ECE 385

Spring 2017

Final Project

Paint Application on FPGA

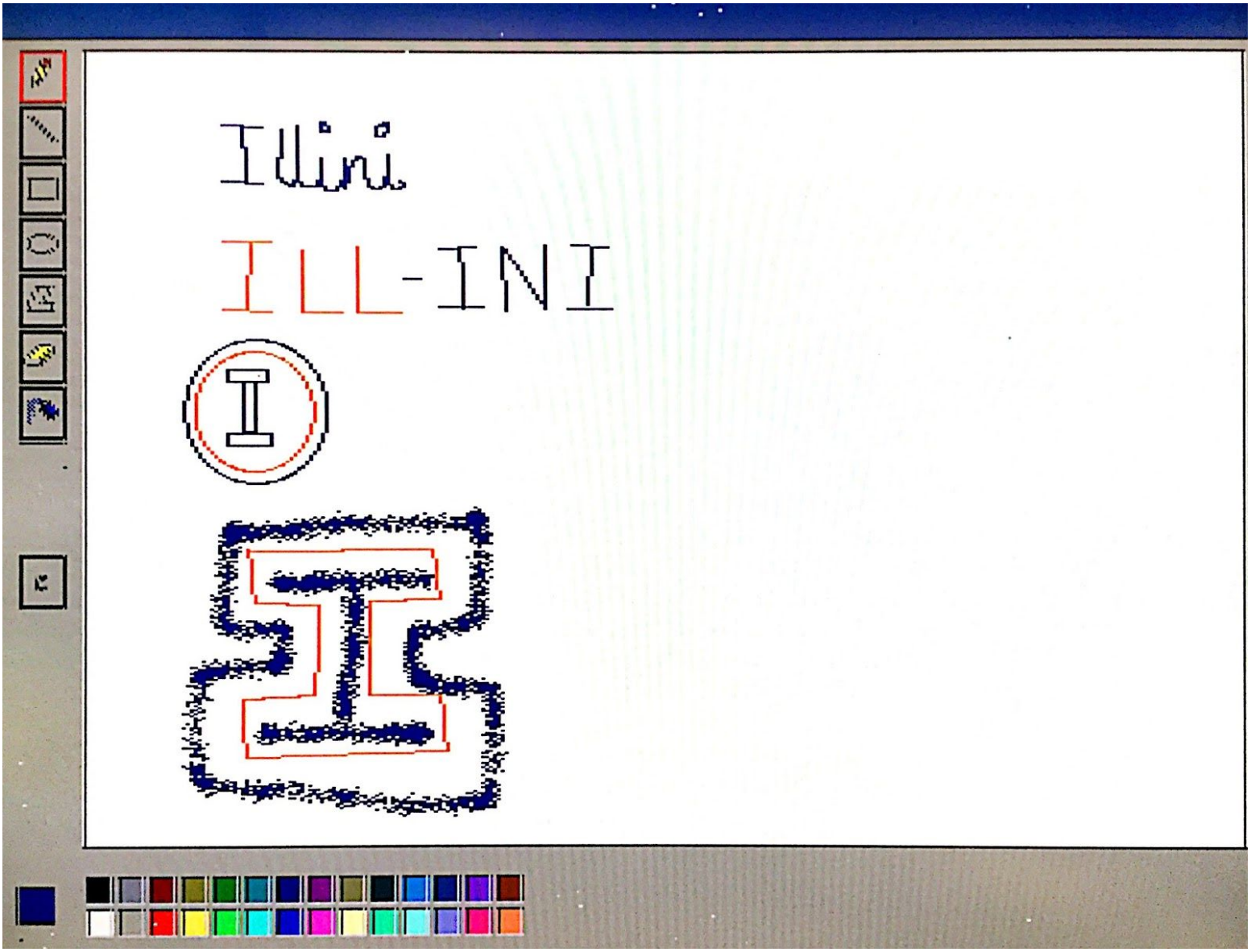
Agnivah Poddar, Arnav Agarwal

ABO, Thursday 3pm

TA: Zhenhong Liu

INTRODUCTION

We implemented a paint application, similar to MS Paint on the FPGA. The application used a NIOS-II processor and a PS/2 Mouse to interface with the IP core, which uses a VGA controller and the SRAM as a frame buffer. The application features several selections to draw on the screen and a color palette containing 28 colors. The mouse is used to select and draw the options on the screen in the selected color, also chosen by clicking the left mouse button.



WRITTEN DESCRIPTION

1) OVERVIEW OF THE APPLICATION

The application is very similar to use as MS Paint. The user uses the mouse to interact with the application. The application allows you to draw the following things on the screen of the monitor -

1. Free Draw - Use the mouse as a pencil and draw wherever the user wants to on the canvas by drawing at the point where the mouse button is clicked.
2. Line - Draws a line between two points which are selected by the user clicking on two points on the screen.
3. Rectangle - Similar to line, it draws a square between two points selected by the user, which act as the ends of the diagonal of the rectangle.
4. Circle - Draws a circle between two points selected by the user, using the points as the ends of its diameter.
5. Polygon - Allows the user to continue drawing lines until it forms a closed figure. Therefore, the user can draw continuous lines one after the other till it forms a closed figure.
6. Spray - Sprays color over the canvas on the screen under the cursor as the user moves the mouse around while holding the left button.

