.00.b
 PARAMETER C_BASEADDR = 0x00000000
 PARAMETER C_HIGHADDR = 0x00001FFF
 BUS_INTERFACE SLMB = d_lmb
 BUS_INTERFACE BRAM_PORT = conn_0
END

# Instruction LMB bus
BEGIN lmb_v10
 PARAMETER INSTANCE = i_lmb
 PARAMETER HW_VER = 1.00.a
 PORT LMB_Clk = sys_clk
 PORT SYS_Rst = fpga_reset
END

BEGIN lmb_bram_if_cntlr
 PARAMETER INSTANCE = lmb_instruction_controller
 PARAMETER HW_VER = 1.00.b
 PARAMETER C_BASEADDR = 0x00000000
 PARAMETER C_HIGHADDR = 0x00001FFF
 BUS_INTERFACE SLMB = i_lmb
 BUS_INTERFACE BRAM_PORT = conn_1
END

# The actual block memory
BEGIN bram_block
 PARAMETER INSTANCE = bram
 PARAMETER HW_VER = 1.00.a
 BUS_INTERFACE PORTA = conn_0
 BUS_INTERFACE PORTB = conn_1
END

# Clock divider to make the whole thing run
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

# The OPB bus controller connected to the Microblaze
# All peripherals are connected to this
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

# UART: Serial port hardware
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
 BUS_INTERFACE SOPB = myopb_bus
 PORT OPB_Clk = sys_clk
 PORT RX = RS232_RD
 PORT TX = RS232_TD
 PORT Interrupt = uart_intr
END

# The main processor core
BEGIN microblaze
 PARAMETER INSTANCE = mymicroblaze
 PARAMETER HW_VER = 2.00.a
 PARAMETER C_USE_BARREL = 1
 PARAMETER C_USE_ICACHE = 0
 BUS_INTERFACE DLMB = d_lmb
 BUS_INTERFACE ILMB = i_lmb
 BUS_INTERFACE DOPB = myopb_bus
 BUS_INTERFACE IOPB = myopb_bus
 PORT Clk = sys_clk
 PORT Reset = fpga_reset
 PORT Interrupt = intr
END
```

pcores/opb_xsb300e_vga_v1_00_a/hdl/vhdl/opb_xsb300e_vga.vhd

```
----------------------------------------------------------------------------
--
-- Component: MUX21
--
----------------------------------------------------------------------------
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

entity mux21 is
  port(
    sel : in std_logic;
    input1, input0 : in std_logic;
    output : out std_logic
  );
```

```
end mux21;

architecture mux21_arch of mux21 is
begin
  process(sel)
    begin
      if sel = '1' then
        output <= input1;
      else
        output <= input0;
      end if;
  end process;
end mux21_arch;

--------------------------------------------------------------------------
--
-- Component: MUX21_20 for 20 bits
--
--------------------------------------------------------------------------
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

entity mux21_20 is
  port(
    sel : in std_logic;
    input1, input0 : in std_logic_vector( 19 downto 0 );
    output : out std_logic_vector( 19 downto 0 )
  );
end mux21_20;

architecture mux21_20_arch of mux21_20 is
begin
  process(sel)
    begin
      if sel = '1' then
        output <= input1;
      else
        output <= input0;
      end if;
  end process;
end mux21_20_arch;

--------------------------------------------------------------------------
--
-- Component: FSM for Bresenham's Algorithm
--
--------------------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity fsm_bresenhams is
   port (
    -- OPB signals
```

```
     X0      : in std_logic_vector (15 downto 0);
     X1      : in std_logic_vector (15 downto 0);
     Y0      : in std_logic_vector (15 downto 0);
     Y1      : in std_logic_vector (15 downto 0);
     Color   : in std_logic_vector (15 downto 0);
     start   : in std_logic;
     ready   : out std_logic;
     OPB_Clk : in std_logic;
     OPB_Rst : in std_logic;
     stall   : in std_logic;
     hi_bit  : out std_logic;
     low_bit : out std_logic;
     wr      : out std_logic;
     A       : out std_logic_vector (19 downto 0);
     D       : out std_logic_vector (15 downto 0)
   );

end fsm_bresenhams;

architecture fsm_bresenhams_arch of fsm_bresenhams is

   signal address : std_logic_vector(19 downto 0):=X"00000";
   signal data : std_logic_vector(15 downto 0):= X"0000";
   signal write_or_not : std_logic:='0';
   signal ready_sig   : std_logic;
   signal hi_bit_sig  : std_logic;
   signal low_bit_sig  : std_logic;
   signal v_or_h : std_logic; -- 1 is vertical, 0 is h
   signal steep  : std_logic;
   signal ystep  : std_logic;
   signal borrow_bit  : std_logic;
   signal temp_x, temp_y, x0_prime1, y0_prime1, x0_prime2, y0_prime2,
x1_prime1, y1_prime1, x1_prime2, y1_prime2, abs_y1_y0, abs_x1_x0,
deltax, deltay, error_val : std_logic_vector (15 downto 0);
   signal current_state : std_logic_vector(3 downto 0):="0000";

begin

   draw: process (OPB_Clk,OPB_Rst)

 begin

         if OPB_Rst = '1' then
          data <= ( others => '0');
          address <= ( others => '0');
        ready_sig <= '0';
        current_state <= "0000";
        write_or_not <= '0';

        borrow_bit <= '0';
        elsif OPB_Clk'event and OPB_Clk = '1' and stall = '0' then
            write_or_not <= '0';
            if start = '1' then
                  if current_state = "0000" then
                        if X0 > X1 then
                              abs_x1_x0 <= X0 - X1;
                        else
```

```
            abs_x1_x0 <= X1 - X0;
      end if;
      if Y0 > Y1 then
            abs_y1_y0 <= Y0 - Y1;
      else
            abs_y1_y0 <= Y1 - Y0;
      end if;
      current_state <= "0010";
elsif current_state = "0010" then
      if abs_y1_y0 > abs_x1_x0 then --swap
            steep <= '1';
            x0_prime1 <= Y0;
            y0_prime1 <= X0;
            x1_prime1 <= Y1;
            y1_prime1 <= X1;
            x0_prime2 <= abs_y1_y0;
            abs_y1_y0 <= abs_x1_x0;
            abs_x1_x0 <= x0_prime2;
      else
            steep <= '0';
            x0_prime1 <= X0;
            y0_prime1 <= Y0;
            x1_prime1 <= X1;
            y1_prime1 <= Y1;
      end if;
      current_state <= "0011";
elsif current_state = "0011" then
      if x0_prime1 > x1_prime1 then
            x0_prime2 <= x1_prime1;
            y0_prime2 <= y1_prime1;
            x1_prime2 <= x0_prime1;
            y1_prime2 <= y0_prime1;
      else
            x0_prime2 <= x0_prime1;
            y0_prime2 <= y0_prime1;
            x1_prime2 <= x1_prime1;
            y1_prime2 <= y1_prime1;
      end if;
      current_state <= "0100";
elsif current_state = "0100" then
      deltax <= x1_prime2 - x0_prime2;
      deltay <= abs_y1_y0;
      error_val <= ( others => '0' );
      temp_y <= y0_prime2;
      current_state <= "0101";
elsif current_state = "0101" then
      if y1_prime2 > y0_prime2 then
            ystep <= '1';
      else
            ystep <= '0';
      end if;
      temp_x <= x0_prime2;
      current_state <= "0110";
elsif current_state = "0110" then
      ready_sig <= '0';
      data <= Color;
      write_or_not <= '1';
```

```
                        low_bit_sig <= '1';
                        hi_bit_sig <= '1';
                        -- draw pixel
                        if steep = '1' then -- (tempy, tempx)
                              address <= temp_y(15 downto 1) +
(temp_x(11 downto 0) & "00000000") + (temp_x(13 downto 0) & "000000");
                        else -- (tempx, tempy)
                              address <= temp_x(15 downto 1) +
(temp_y(11 downto 0) & "00000000") + (temp_y(13 downto 0) & "000000");
                        end if;

                        if temp_x = x1_prime2 then
                              current_state <= "1111";
                        else
                              if borrow_bit = '1' then
                                    y0_prime1 <= error_val + deltay -
deltax;
                              end if;
                              error_val <= error_val + deltay;
                              current_state <= "0111";
                        end if;
                  elsif current_state = "0111" then

                        if borrow_bit = '1' then
                              if error_val > deltax then -- return
borrow
                                    x0_prime1 <= y0_prime1(14 downto 0)
& '0';
                                    error_val <= error_val - deltax;
                                    borrow_bit <= '0';
                              end if;
                        else
                              --temp_val <= error_val(15 downto 0) &
'0';
                             x0_prime1 <= error_val(14 downto 0) & '0';
                        end if;
                        current_state <= "1000";
                  elsif current_state = "1000" then
                        if x0_prime1 > deltax and borrow_bit = '0' then
                              if ystep = '1' then
                                    temp_y <= temp_y + 1;
                              else
                                    temp_y <= temp_y - 1;
                              end if;
                              --error_val <= error_val - deltax;
                              if deltax > error_val then
                                    borrow_bit <= '1';
                              else
                                    borrow_bit <= '0';
                                    error_val <= error_val - deltax;
                              end if;
                        end if;
                        temp_x <= temp_x +1;
                        current_state <= "0110";
                  elsif current_state = "1111" then --end state
                        current_state <= "0000";
                        ready_sig <= '1';
```

```
                end if;
            else
                low_bit_sig <= '1';
                hi_bit_sig <= '1';
                current_state <= "0000";
                ready_sig <= '0';
            end if;

            wr <= write_or_not;
            A<=address;
            D<=data;
            hi_bit <= hi_bit_sig;
            low_bit <= low_bit_sig;
        end if;

  end process draw;
  ready <= ready_sig;

end fsm_bresenhams_arch;

--------------------------------------------------------------------------------
--
-- Text-mode VGA controller for the XESS-300E
--
-- Uses an OPB interface, e.g., for use with the Microblaze soft core
--
-- Stephen A. Edwards
-- sedwards@cs.columbia.edu
--
--------------------------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity opb_xsb300e_vga is

  generic (
    C_OPB_AWIDTH : integer                   := 32;
    C_OPB_DWIDTH : integer                   := 32;
    --C_BASEADDR   : std_logic_vector(31 downto 0) := X"FEFF1000";
    --C_HIGHADDR   : std_logic_vector(31 downto 0) := X"FEFF1FFF"
    C_BASEADDR   : std_logic_vector(31 downto 0) := X"00800000";
    C_HIGHADDR   : std_logic_vector(31 downto 0) := X"00800FFF"
  );

  port (
    -- OPB signals
    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;
```

```
    VGA_DBus       : out std_logic_vector (31 downto 0);
    VGA_errAck     : out std_logic;
    VGA_retry      : out std_logic;
    VGA_toutSup    : out std_logic;
    VGA_xferAck    : out std_logic;

    Pixel_Clock    : in std_logic;

    -- PAD IO Output signals
    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);

    -- Video Output signals
    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
  );

end opb_xsb300e_vga;

architecture Behavioral of opb_xsb300e_vga is

  component vga
    port (
      clk : in std_logic;
      pix_clk : in std_logic;
      rst : in std_logic;
      video_data : in std_logic_vector(15 downto 0);
      video_addr : out std_logic_vector(19 downto 0);
      video_req : out std_logic;
      vidout_clk : out std_logic;
      vidout_RCR : out std_logic_vector(9 downto 0);
      vidout_GY : out std_logic_vector(9 downto 0);
      vidout_BCB : 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
    );
  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;

  component mux21
    port (
      sel : in std_logic;
      input1, input0 : in std_logic;
      output : out std_logic
    );
  end component;

  component mux21_20
    port (
      sel : in std_logic;
      input1, input0 : in std_logic_vector( 19 downto 0 );
      output : out std_logic_vector( 19 downto 0 )
    );
  end component;

  component fsm_bresenhams
    port (
      X0      : in std_logic_vector (15 downto 0);
      X1      : in std_logic_vector (15 downto 0);
      Y0      : in std_logic_vector (15 downto 0);
      Y1      : in std_logic_vector (15 downto 0);
      Color   : in std_logic_vector (15 downto 0);
      start   : in std_logic;
      ready   : out std_logic;
      OPB_Clk : in std_logic;
      OPB_Rst : in std_logic;

      stall   : in std_logic;
      hi_bit  : out std_logic;
      low_bit : out std_logic;
      wr      : out std_logic;
      A       : out std_logic_vector (19 downto 0);
      D       : out std_logic_vector (15 downto 0)
    );
  end component;

------------------------------------------------------------------------
--
-- OPB-RAM controller
--
------------------------------------------------------------------------
```

```
 --constant BASEADDR : std_logic_vector(31 downto 0) := X"FEFF1000";
 constant BASEADDR : std_logic_vector(31 downto 0) := X"00800000";

 -- Signals for OPB
 signal opb_A, opb_D : std_logic_vector(31 downto 0);
 signal opb_RW : std_logic;