It also allows the user to -

1. Eraser - Erases the pixel on the screen under the cursor if the user clicks/holds the left button on the canvas.
2. Clear Canvas - Clears the canvas, that is, makes it blank to make a new drawing.

2) SYSTEM ON CHIP

The NIOS II acts as the system controller to handle tasks that do not need to be high performance (for example user interface, data input and output) while the accelerator peripheral in the FPGA logic handles the high performance operations. In our project, we used the Qsys wizard again to instantiate PIO blocks and a NIOS II processor to perform the task in hand. The modules used are described in detail below:

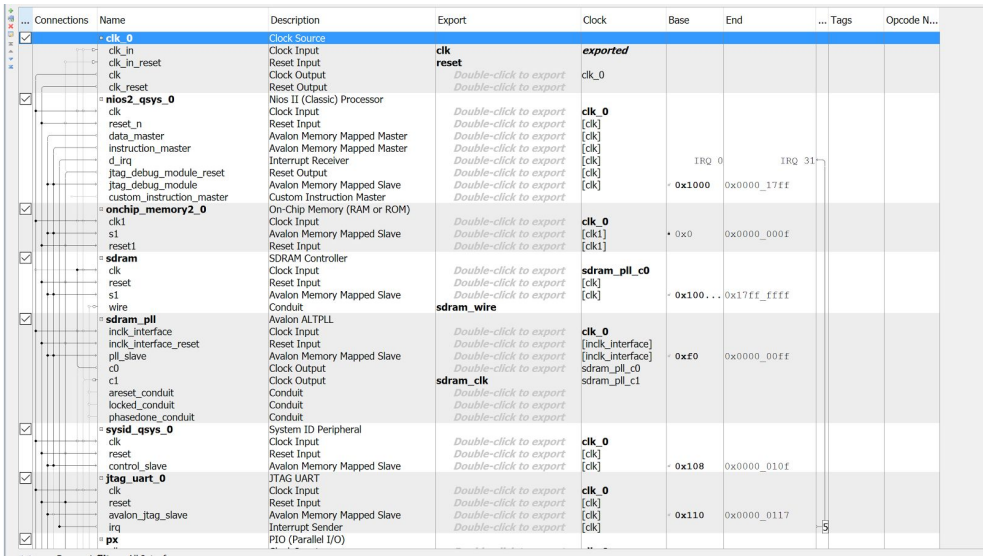
- **Clk_0** - This is the 50MHz clock that is used to control the circuit
- **Nios2_qsys_0** - This is the NIOS processor block that controls and handles other hardware. It provides the data and instruction buses that is accessed by other blocks.
- **Onchip_memory2_0 -** This is a storage unit that is built in on the processor chip. This storage unit is limited in size and is only 16 bytes for our usage but is used for immediate storage by the processor. It provides high throughput and low latency.
- **Sdram -** This is the larger off-chip Synchronous Dynamic RAM that stores the program. It is 1Gbit in size and its instantiation parameters are described later below.

SDRAM Parameter	Short Name	Parameter Value
Data Width	[width]	32 bits
# of Rows	[nrows]	13
# of Columns	[ncols]	10
# of Chip Selects	[ncs]	1
# of Banks	[nbanks]	4

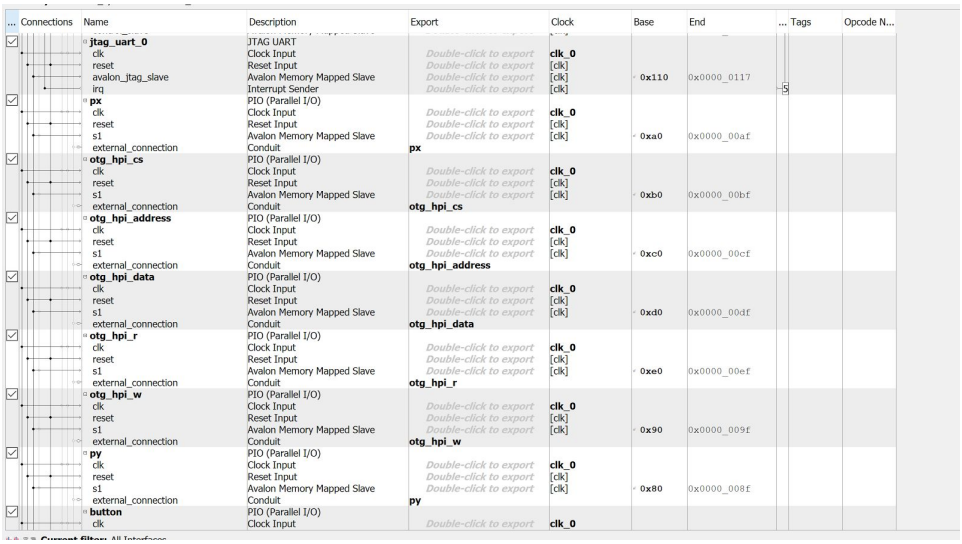
- **Sdram_pll** - This is a controller for the SDRAM and uses a second clock that is 3ns faster to make sure the SDRAM is refreshed for data retention prior to any reading or writing instructions to it.
- **Sysid_qsys_0 -** This is used to check the compatibility of the hardware and software and is used for debugging.
- **Jtag_uart_0 -** This is a serial communication block which allows us to print statements using the printf command from the NIOS II system to the stdout via the USB programming cable. The process of sending and receiving strings to console can be a slow process and so this is connected on interrupt receiver IRQ 5 so that processor is not blocked when a print interrupt request is being served. The processor unit doesn't need to be blocked while waiting for the transmission of the string data.