 -- Signals for FSM
 signal fsm_WR : std_logic;
 signal fsm_A : std_logic_vector(19 downto 0);
 signal fsm_D : std_logic_vector(15 downto 0);


 -- Signals for VGA
 signal vga_A : std_logic_vector(19 downto 0);
 signal vga_D : std_logic_vector(15 downto 0);
 signal vga_request : std_logic;

 -- Signals for the MUXs
 signal mux_address_A : std_logic_vector(19 downto 0);
 signal mux_request_WR : std_logic;


signal vga_req_1, vga_req_2 : std_logic;
signal vga_D_int : std_logic_vector(15 downto 0);

signal X0, Y0, X1, Y1, Color : std_logic_vector(15 downto 0);
signal active_sig : std_logic;
signal ready : std_logic;
signal hi_bit, low_bit : std_logic;

type opb_state is (IDLE, COMMON, XFER);
signal cstate, nxstate : opb_state;

signal cs, wr, rd : std_logic;

 --Signals for PadIO

 begin

 vga1 : vga
    port map (
      clk => OPB_Clk,
      pix_clk => Pixel_Clock,
      rst => OPB_Rst,
      video_data => vga_D,
      video_addr => vga_A,
      video_req => vga_request,
      vidout_clk => VIDOUT_Clk,
      vidout_RCR => VIDOUT_RED,
      vidout_GY => VIDOUT_GREEN,
      vidout_BCB => VIDOUT_BLUE,
      vidout_BLANK_N => VIDOUT_BLANK_N,
      vidout_HSYNC_N => VIDOUT_HSYNC_N,
      vidout_VSYNC_N => VIDOUT_VSYNC_N
    );
```

```
padio1 : 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 => mux_address_A,
    pb_ub => hi_bit,
    pb_lb => low_bit,
    pb_wr => mux_request_WR,
    pb_rd => vga_request,
    ram_ce => '1',
    pb_dread => vga_D_int,
    pb_dwrite => fsm_D
  );

fsm: fsm_bresenhams
  port map(
    X0 => X0,
    X1 => X1,
    Y0 => Y0,
    Y1 => Y1,
    Color => Color,
    start => active_sig,
    ready => ready,
    OPB_Clk => OPB_Clk,
    OPB_Rst => OPB_Rst,
    stall => vga_request,
    hi_bit => hi_bit,
    low_bit => low_bit,
    wr => fsm_WR,
    A => fsm_A,
    D => fsm_D
  );

mux_request : mux21
  port map (
    sel => vga_request,
    input1 => '0',
    input0 => fsm_WR,
    output => mux_request_WR
  );

mux_address : mux21_20
  port map (
    sel => vga_request,
    input1 => vga_A,
    input0 => fsm_A,
    output => mux_address_A
  );
```

```
-- Chip select for this entity - '1' when address is in the range
0xFEFF1000 - 0xFEFF1FFF
-- Chip select for this entity - '1' when address is in the range
0x00800000 - 0x00800FFF
cs <= OPB_select when OPB_ABus(31 downto 12) = "00000000100000000000"
else '0';

process(OPB_Clk)
begin
  if OPB_Clk'event and OPB_Clk='1' then
    vga_req_1 <= vga_request;
    vga_req_2 <= vga_req_1;
    if vga_req_2 = '1' then
      vga_D <= vga_D_int;
    end if;
  end if;
end process;

  -- Unused OPB control signals
  VGA_errAck <= '0';
  VGA_retry <= '0';
  VGA_toutSup <= '0';

  -- Latch the relevant OPB signals from the OPB, since they arrive
late
  LatchOPB: process (OPB_Clk, OPB_Rst)
  begin
    if OPB_Rst = '1' then
      opb_A <= ( others => '0' );
      opb_D <= ( others => '0' );
      opb_RW <= '1';
      cstate <= IDLE;
    elsif OPB_Clk'event and OPB_Clk = '1' then
      opb_A <= OPB_ABus;
      opb_D <= OPB_DBus;
      opb_RW <= OPB_RNW;
      cstate <= nxstate;
    end if;
  end process LatchOPB;

process (cstate, cs)
  begin
    nxstate <= cstate;
    wr <= '0';
    rd <= '0';
    VGA_xferAck  <= '0';

    case cstate is
      when IDLE => if cs = '1' then
        nxstate <= COMMON;
      end if;
      when COMMON =>
        if opb_RW = '0' then           -- write
          wr <= '1';
        else                            -- read
```

```
          rd <= '1';
        end if;
        nxstate <= XFER;
      when XFER =>
        VGA_xferAck  <= '1';
        nxstate <= IDLE;
    end case;

  end process;

 process (OPB_Clk, OPB_Rst)
  begin
    if OPB_Rst = '1' then
      VGA_Dbus <= ( others => '0');
      X0 <= (others => '0');
      Y0 <= (others => '0');
      X1 <= (others => '0');
      Y1 <= (others => '0');
      active_sig <= '0';

    elsif OPB_Clk'event and OPB_Clk = '1' then
      if wr = '1' then
      -- write
        case opb_A(5 downto 2) is
          when "0000" => X0 <= opb_d(15 downto 0);    --0xFEFF1X ..00
00..
          when "0001" => X1 <= opb_d(15 downto 0);    --0xFEFF1X ..00
01..
          when "0010" => Y0 <= opb_d(15 downto 0);    --0xFEFF1X ..00
10..
          when "0011" => Y1 <= opb_d(15 downto 0);    --0xFEFF1X ..00
11..
          when "0100" => Color <= opb_d(15 downto 0);active_sig  <=  '1';
            --0xFEFF1X ...1 00..
          when others => null;
        end case;
      end if;
      if rd = '1' then
        -- read
        case opb_A(5 downto 2) is
          when "0100" =>  VGA_Dbus <= (active_sig, others => '0');
        when "1000" =>  VGA_Dbus <= X1 & X0;
        --when "101" =>  VGA_Dbus <= X1;
        when "1010" =>  VGA_Dbus <= Y1 & Y0;
        --when "111" =>  VGA_Dbus <= Y1;
          when others => null;
        end case;
      else
         VGA_Dbus <= ( others => '0');
      end if;

      if ready = '1' then
        active_sig <= '0';
      end if;

    end if;
  end process;
```

```
end Behavioral;
```

pcores/opb_xsb300e_vga_v1_00_a/hdl/vhdl/pad_io.vhd

```
----------------------------------------------------------------------
--
-- I/O Pads and associated circuity for the XSB-300E board
--
-- Cristian Soviani, Dennis Lim, and Stephen A. Edwards
--
-- Pad drivers, a little glue logic, and flip-flops for most bus
signals
--
-- All signals, both control and the data and address busses, are
registered.
-- FDC and FDP flip-flops are forced into the pads to do this.
--
-- Only the data bus is mildly complex: it latches data in both
directions
-- as well as delaying the tristate control signals a cycle.
--
----------------------------------------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity pad_io is
  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 pad_io;

architecture Behavioral of pad_io is

  -- Flip-flop with asynchronous clear

  component FDC
    port (
      C : in std_logic;
```

```
      CLR : in std_logic;
      D : in std_logic;
      Q : out std_logic);
  end component;

  -- Flip-flop with asynchronous preset

  component FDP
    port (
      C : in std_logic;
      PRE : in std_logic;
      D : in std_logic;
      Q : out std_logic);
  end component;

  -- Setting the iob attribute to "true" ensures that instances of
these
  -- components are placed inside the I/O pads and are therefore very
fast

  attribute iob : string;
  attribute iob of FDC : component is "true";
  attribute iob of FDP : component is "true";

  -- Fast off-chip output buffer, low-voltage TTL levels, 24 mA drive
  -- I is the on-chip signal, O is the pad

  component OBUF_F_24
    port (
      O : out STD_ULOGIC;
      I : in STD_ULOGIC);
  end component;

  -- Fast off-chip input/output buffer, low-voltage TTL levels, 24 mA
drive
  -- T is the tristate control input, IO is the pad,

  component IOBUF_F_24
    port (
      O : out STD_ULOGIC;
      IO : inout STD_ULOGIC;
      I : in STD_ULOGIC;
      T : in STD_ULOGIC);
  end component;

  signal pb_addr_1: std_logic_vector(19 downto 0);
  signal pb_dwrite_1: std_logic_vector(15 downto 0);
  signal pb_tristate: std_logic_vector(15 downto 0);
  signal pb_dread_a: std_logic_vector(15 downto 0);
  signal we_n, pb_we_n1: std_logic;
  signal oe_n, pb_oe_n1: std_logic;
  signal lb_n, pb_lb_n1: std_logic;
  signal ub_n, pb_ub_n1: std_logic;
  signal ramce_n, ram_ce_n1: std_logic;
  signal dataz : std_logic;

begin
```

```
-- Write enable

we_n <= not pb_wr;

we_ff : FDP port map (
  C => clk, PRE => rst,
  D => we_n,
  Q => pb_we_n1);

we_pad : OBUF_F_24 port map (
  O => PB_WE_N,
  I => pb_we_n1);

-- Output Enable

oe_n <= not pb_rd;

oe_ff : FDP port map (
  C => clk,
  PRE => rst,
  D => oe_n,
  Q => pb_oe_n1);

oe_pad : OBUF_F_24 port map (
  O => PB_OE_N,
  I => pb_oe_n1);

-- RAM Chip Enable

ramce_n <= not ram_ce;

ramce_ff : FDP port map (
  C => clk, PRE => rst,
  D => ramce_n,
  Q => ram_ce_n1);

ramce_pad : OBUF_F_24 port map (
  O => RAM_CE_N,
  I => ram_ce_n1);

-- Upper byte enable

ub_n <= not pb_ub;

ub_ff : FDP port map (
  C => clk, PRE => rst,
  D => ub_n,
  Q => pb_ub_n1);

ub_pad : OBUF_F_24 port map (
  O => PB_UB_N,
  I => pb_ub_n1);

-- Lower byte enable

lb_n <= not pb_lb;
```

```
  lb_ff : FDP port map (
    C => clk,
    PRE => rst,
    D => lb_n,
    Q => pb_lb_n1);

  lb_pad : OBUF_F_24 port map (
    O => PB_LB_N,
    I => pb_lb_n1);

  -- 20-bit address bus

  addressbus : for i in 0 to 19 generate
    address_ff : FDC port map (
      C => clk, CLR => rst,
      D => pb_addr(i),
      Q => pb_addr_1(i));

    address_pad : OBUF_F_24 port map (
      O => PB_A(i),
      I => pb_addr_1(i));
  end generate;

  -- 16-bit data bus

  dataz <= (not pb_wr) or pb_rd;

  databus : for i in 0 to 15 generate
    dtff : FDP port map (               -- Trisate enable
      C => clk,
      PRE => rst,
      D => dataz,
      Q => pb_tristate(i));

    drff : FDP port map (
      C => clk,
      PRE => rst,
      D => pb_dread_a(i),
      Q => pb_dread(i));

    dwff : FDP port map (
      C => clk,
      PRE => rst,
      D => pb_dwrite(i),
      Q => pb_dwrite_1(i));

    data_pad : IOBUF_F_24 port map (
      O => pb_dread_a(i),
      IO => PB_D(i),
      I => pb_dwrite_1(i),
      T => pb_tristate(i));
  end generate;

end Behavioral;
```

pcores/opb_xsb300e_vga_v1_00_a/hdl/vhdl/vga.vhd

```
--------------------------------------------------------------------------
--
-- VGA video generator
--
-- Uses the vga_timing module to generate hsync etc.
-- Massages the RAM address and requests cycles from the memory
controller
-- to generate video using one byte per pixel
--
-- Cristian Soviani, Dennis Lim, and Stephen A. Edwards
--
--------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity vga is
  port (
    clk            : in std_logic;
    pix_clk        : in std_logic;
    rst            : in std_logic;
    video_data     : in std_logic_vector(15 downto 0);
    video_addr     : out std_logic_vector(19 downto 0);
    video_req      : out std_logic;
    VIDOUT_CLK     : out std_logic;
    VIDOUT_RCR     : out std_logic_vector(9 downto 0);
    VIDOUT_GY      : out std_logic_vector(9 downto 0);
    VIDOUT_BCB     : 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);
end vga;

architecture Behavioral of vga is

  -- 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";
```

```
  component vga_timing
    port (
      h_sync_delay         : out std_logic;
      v_sync_delay         : out std_logic;
      blank                : out std_logic;
      vga_ram_read_address : out std_logic_vector (19 downto 0);
      pixel_clock          : in std_logic;
      reset                : in std_logic);
  end component;

  signal r                      : std_logic_vector (9 downto 0);
  signal g                      : std_logic_vector (9 downto 0);
  signal b                      : std_logic_vector (9 downto 0);
  signal blank,blankn                 : std_logic;
  signal hsync,hsyncn                 : std_logic;
  signal vsync,vsyncn                 : std_logic;
  signal vga_ram_read_address   : std_logic_vector(19 downto 0);
  signal vreq                   : std_logic;
  signal vreq_1                 : std_logic;
  signal load_video_word        : std_logic;
  signal vga_shreg              : std_logic_vector(15 downto 0);

begin

  st : vga_timing port map (
    pixel_clock => pix_clk,
    reset => rst,
    h_sync_delay => hsync,
    v_sync_delay => vsync,
    blank => blank,
    vga_ram_read_address => vga_ram_read_address);