- **PIO Blocks -**

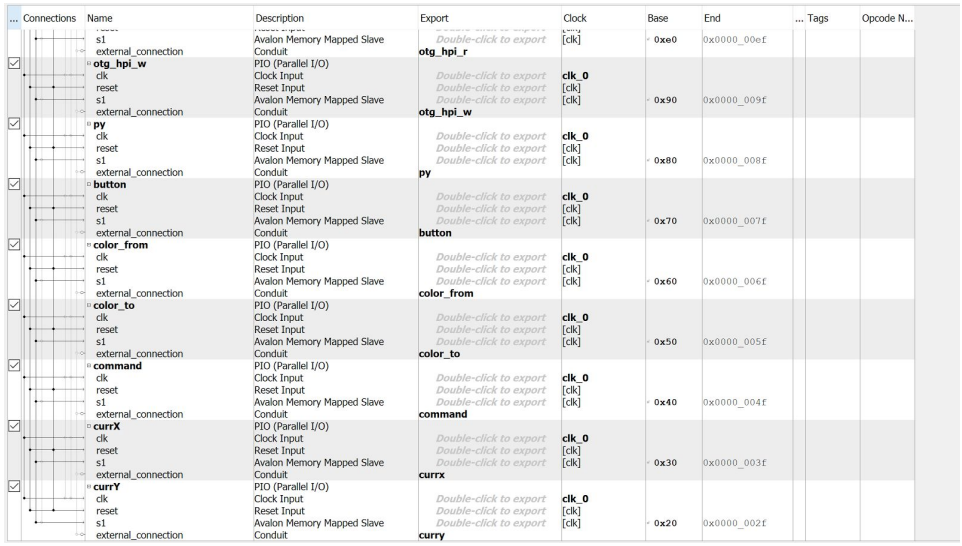
Name	Direction	Width	Purpose
px	In	10	Gets the x-coordinate of cursor from the PS/2 Mouse.
py	In	10	Gets the y-coordinate of cursor from the PS/2 Mouse.
Otg_hpi_cs	Out	1	(USED FOR USB MOUSE) Chip select signal. Specifies if the RAM on the HPI should be active. Active low.
Otg_hpi_address	Out	2	(USED FOR USB MOUSE) Address signal. Specifies the port register and the access.
Otg_hpi_r	Out	1	(USED FOR USB MOUSE) Enable Read signal. Specifies if the value can be read from the HPI RAM. Active low.
Otg_hpi_w	Out	1	(USED FOR USB MOUSE) Enable Write signal. Specifies if the value can be written to the HPI RAM. Active low.
Otg_hpi_data	Inout	16	(USED FOR USB MOUSE) The parallel data-bus of the Host Port Interface.
Color_from	Out	16	Color that needs to be written to the SRAM.
Color_to	In	16	Color that is being read from the SRAM.
Currx	Out	10	The x coordinate at which the color needs to be written.
Curry	Out	10	The y coordinate at which the color needs to be written.
Command	Out	1	The select signal to enable writing to the SRAM



Connections	Name	Description	Export	Clock	Base	End	Tags	Opcode N...
	clk_0	Clock Source						
	clk_in	Clock Input	**clk**	***exported***				
	clk_in_reset	Reset Input	**reset**					
	clk	Clock Output	*Double-click to export*	clk_0				
	clk_reset	Reset Output	*Double-click to export*					
☑	**nios2_qsys_0**	Nios II (Classic) Processor						
	clk	Clock Input	*Double-click to export*	**clk_0**				
	reset_n	Reset Input	*Double-click to export*	[clk]				
	data_master	Avalon Memory Mapped Master	*Double-click to export*	[clk]				
	instruction_master	Avalon Memory Mapped Master	*Double-click to export*	[clk]				
	d_irq	Interrupt Receiver	*Double-click to export*	[clk]	IRQ 0	IRQ 31		
	jtag_debug_module_reset	Reset Output	*Double-click to export*	[clk]				
	jtag_debug_module	Avalon Memory Mapped Slave	*Double-click to export*	[clk]	0x1000	0x0000_17ff		
	custom_instruction_master	Custom Instruction Master	*Double-click to export*					
☑	**onchip_memory2_0**	On-Chip Memory (RAM or ROM)						
	clk1	Clock Input	*Double-click to export*	**clk_0**				
	s1	Avalon Memory Mapped Slave	*Double-click to export*	[clk1]	0x0	0x0000_000f		
	reset1	Reset Input	*Double-click to export*	[clk1]				
☑	**sdram**	SDRAM Controller						
	clk	Clock Input	*Double-click to export*	**sdram_pll_c0**				
	reset	Reset Input	*Double-click to export*	[clk]				
	s1	Avalon Memory Mapped Slave	*Double-click to export*	[clk]	0x100...	0x17ff_ffff		
	wire	Conduit	**sdram_wire**					
☑	**sdram_pll**	Avalon ALTPLL						
	inclk_interface	Clock Input	*Double-click to export*	**clk_0**				
	inclk_interface_reset	Reset Input	*Double-click to export*	[inclk_interface]				
	pll_slave	Avalon Memory Mapped Slave	*Double-click to export*	[inclk_interface]	0xf0	0x0000_00ff		
	c0	Clock Output	*Double-click to export*	sdram_pll_c0				
	c1	Clock Output	**sdram_clk**	sdram_pll_c1				
	areset_conduit	Conduit	*Double-click to export*					
	locked_conduit	Conduit	*Double-click to export*					
	phasedone_conduit	Conduit	*Double-click to export*					
☑	**sysid_qsys_0**	System ID Peripheral						
	clk	Clock Input	*Double-click to export*	**clk_0**				
	reset	Reset Input	*Double-click to export*	[clk]				
	control_slave	Avalon Memory Mapped Slave	*Double-click to export*	[clk]	0x108	0x0000_010f		
☑	**jtag_uart_0**	JTAG UART						
	clk	Clock Input	*Double-click to export*	**clk_0**				
	reset	Reset Input	*Double-click to export*	[clk]				
	avalon_jtag_slave	Avalon Memory Mapped Slave	*Double-click to export*	[clk]	0x110	0x0000_0117		
	irq	Interrupt Sender	*Double-click to export*	[clk]			0	
☑	**px**	PIO (Parallel I/O)						



Connections	Name	Description	Export	Clock	Base	End	Tags	Opcode N...
☑	**jtag_uart_0**	JTAG UART						
	clk	Clock Input	*Double-click to export*	**clk_0**				
	reset	Reset Input	*Double-click to export*	[clk]				
	avalon_jtag_slave	Avalon Memory Mapped Slave	*Double-click to export*	[clk]	0x110	0x0000_0117		
	irq	Interrupt Sender	*Double-click to export*	[clk]			0	
☑	**px**	PIO (Parallel I/O)						
	clk	Clock Input	*Double-click to export*	**clk_0**				
	reset	Reset Input	*Double-click to export*	[clk]				
	s1	Avalon Memory Mapped Slave	*Double-click to export*	[clk]	0xa0	0x0000_00af		
	external_connection	Conduit	**px**					
☑	**otg_hpi_cs**	PIO (Parallel I/O)						
	clk	Clock Input	*Double-click to export*	**clk_0**				
	reset	Reset Input	*Double-click to export*	[clk]				
	s1	Avalon Memory Mapped Slave	*Double-click to export*	[clk]	0xb0	0x0000_00bf		
	external_connection	Conduit	**otg_hpi_cs**					
☑	**otg_hpi_address**	PIO (Parallel I/O)						
	clk	Clock Input	*Double-click to export*	**clk_0**				
	reset	Reset Input	*Double-click to export*	[clk]				
	s1	Avalon Memory Mapped Slave	*Double-click to export*	[clk]	0xc0	0x0000_00cf		
	external_connection	Conduit	**otg_hpi_address**					
☑	**otg_hpi_data**	PIO (Parallel I/O)						
	clk	Clock Input	*Double-click to export*	**clk_0**				
	reset	Reset Input	*Double-click to export*	[clk]				
	s1	Avalon Memory Mapped Slave	*Double-click to export*	[clk]	0xd0	0x0000_00df		
	external_connection	Conduit	**otg_hpi_data**					
☑	**otg_hpi_r**	PIO (Parallel I/O)						
	clk	Clock Input	*Double-click to export*	**clk_0**				
	reset	Reset Input	*Double-click to export*	[clk]				
	s1	Avalon Memory Mapped Slave	*Double-click to export*	[clk]	0xe0	0x0000_00ef		
	external_connection	Conduit	**otg_hpi_r**					
☑	**otg_hpi_w**	PIO (Parallel I/O)						
	clk	Clock Input	*Double-click to export*	**clk_0**				
	reset	Reset Input	*Double-click to export*	[clk]				
	s1	Avalon Memory Mapped Slave	*Double-click to export*	[clk]	0x90	0x0000_009f		
	external_connection	Conduit	**otg_hpi_w**					
☑	**py**	PIO (Parallel I/O)						
	clk	Clock Input	*Double-click to export*	**clk_0**				
	reset	Reset Input	*Double-click to export*	[clk]				
	s1	Avalon Memory Mapped Slave	*Double-click to export*	[clk]	0x80	0x0000_008f		
	external_connection	Conduit	**py**					
☑	**button**	PIO (Parallel I/O)						
	clk	Clock Input	*Double-click to export*	**clk_0**				