  -- Video request is true when the RAM address is even

  -- FIXME: This should be disabled during blanking to reduce memory
traffic

  vreq <= not vga_ram_read_address(0);

  -- Generate load_video_word by delaying vreq two cycles

  process (pix_clk)
  begin
    if pix_clk'event and pix_clk='1' then
      vreq_1 <= vreq;
      load_video_word <= vreq_1;
    end if;
  end process;

  -- Generate video_req (to the RAM controller) by delaying vreq by
  -- a cycle synchronized with the pixel clock

  process (clk)
  begin
    if clk'event and clk='1' then
      video_req <= pix_clk and vreq;
```

```
    end if;
  end process;

  -- The video address is the upper 19 bits from the VGA timing
generator
  -- because we are using two pixels per word and the RAM address
counts words

  video_addr <= '0' & vga_ram_read_address(19 downto 1);

  -- The video shift register: either load it from RAM or shift it up a
byte

  process (pix_clk)
  begin
    if pix_clk'event and pix_clk='1' then
      if load_video_word = '1' then
        vga_shreg <= video_data;
      else
        -- Shift the low byte of read video data into the high byte
        vga_shreg <= vga_shreg(7 downto 0) & "00000000";
      end if;
    end if;
  end process;

  -- Copy the upper byte of the video word to the color signals
  -- Note that we use three bits for red and green and two for blue.

  r(9 downto 7) <= vga_shreg (15 downto 13);
  r(6 downto 0) <= "0000000";
  g(9 downto 7) <= vga_shreg (12 downto 10);
  g(6 downto 0) <= "0000000";
  b(9 downto 8) <= vga_shreg (9 downto 8);
  b(7 downto 0) <= "00000000";

  -- Video clock I/O pad to the DAC

  vidclk : OBUF_F_12 port map (
    O => VIDOUT_clk,
    I => pix_clk);

  -- Control signals: hsync, vsync, and blank

  hsyncn <= not hsync;
  hsync_ff : FD port map (
    C => pix_clk,
    D => hsyncn,
    Q => VIDOUT_HSYNC_N );

  vsyncn <= not vsync;
  vsync_ff : FD port map (
    C => pix_clk,
    D => vsyncn,
    Q => VIDOUT_VSYNC_N );

  blankn <= not blank;
  blank_ff :  FD port map (
```

```
    C => pix_clk,
    D => blankn,
    Q => VIDOUT_BLANK_N );

  -- Three digital color signals

  rgb_ff : for i in 0 to 9 generate

    r_ff : FD port map (
      C => pix_clk,
      D => r(i),
      Q => VIDOUT_RCR(i) );

    g_ff : FD port map (
      C => pix_clk,
      D => g(i),
      Q => VIDOUT_GY(i) );

    b_ff : FD port map (
      C => pix_clk,
      D => b(i),
      Q => VIDOUT_BCB(i) );

  end generate;

end Behavioral;
```

pcores/opb_xsb300e_vga_v1_00_a/hdl/vhdl/vga_timing.vhd

```
----------------------------------------------------------------------
--
-- VGA timing and address generator
--
-- Fixed-resolution address generator.  Generates h-sync, v-sync, and
blanking
-- signals along with a 20-bit RAM address.  H-sync and v-sync signals
are
-- delayed two cycles to compensate for the DAC pipeline.
--
-- Cristian Soviani, Dennis Lim, and Stephen A. Edwards
--
----------------------------------------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity vga_timing is
  port (
    pixel_clock  : in std_logic;
    reset        : in std_logic;
    h_sync_delay : out std_logic;
    v_sync_delay : out std_logic;
    blank        : out std_logic;
    vga_ram_read_address : out std_logic_vector(19 downto 0));
end vga_timing;
```

```
architecture Behavioral of vga_timing is

  constant SRAM_DELAY : integer := 3;

-- 640 X 480 @ 60Hz with a 25.175 MHz pixel clock
  constant H_ACTIVE      : integer := 640;
  constant H_FRONT_PORCH : integer := 16;
  constant H_BACK_PORCH  : integer := 48;
  constant H_TOTAL       : integer := 800;

  constant V_ACTIVE      : integer := 480;
  constant V_FRONT_PORCH : integer := 11;
  constant V_BACK_PORCH  : integer := 31;
  constant V_TOTAL       : integer := 524;

  signal line_count  : std_logic_vector (9 downto 0);   -- Y coordinate
  signal pixel_count : std_logic_vector (10 downto 0);  -- X coordinate

  signal h_sync : std_logic;  -- horizontal sync
  signal v_sync : std_logic;  -- vertical sync

  signal h_sync_delay0 : std_logic; -- h_sync delayed 1 clock
  signal v_sync_delay0 : std_logic; -- v_sync delayed 1 clock

  signal h_blank : std_logic;       -- horizontal blanking
  signal v_blank : std_logic;       -- vertical blanking

  -- flag to reset the ram address during vertical blanking
  signal reset_vga_ram_read_address : std_logic;

  -- flag to hold the address during horizontal blanking
  signal hold_vga_ram_read_address : std_logic;

  signal ram_address_counter : std_logic_vector (19 downto 0);

begin

  -- Pixel counter

  process ( pixel_clock, reset )
  begin
    if reset = '1' then
      pixel_count <= "00000000000";
    elsif pixel_clock'event and pixel_clock = '1' then
      if pixel_count = (H_TOTAL - 1) then
        pixel_count <= "00000000000";
      else
        pixel_count <= pixel_count + 1;
      end if;
    end if;
  end process;

  -- Horizontal sync

  process ( pixel_clock, reset )
  begin
```

```
    if reset = '1' then
      h_sync <= '0';
    elsif pixel_clock'event and pixel_clock = '1' then
      if pixel_count = (H_ACTIVE + H_FRONT_PORCH - 1) then
        h_sync <= '1';
      elsif pixel_count = (H_TOTAL - H_BACK_PORCH - 1) then
        h_sync <= '0';
      end if;
    end if;
  end process;

  -- Line counter

  process ( pixel_clock, reset )
  begin
    if reset = '1' then
      line_count <= "0000000000";
    elsif pixel_clock'event and pixel_clock = '1' then
      if ((line_count = V_TOTAL - 1) and (pixel_count = H_TOTAL - 1))
then
        line_count <= "0000000000";
      elsif pixel_count = (H_TOTAL - 1) then
        line_count <= line_count + 1;
      end if;
    end if;
  end process;