Connections	Name	Description	Export	Clock	Base	End	Tags	Opcode N...
	s1	Avalon Memory Mapped Slave	*Double-click to export*	[clk]	0xe0	0x0000_00ef		
	external_connection	Conduit	**otg_hpi_r**					
☑	**otg_hpi_w**	PIO (Parallel I/O)						
	clk	Clock Input	*Double-click to export*	**clk_0**				
	reset	Reset Input	*Double-click to export*	[clk]				
	s1	Avalon Memory Mapped Slave	*Double-click to export*	[clk]	0x90	0x0000_009f		
	external_connection	Conduit	**otg_hpi_w**					
☑	**py**	PIO (Parallel I/O)						
	clk	Clock Input	*Double-click to export*	**clk_0**				
	reset	Reset Input	*Double-click to export*	[clk]				
	s1	Avalon Memory Mapped Slave	*Double-click to export*	[clk]	0x80	0x0000_008f		
	external_connection	Conduit	**py**					
☑	**button**	PIO (Parallel I/O)						
	clk	Clock Input	*Double-click to export*	**clk_0**				
	reset	Reset Input	*Double-click to export*	[clk]				
	s1	Avalon Memory Mapped Slave	*Double-click to export*	[clk]	0x70	0x0000_007f		
	external_connection	Conduit	**button**					
☑	**color_from**	PIO (Parallel I/O)						
	clk	Clock Input	*Double-click to export*	**clk_0**				
	reset	Reset Input	*Double-click to export*	[clk]				
	s1	Avalon Memory Mapped Slave	*Double-click to export*	[clk]	0x60	0x0000_006f		
	external_connection	Conduit	**color_from**					
☑	**color_to**	PIO (Parallel I/O)						
	clk	Clock Input	*Double-click to export*	**clk_0**				
	reset	Reset Input	*Double-click to export*	[clk]				
	s1	Avalon Memory Mapped Slave	*Double-click to export*	[clk]	0x50	0x0000_005f		
	external_connection	Conduit	**color_to**					
☑	**command**	PIO (Parallel I/O)						
	clk	Clock Input	*Double-click to export*	**clk_0**				
	reset	Reset Input	*Double-click to export*	[clk]				
	s1	Avalon Memory Mapped Slave	*Double-click to export*	[clk]	0x40	0x0000_004f		
	external_connection	Conduit	**command**					
☑	**currX**	PIO (Parallel I/O)						
	clk	Clock Input	*Double-click to export*	**clk_0**				
	reset	Reset Input	*Double-click to export*	[clk]				
	s1	Avalon Memory Mapped Slave	*Double-click to export*	[clk]	0x30	0x0000_003f		
	external_connection	Conduit	**currx**					
☑	**currY**	PIO (Parallel I/O)						
	clk	Clock Input	*Double-click to export*	**clk_0**				
	reset	Reset Input	*Double-click to export*	[clk]				
	s1	Avalon Memory Mapped Slave	*Double-click to export*	[clk]	0x20	0x0000_002f		
	external_connection	Conduit	**curry**					

3) SOFTWARE

STEP 0 - SETTING UP THE BACKGROUND

The software in our code acts as the main control panel of our application. It starts of by drawing the initial background of the application, that is, the interface. The interface was designed by making an image. The image was then converted into pixel values using a MATLAB code. Then, using another C program, the pixel was made into a double dimensional array, containing the color value and the continuous frequency of the value. Then using these values, the initial setup of the application is complete and the background image is drawn on the screen.

STEP 1 - TOOL and COLOR SELECTION

Depending on where the cursor of the mouse is, the tools are selected. This works by reading the position of the cursor of the PS/2 mouse which is interfaced in hardware. Using IORD, we read the x-position and y-position of the cursor. Therefore, by looking at the location of the cursor on the screen, the program selects the tool at that location and assigns the value to the cursor_type. Cursor_type can take the following values -

- PENCIL
- LINE
- RECT
- CIRCLE
- POLYGON
- ERASER
- SPRAY
- FILL
- PICK
- CLEAR

Similarly, depending on the location of the cursor, the color is also selected. We have a palette of 28 colors to choose from.

STEP 2 - PERFORMING THE FUNCTION

Once the tool has been selected, the desired function must be performed. Therefore, depending on the tool selection, it does the following tasks for the selected tool -

- PENCIL - This tool allows the user to free draw. Therefore, by keeping the left button of the mouse pressed, the user can freely draw anywhere on the canvas. This works by writing the selected color's value at the location specified by the screen. Using IOWR, we write the x and y coordinates (currX and currY) to the hardware and that writes the color pixel (color_from) to the SRAM which is mapped to the VGA.

- LINE - This tool draws a line between the two points on which the user clicks. It stores the starting and ending coordinates and then uses the Bresenham algorithm to draw the line. The algorithm was implemented both in SystemVerilog and C.

  For the hardware, similar to the PENCIL, the values are written to hardware using write. This sends the values of the start and end coordinates (S_X, S_Y, E_X, E_Y) as well as the color which needs to be drawn (color_from). It also sends a start signal to start the state machine for the algorithm in hardware.

  For the software code, it performs the implementation in C and writes every pixel on the screen using IOWR as it is calculated. This works faster since there is less communication between hardware and software and is therefore used for the application.
- RECT - This tool draws a square between the two points, using the points as the ends of its diagonal. Similar to line, it uses Bresenham's algorithm to draw the four lines between the points (x1,y1)(x1,y2), (x1,y1)(x2,y1), (x1,y2)(x2,y2) and (x2,y1)(x2,y2)
- CIRCLE - This tool draws a circle using the inputted two points as the ends of its diameter. Then we use an algorithm implemented in software that sends each coordinate present on the circle to hardware using IOWR. It sends currX, currY and color_from.
- POLYGON - This tool allows the user to draw continuous lines to form a polygon. It sends the end points between each click to draw a lines continuously. Therefore, this too uses the same functions and signals like line.
- ERASER - This tool erases whatever is drawn at that point. Basically, it just writes the color white at the location pointed by the cursor. Therefore, it sends also sends currX, currY and color_from(white). Basically, this is a free draw using the color white.
- SPRAY - This tool sprays on the canvas in a free draw. It uses a random number generator to decide the radius as well as if it should draw a pixel at a location 4 by 4 pixels near the cursor or at the point of the cursor, thereby creating a spray look. It then sends the coordinates generated by it one by one to write to the SRAM similar to all the other functions.
- CLEAR - This tool clears the canvas by overwriting all the coordinates under the canvas by the color white. It loops over each pixel coordinate and sends it to the hardware along with the color white to clear the screen.