  -- Vertical sync

  process ( pixel_clock, reset )
  begin
    if reset = '1' then
      v_sync <= '0';
    elsif pixel_clock'event and pixel_clock = '1' then
      if line_count = (V_ACTIVE + V_FRONT_PORCH -1) and
         pixel_count = (H_TOTAL - 1) then
        v_sync <= '1';
      elsif line_count = (V_TOTAL - V_BACK_PORCH - 1) and
            pixel_count = (H_TOTAL - 1) then
        v_sync <= '0';
      end if;
    end if;
  end process;

  -- Add two-cycle delays to h/v_sync to compensate for the DAC
pipeline

  process ( pixel_clock, reset )
  begin
    if reset = '1' then
      h_sync_delay0 <= '0';
      v_sync_delay0 <= '0';
      h_sync_delay  <= '0';
      v_sync_delay  <= '0';
    elsif pixel_clock'event and pixel_clock = '1' then
      h_sync_delay0 <= h_sync;
      v_sync_delay0 <= v_sync;
```

```
      h_sync_delay  <= h_sync_delay0;
      v_sync_delay  <= v_sync_delay0;
    end if;
  end process;

  -- Horizontal blanking

  -- The constants are offset by two to compensate for the delay
  -- in the composite blanking signal

  process ( pixel_clock, reset )
  begin
    if reset = '1' then
      h_blank <= '0';
    elsif pixel_clock'event and pixel_clock = '1' then
      if pixel_count = (H_ACTIVE - 2) then
        h_blank <= '1';
      elsif pixel_count = (H_TOTAL - 2) then
        h_blank <= '0';
      end if;
    end if;
  end process;

  -- Vertical Blanking

  -- The constants are offset by two to compensate for the delay
  -- in the composite blanking signal

  process ( pixel_clock, reset )
  begin
    if reset = '1' then
      v_blank <= '0';
    elsif pixel_clock'event and pixel_clock = '1' then
      if line_count = (V_ACTIVE - 1) and pixel_count = (H_TOTAL - 2)
then
        v_blank <= '1';
      elsif line_count = (V_TOTAL - 1) and pixel_count = (H_TOTAL - 2)
then
        v_blank <= '0';
      end if;
    end if;
  end process;

  -- Composite blanking

  process ( pixel_clock, reset )
  begin
    if reset = '1' then
      blank <= '0';
    elsif pixel_clock'event and pixel_clock = '1' then
      if (h_blank or v_blank) = '1' then
        blank <= '1';
      else
        blank <= '0';
      end if;
    end if;
  end process;
```

```
  -- RAM address counter

  -- Two control signals:

  -- reset_ram_read_address is active from the end of each field until
the
  -- beginning of the next

  -- hold_vga_ram_read_address is active from the end of each line to
the
  -- start of the next

  process ( pixel_clock, reset )
  begin
    if reset = '1' then
      reset_vga_ram_read_address <= '0';
    elsif pixel_clock'event and pixel_clock = '1' then
      if line_count = V_ACTIVE - 1 and
         pixel_count = ( (H_TOTAL - 1 ) - SRAM_DELAY ) then
        -- reset the address counter at the end of active video
        reset_vga_ram_read_address <= '1';
      elsif line_count = V_TOTAL - 1 and
            pixel_count =((H_TOTAL -1) - SRAM_DELAY) then
        -- re-enable the address counter at the start of active video
        reset_vga_ram_read_address <= '0';
      end if;
    end if;
  end process;

  process ( pixel_clock, reset )
  begin
    if reset = '1' then
      hold_vga_ram_read_address <= '0';
    elsif pixel_clock'event and pixel_clock = '1' then
      if pixel_count = ((H_ACTIVE - 1) - SRAM_DELAY) then
        -- hold the address counter at the end of active video
        hold_vga_ram_read_address <= '1';
      elsif pixel_count = ((H_TOTAL - 1) - SRAM_DELAY) then
        -- re-enable the address counter at the start of active video
        hold_vga_ram_read_address <= '0';
      end if;
    end if;
  end process;

  process ( pixel_clock, reset )
  begin
    if reset = '1' then
      ram_address_counter <= "00000000000000000000";
    elsif pixel_clock'event and pixel_clock = '1' then
      if reset_vga_ram_read_address = '1' then
        ram_address_counter <= "00000000000000000000";
      elsif hold_vga_ram_read_address = '0' then
        ram_address_counter <= ram_address_counter + 1;
      end if;
    end if;
  end process;
```

```
  vga_ram_read_address <= ram_address_counter;

end Behavioral;
```

c_source_files/main.c

```
#include "xbasic_types.h"
#include "xio.h"
#include "xintc_l.h"
#include "xuartlite_l.h"
#include "xparameters.h"
#define REG1ADDR 0x00800000
#define REG2ADDR 0x00800004
#define REG3ADDR 0x00800008
#define REG4ADDR 0x0080000C
#define REG5ADDR 0x00800010
#define REG6ADDR 0x00800014
#define REG7ADDR 0x00800018
#define REG8ADDR 0x0080001C
#define W 640
#define H 480
#define l_maze 10
#define w_maze 10
#define XMIN 130
#define XMAX 510
#define YMIN 50
#define YMAX 430
#define buffer_boundary 5
//#define shift_x 256
//#define shift_y 64
#define shift_x 200
#define shift_y 0

int x0_pli, y0_pli, x1_pli, y1_pli;
int volatile uart_interrupt_count = 0;
char uart_character;
char uart_buffer[5];
int volatile cp_i;
int rp_i;
volatile mutex;
int BLOCK_SIZE;
short draw_read=0;

int half_len;
int square_half_len;

/*
 * Interrupt service routine for the UART
 */
void uart_handler(void *callback)
{
  Xuint32 IsrStatus;
  Xuint8 incoming_character;
  mutex=0;
```

```
  /* Check the ISR status register so we can identify the interrupt
source */
  IsrStatus = XIo_In32(XPAR_MYUART_BASEADDR + XUL_STATUS_REG_OFFSET);

  if ((IsrStatus & (XUL_SR_RX_FIFO_FULL |
XUL_SR_RX_FIFO_VALID_DATA)) != 0) {
    /* The input FIFO contains data: read it */
    incoming_character =
      (Xuint8) XIo_In32( XPAR_MYUART_BASEADDR + XUL_RX_FIFO_OFFSET );

    uart_character = incoming_character;
    ++uart_interrupt_count;

    uart_buffer[cp_i]=incoming_character;
    cp_i++;
    if(cp_i==buffer_boundary)
      cp_i=0;
  }
  mutex=1;
}

void draw_line_ver(int x0, int y0, int y1, int color) // |
{
      //x0_pli=(x0 >> 1) - (y0 >> 1) + shift_x;
      //y0_pli=(x0 >> 1) + (y0 >> 1) - shift_y;
      //x1_pli=(x0 >> 1) - (y1 >> 1) + shift_x;
      //y1_pli=(x0 >> 1) + (y1 >> 1) - shift_y;

      x0_pli=(((x0 >> 1) + (x0 >> 2) + (x0 >> 3) - (y0 >> 1)) >> 1) +
shift_x;
      y0_pli=(((x0 >> 1) + (y0 >> 1) + (y0 >> 2) + (y0 >> 3)) >> 1) -
shift_y;
      x1_pli=(((x0 >> 1) + (x0 >> 2) + (x0 >> 3) - (y1 >> 1)) >> 1) +
shift_x;
      y1_pli=(((x0 >> 1) + (y1 >> 1) + (y1 >> 2) + (y1 >> 3)) >> 1) -
shift_y;

      /*
      if(x0_pli < 0)
            x0_pli=0;
      if(x1_pli < 0)
            x1_pli=0;
      if(y0_pli < 0)
            y0_pli=0;
      if(y1_pli < 0)
            y1_pli=0;
            */

      XIo_Out16(REG1ADDR, x0_pli);
      XIo_Out16(REG2ADDR, x1_pli);
      XIo_Out16(REG3ADDR, y0_pli);
      XIo_Out16(REG4ADDR, y1_pli);
      XIo_Out16(REG5ADDR, color);
      draw_read=XIo_In16(REG5ADDR);
      while(draw_read!=0)
            draw_read=XIo_In16(REG5ADDR);
}
```

```
//void draw_line_hon(int x0, int y0, int x1, int color) // --
void draw_line_hon(int x0, int y0, int x1, int color) // --
{
	//x0_pli=(x0 >> 1) - (y0 >> 1) + shift_x;
	//y0_pli=(x0 >> 1) + (y0 >> 1) - shift_y;
	//x1_pli=(x1 >> 1) - (y0 >> 1) + shift_x;
	//y1_pli=(x1 >> 1) + (y0 >> 1) - shift_y;
	x0_pli=(((x0 >> 1) + (x0 >> 2) + (x0 >> 3) - (y0 >> 1)) >> 1) +
shift_x;
	y0_pli=(((x0 >> 1) + (y0 >> 1) + (y0 >> 2) + (y0 >> 3)) >> 1) -
shift_y;
	x1_pli=(((x1 >> 1) + (x1 >> 2) + (x1 >> 3) - (y0 >> 1)) >> 1) +
shift_x;
	y1_pli=(((x1 >> 1) + (y0 >> 1) + (y0 >> 2) + (y0 >> 3)) >> 1) -
shift_y;

	/*
	if(x0_pli < 0)
		x0_pli=0;
	if(x1_pli < 0)
		x1_pli=0;
	if(y0_pli < 0)
		y0_pli=0;
	if(y1_pli < 0)
		y1_pli=0;
		*/

	XIo_Out16(REG1ADDR, x0_pli);
	XIo_Out16(REG2ADDR, x1_pli);
	XIo_Out16(REG3ADDR, y0_pli);
	XIo_Out16(REG4ADDR, y1_pli);
	XIo_Out16(REG5ADDR, color);
	draw_read=XIo_In16(REG5ADDR);
	while(draw_read!=0)
		draw_read=XIo_In16(REG5ADDR);
}

void draw_line_hon_no_rotate(int x0, int y0, int x1, int color) // --
{
	XIo_Out16(REG1ADDR, x0);
	XIo_Out16(REG2ADDR, x1);
	XIo_Out16(REG3ADDR, y0);
	XIo_Out16(REG4ADDR, y0);
	XIo_Out16(REG5ADDR, color);
	draw_read=XIo_In16(REG5ADDR);
	while(draw_read!=0)
		draw_read=XIo_In16(REG5ADDR);
}

void draw_circle(int center_x, int center_y, int color)
{
	int i;
	int x, y;

	x0_pli=(((center_x >> 1) + (center_x >> 2) + (center_x >> 3) -
(center_y >> 1)) >> 1) + shift_x; //for 30 degree version
```

```
      y0_pli=(((center_x >> 1) + (center_y >> 1) + (center_y >> 2) +
(center_y >> 3)) >> 1) - shift_y; //for 30 degree version

      for(y=-half_len;y<half_len;y++)
      {
        for(x=0;x*x+y*y<square_half_len;x++);
        {
            draw_line_hon_no_rotate(x0_pli-x, y0_pli+y, x0_pli+x,
color); //for 30 degree version
          //draw_line_hon(center_x-x, center_y+y, center_x+x, color);
//for 45 degree version
        }
      }
}

struct myUnit
{
   short setnum;
   char bottom;
   char right;
};
struct myUnit myMaze[l_maze][w_maze];

//checks if all units are in the same set
int SameSet(void){
  int i, j;
  for(i=0;i<l_maze;i++){
    for(j=0;j<w_maze;j++){
      if(i==l_maze-1 && j==w_maze-1)
      return 1;
      else if(j!=w_maze-1){
      if((myMaze[i][j].setnum) != (myMaze[i][j+1].setnum))
        return 0;
      }
      else{
      if((myMaze[i][j].setnum) != (myMaze[i+1][0].setnum))
        return 0;
      }
    }
  }
  return 1;
}

void draw_hor(int i, int j)
{
      draw_line_hon(j*BLOCK_SIZE+XMIN, YMIN+(i+1)*BLOCK_SIZE,
(j+1)*BLOCK_SIZE+XMIN+2, 0x1111);
}

void draw_ver(int i, int j)
{
      draw_line_ver((j+1)*BLOCK_SIZE+XMIN, YMIN+i*BLOCK_SIZE,
YMIN+(i+1)*BLOCK_SIZE+2, 0x1111);
}

void printMaze(void){
  int i,j,r,b;
```

```
  for(i=0;i<l_maze;i++){
    for(j=0;j<w_maze;j++){

      r=myMaze[i][j].right;
      b=myMaze[i][j].bottom;

      if(b && !(i==l_maze-1 && j==w_maze-1))
            draw_hor(i,j);
      if(r)
            draw_ver(i,j);
     }
  }
  draw_line_ver(XMIN, YMIN, YMAX, 0x1111);
  draw_line_hon(XMIN+BLOCK_SIZE, YMIN, XMAX, 0x1111);
}

void create_maze()
{
  int i, j,randUnit2,r, c, k, temp;
  k=1;