The software communicates with the PIOs in hardware using IORD and IOWR. Using IORD, we can read the values from hardware. Similarly, using IOWR, we can write to the hardware ports. Using the system.h file generated while generating the BSP, we get the base addresses of the hardware ports.

Basic functionality of software functions

- **delay() -** This function basically adds a delay by running an empty for loop for 1000 iterations. This is used to synchronize mouse interaction with the specific function it is supposed to do in order to avoid any unwanted actions.
- **initialPutPixel -** This function is used to write to the the SRAM. It takes in the current x and y position of the mouse and sends that data to the hardware. Along with that it also sends the current color value and write_enable signal for the SRAM, to the hardware.

```
void initialPutPixel(int x,int y, int color)
{
    int command = 1;
    IOWR(CURRX_BASE,0,x);
    IOWR(CURRY_BASE,0,y);
    IOWR(COLOR_FROM_BASE,0,color);
    IOWR(COMMAND_BASE,0,command);
    IOWR(COMMAND_BASE,0,0);
}
```

- **getPixel -** This function takes in the current x and current y position of the mouse cursor. It then reads the color of the pixel at that location by reading from the PIO used to communicate color data between software and hardware. It then returns that value.

```
int getPixel(int x, int y)
{
        int command = 0;
        IOWR(CURRX_BASE,0,x);
        IOWR(CURRY_BASE,0,y);
        IOWR(COMMAND_BASE,0,command);
        int val = IORD(COLOR_TO_BASE,0);
        usleep(10000);
        return val;
}
```

- **initialScreen -** This function is used to load the initial paint interface background image to the VGA. It is called once at the start of the program as the initial setup of the application. It reads data of the compressed background image and then calls initialPutPixel to print to the VGA screen.

```
void initialScreen()
{
    unsigned int x,y,i,b,c,j;
    i=0;
    j=0;
    for(y=0;y<480;y++)
    {
        for(x=0;x<640;x++)
        {
```

```
                    if(background_image[j][1] == 0)
                          j++;
                        b= background_image[j][0] << 8;
                        background_image[j][1] -=1;
                        if(background_image[j][1] == 0)
                                        j++;
                        b = b | background_image[j][0];
                     initialPutPixel(x,y,b);//0xe0e0 + (y<<8 | y));
                     i=i+1;
                  }
                  usleep(10);
               }
            }
```

- **Linehelper -** This is a helper function that is used to draw a line on the canvas when the Line tool is being used. According to our line drawing algorithm, it takes in the start_x and start_y coordinates for the line based on where the mouse was clicked first. Then it waits for the button to be pressed again to decide the end points of the line and then uses Bresenham's drawing algorithm to draw the line between those two points. Special care has been taken to make sure that the button click is registered only once using a flag check and a delay. This helps restrict to functionality to only when mouse button is released and not consider one click as multiple.

```
int linehelper(int x, int y, int color)
{
        int i =0;
        for(i=0; i<1000;i++);
        int but = 0;
        int flag = 0;
        int end_x,end_y;
        while(1)
        {
                but = IORD(BUTTON_BASE, 0);
                if(but == 0)
                        flag = 1;
                if(flag == 1 && but == 1)
                {
                        end_x = IORD(PX_BASE, 0);
                        end_y = IORD(PY_BASE, 0);
                        Bresenham(x,y,end_x,end_y,color);
                        while(IORD(BUTTON_BASE,0));
                        return 0;
                }
        }
}
```

- **Bresenham -** This is the line drawing algorithm described later. It draws the best fit line between two points.

```
void Bresenham(int x1, int y1, int x2, int y2,int color)
{
     int slope;
     int dx, dy, incE, incNE, d, x, y;
     int flag_dydx = 0, temp1,steep;
     int abs_dx , abs_dy;
     // Reverse lines where x1 > x2
     abs_dx = abs(x2 - x1);
     abs_dy = abs(y2 - y1);
     if(abs_dy > abs_dx)
     {
         steep = 1;
     }
     else
     {
         steep = 0;
     }
     if(steep ==1)
     {
        //swap(x1,y1) and swap(x2,y1)
         temp1 = x1;
         x1 = y1;
         y1 = temp1;
         temp1 = x2;
         x2 = y2;
         y2 = temp1;
     }
     if (x1 > x2)
     {
         //Bresenham(x2, y2, x1, y1);
         temp1 = x1;
         x1 = x2;
         x2 = temp1;
         temp1 = y1;
         y1 = y2;
         y2 = temp1;
         //return;
     }
     dx = x2 - x1;
     dy = y2 - y1;
     // Adjust y-increment for negatively sloped lines
     if (dy < 0)
     {
         slope = -1;
         dy = -dy;
     }
```

```
else
{
    slope = 1;
}
// Bresenham constants
incE = 2 * dy;
incNE = 2 * dy - 2 * dx;
d = 2 * dy - dx;
y = y1;
// Blit
for (x = x1; x <= x2; x++)
{
   if(steep == 1)
   {
     initialPutPixel(y,x,color);
   }
   else
   {
      initialPutPixel(x,y,color);
   }
    if (d <= 0)
    {
        d += incE;
    }
    else
    {
        d += incNE;
        y += slope;

    }
}
}
```

- **Circlehelper -** This is a helper function that is used to draw a circle on the canvas when the Circle tool is being used. According to our circle drawing algorithm, it takes in the start_x and start_y coordinates for one end of the diameter based on where the mouse was clicked first. Then it waits for the button to be pressed again to decide the end points of the diameter and then computes the center and radius. It then calls the DrawCircle function with the center and radius to draw the circle. Special care has been taken to make sure that the button click is registered only once using a flag check and a delay. This helps restrict to functionality to only when mouse button is released and not consider one click as multiple.

```
int circlehelper(int x,int y, int color)
{
       int i =0;
       for(i=0; i<1000;i++);
       int but = 0;
       int flag = 0;
       int end_x,end_y,center_x,center_y, rad;
```

```
        while(1)
        {
                but = IORD(BUTTON_BASE, 0);
                if(but == 0)
                        flag = 1;
                if(flag == 1 && but == 1)
                {
                     end_x = IORD(PX_BASE, 0);
                     end_y = IORD(PY_BASE, 0);
                     center_x = (x + end_x)/2;
                     center_y = (y + end_y)/2;
                     rad = sqrt(pow((x-end_x),2) +
                   pow((y-end_y),2))/2;
                     DrawCircle(center_x,center_y,rad,color);
                     while(IORD(BUTTON_BASE,0));
                     return 0;
                }
        }
}
```

- **DrawCircle -** This is the circle drawing algorithm described later. It draws the circle using center and radius as parameters.

```
void DrawCircle( int x0, int y0, int radius, int color)
{
   int x_plus, y_plus;
   int f = 1 - radius;
   int ddF_x = 0;
   int ddF_y = -2 * radius;
   int x = 0;
   int y = radius;
   y_plus = y0 +radius;
   initialPutPixel(x0, y_plus, color);
   delay();
   delay();
   y_plus = y0 - radius;
   initialPutPixel(x0, y_plus, color);
   delay();
   delay();
   x_plus = x0 + radius;
   initialPutPixel(x_plus, y0, color);
   delay();
   delay();
   x_plus = x0 - radius;
   initialPutPixel(x_plus, y0, color);
   delay();
   delay();
   while(x < y)
   {
     if(f >= 0)
     {
```

```
      y--;
      ddF_y += 2;
      f += ddF_y;
    }
    x++;
    ddF_x += 2;
    f += ddF_x + 1;
    x_plus = x0 + x;
    y_plus = y0 + y;
    initialPutPixel(x_plus, y_plus, color);
    delay();
    delay();
    x_plus = x0 - x;
    y_plus = y0 + y;
    initialPutPixel(x_plus, y_plus, color);
    delay();
    delay();
    x_plus = x0 + x;
    y_plus = y0 - y;
    initialPutPixel(x_plus, y_plus, color);
    delay();
    delay();
    x_plus = x0 - x;
    y_plus = y0 - y;
    initialPutPixel(x_plus, y_plus, color);
    delay();
    delay();
    x_plus = x0 + y;
    y_plus = y0 + x;
    initialPutPixel(x_plus, y_plus, color);
    delay();
    delay();
    x_plus = x0 - y;
    y_plus = y0 + x;
    initialPutPixel(x_plus, y_plus, color);
    delay();
    delay();
    x_plus = x0 + y;
    y_plus = y0 - x;
    initialPutPixel(x_plus, y_plus, color);
    delay();
    delay();
    x_plus = x0 - y;
    y_plus = y0 - x;
    initialPutPixel(x_plus, y_plus, color);
    delay();
    delay();
  }
}
```

- **Squarehelper -** This is a helper function that is used to draw a rectangle on the canvas when the Rect tool is being used. According to our rectangle drawing algorithm, it takes in the start_x and start_y coordinates for one end of the diagonal of the rectangle based on where the mouse was clicked first. Then it waits for the button to be pressed again to decide the end points of the diagonal and sends these two coordinates to the DrawSquare function. Special care has been taken to make sure that the button click is registered only once using a flag check and a delay. This helps restrict to functionality to only when mouse button is released and not consider one click as multiple.

```
int squarehelper ( int x1, int x2, int color)
{
	int i =0;
	for(i=0; i<1000;i++);
	int but = 0;
	int flag = 0;
	int end_x,end_y;
	while(1)
	{
		but = IORD(BUTTON_BASE, 0);
		if(but == 0)
			flag = 1;
		if(flag == 1 && but == 1)
		{
			end_x = IORD(PX_BASE, 0);
			end_y = IORD(PY_BASE, 0);
			DrawSquare(x1,x2,end_x,end_y,color);
			while(IORD(BUTTON_BASE,0));
			return 0;
		}
	}
}
```