  //maze initialization, sets every wall to 1
  for(i=0;i<l_maze;i++)
    for(j=0;j<w_maze;j++){
      myMaze[i][j].setnum=k;
      myMaze[i][j].right=1;
      myMaze[i][j].bottom=1;
      k++;
    }

  while(SameSet()!=1){
    r=rand() % l_maze;
    c=rand() % w_maze;
    randUnit2 = rand() % 2;

    if(randUnit2==0 && myMaze[r][c].right==1 && c!=w_maze-1 &&
myMaze[r][c].setnum != myMaze[r][c+1].setnum){
      temp=myMaze[r][c+1].setnum;
      myMaze[r][c].right=0;

      for(i=0;i<l_maze;i++)
      for(j=0;j<w_maze;j++){
        if(myMaze[i][j].setnum == temp)
          myMaze[i][j].setnum = myMaze[r][c].setnum;
      }
    }
    else if(randUnit2==1 && myMaze[r][c].bottom==1 && r!=l_maze-1 &&
myMaze[r][c].setnum != myMaze[r+1][c].setnum){
      temp=myMaze[r+1][c].setnum;
      myMaze[r][c].bottom=0;

      for(i=0;i<l_maze;i++)
      for(j=0;j<w_maze;j++){
        if(myMaze[i][j].setnum == temp)
          myMaze[i][j].setnum = myMaze[r][c].setnum;
      }
```

```
    }
  }
  printMaze();
}


int main()
{
  char buf[2];
  int i, center_x, center_y, tempx, tempy;
  cp_i=0;
  rp_i=0;

  XIntc_RegisterHandler( XPAR_INTC_BASEADDR, XPAR_MYUART_DEVICE_ID,
(XInterruptHandler)uart_handler, (void *)0);
  XIntc_mEnableIntr( XPAR_INTC_BASEADDR, XPAR_MYUART_INTERRUPT_MASK);
  XIntc_mMasterEnable( XPAR_INTC_BASEADDR );
  XIntc_Out32(XPAR_INTC_BASEADDR + XIN_MER_OFFSET,
XIN_INT_MASTER_ENABLE_MASK);
  microblaze_enable_interrupts();
  XUartLite_mEnableIntr(XPAR_MYUART_BASEADDR);

  //clean the screen
  for(i=0;i<480;i++)
  {
      XIo_Out16(REG1ADDR, 0);
      XIo_Out16(REG2ADDR, 639);
      XIo_Out16(REG3ADDR, i);
      XIo_Out16(REG4ADDR, i);
      XIo_Out16(REG5ADDR, 0x0000);
      draw_read=XIo_In16(REG5ADDR);
      while(draw_read!=0)
            draw_read=XIo_In16(REG5ADDR);
  }

  BLOCK_SIZE= (XMAX - XMIN)/w_maze;
  center_x=XMIN+BLOCK_SIZE/2;
  center_y=YMIN+BLOCK_SIZE/2;

  //half_len= (BLOCK_SIZE>>1) - 7; //for 45 degree version
  half_len= (BLOCK_SIZE>>2) - 3; //for 30 degree version
  square_half_len = half_len*half_len;

  create_maze();

  draw_circle(center_x, center_y, 0xaaaa);
  for (;;) {
    buf[1] = '\0';
    print(""); //strange, we need this action to get UART input
    while (rp_i != cp_i) //rp_i, cp_i are the two indexes for competing
method in circular buffer implementation
    {
      while(mutex==0) ;
      buf[0] = uart_buffer[rp_i];
      draw_circle(center_x, center_y, 0);

      tempx=center_x;
```

```
      tempy=center_y;

      if (buf[0] == 'B') // 2, down
            center_y = center_y + BLOCK_SIZE;
      else if (buf[0] == 'D') // 4, left
            center_x = center_x - BLOCK_SIZE;
      else if (buf[0] == 'C') // 6, right
            center_x = center_x + BLOCK_SIZE;
      else if (buf[0] == 'A') // 8, up
            center_y = center_y - BLOCK_SIZE;

            // boundary conditions
        if (center_y > YMAX - BLOCK_SIZE / 2)
            center_y = YMAX - BLOCK_SIZE / 2;
        else if (center_y < YMIN + BLOCK_SIZE / 2)
              center_y = YMIN + BLOCK_SIZE / 2;

        if (center_x > XMAX - BLOCK_SIZE / 2)
              center_x = XMAX - BLOCK_SIZE / 2;
        else if (center_x < XMIN + BLOCK_SIZE / 2)
              center_x = XMIN + BLOCK_SIZE / 2;

      if(myMaze[(tempy-YMIN)/BLOCK_SIZE][(tempx-
XMIN)/BLOCK_SIZE].right==1 && buf[0]=='C')
            center_x=tempx;

      if(myMaze[(tempy-YMIN)/BLOCK_SIZE][(tempx-XMIN)/BLOCK_SIZE-
1].right==1 && buf[0]=='D')
            center_x=tempx;

      if(myMaze[(tempy-YMIN)/BLOCK_SIZE][(tempx-
XMIN)/BLOCK_SIZE].bottom==1 && buf[0] == 'B')
            center_y=tempy;

      if(myMaze[(tempy-YMIN)/BLOCK_SIZE-1][(tempx-
XMIN)/BLOCK_SIZE].bottom==1 && buf[0] == 'A')
            center_y=tempy;

        // draw the most recent circle
      if((center_x-XMIN)/BLOCK_SIZE==w_maze-1 && (center_y-
YMIN)/BLOCK_SIZE==l_maze-1)
            draw_circle(center_x, center_y, 0x6666);
      else
            draw_circle(center_x, center_y, 0xaaaa);

      //move the "reading pointer index" rp_i one more char right
      rp_i++;
     if(rp_i==buffer_boundary) //when it encoutner the buffer boundary,
return to the header, this is why so called circular buffer
            rp_i=0;
    }
  }

  return 0;
}
```

7. REFERENCES

[1] Lab 5 in Embedded System Design Spring 2004.

http://www1.cs.columbia.edu/~sedwards/classes/2004/4840/index.html

[2] Bresenham's line algorithm - Wikipedia, the free encyclopedia, http://en.wikipedia.org/wiki/Bresenham's_line_algorithm

[3] Foley, Van Dam, Feiner, Hughes, Phillips. Transformation Matrix section in

"Introduction to Computer Graphics".