- **DrawSquare** - This function calls Bresenham four times to draw the four sides of the rectangle.

```
void DrawSquare(int x1, int y1, int x2, int y2,int color)
{
     Bresenham(x1,y1,x1,y2,color);
     delay();
     Bresenham(x1,y2,x2,y2,color);
     delay();
     Bresenham(x2,y2,x2,y1,color);
     delay();
     Bresenham(x2,y1,x1,y1,color);
     delay();
}
```

- **Polygonhelper -** This function is used to draw a polygon of how many ever sides as the user wants it to be. The way the algorithm works is the the user first starts by clicking on the starting point and then every time another click is made it draws a line between the old points and these new points, to give a continuous closed figure. An excerpt of the code is as follows:

```
{
start_x = px;
start_y = py;
initialPutPixel(px,py,ink);
flag_line = 2;
              usleep(5000);
linehelper(start_x, start_y, ink);
       continue;
}
if(flag_line == 2)
{
       end_x = px;
       end_y = py;
       Bresenham(start_x, start_y, end_x, end_y, ink);
start_x = end_x;
start_y = end_y;
}
```

- **Main-** In this function, first the setup is done by calling initialScreen and then it handles the mouse button click. It selects the tool and then based on that it executes its functionality using helper functions. There is also code that handles the color palette. It is here that the position of the mouse is constantly sent to hardware using PIOs and also the click of the button to indicate writing to the SRAM memory as and when required. (Code for handling usb mouse is also written here but is commented out to switch between PS/2 mouse for better functionality)

- **Data.h -** This is the header file that stores data of the background Paint interface in a compressed manner. We wrote a C code to parse the raw image data file and compress it to be able to include it in Eclipse. The code that we wrote is provided below in the appendix.

4) HARDWARE

OVERVIEW

Our hardware is used to interface with the VGA, SRAM and the Mouse. It also performs the Bresenham algorithm to draw a line.

The monitor was connected to the board using the VGA port. The DE2-115 board includes a 15-pin D-SUB connector for VGA output. The VGA synchronization signals are provided directly from the Cyclone IV E FPGA. The resolution used for our project was 640×480.

The SRAM acts as a frame buffer for the VGA. The SRAM has 16-bit wide data and a 20-bit wide address, making it a good fit to work as the frame buffer for the VGA. To interface with the SRAM, we use Mem2IO, which handles all the input output between the SRAM and the NIOS II system. It assigns the data that needs to go to the NIOS II system as well as the SRAM. It also writes the data that needs to go to the SRAM. It also reads the data from both the CPU and the SRAM. Then using a tristate buffer, depending on the select signals, we read or write from the SRAM.

The VGA is interfaced with the SRAM simply by making its value equal to the value of the SRAM at the address specified by the current coordinates of the VGA, that is, DrawX and DrawY.

We used a PS/2 Mouse to get user input. The PS/2 controller interface consists of the PS/2 clock and the PS/2 data inputs, and two 8-bit data ports. One of the 8-bit data ports is used for sending commands to the PS/2 device (mouse or keyboard), the other is for receiving data from the PS/2 device. There are also control signals provided to indicate the arrival of a new command from the PS/2 device and the status of the command transmission to the PS/2 device. The timing control necessary for the PS/2 communication is handled by the controller. The mouse sends the data using 3 byte packets. Every packet consists of the relative movement since the last transmission and the button state.

VGA

We are displaying the output on the VGA and are using the SRAM on the Altera DE2-115 board as the frame buffer. We have implemented a VGA_controller in SystemVerilog to handle a resolution of 640x480 pixels with each pixel having a color depth of 8 bits (3R, 3G, 2B). The VGA_controller provides the coordinates Draw_x and Draw_y which is the point where the VGA is currently printing at.

FRAME BUFFER

The SRAM on the chip is being used as the frame buffer for the VGA and is addressed using 18 bits. The addressing scheme is the higher 9 bits are the higher 9 bits of the x-coordinate and the lower bits are the lower 9 bits of the y-coordinate.

```
SRAM_ADDR = { Coord_X[9:1] , Coord_Y[8:0]};
```

The data width is 16 bits long. Each address in the SRAM stores data of two adjacent pixels. The higher 8 bits hold the data of the odd pixel and the lower 8 bits hold color data of the even pixel.

INTERFACE BETWEEN VGA AND SRAM

We have used the approach that we learnt in lab6 to interface the VGA and CPU with the SRAM. We used a slightly modified Mem2IO and tristate buffer to handle data read and write, from and to memory. The mem2IO loads data from SRAM to the Data sent to CPU when the Output_enable is high.

The tri-state buffer is the interface between mem2IO and SRAM. It makes sure that only when the Write_enable signal is active, data is written into the SRAM. Otherwise a High Z is sent.

MODULE DESCRIPTION

Module: bresenham.sv
Inputs: clk, reset, start, [10:0] x0, [10:0] y0, [10:0]x1, [10:0]y1
Outputs: plot, [10:0] x, [10:0] y, done
Description: Uses the Bresenham's algorithm to output the coordinates between two points to draw a straight line. It features a state machine that keeps outputting x and y coordinates till the initial values don't get updated to the end values.
Purpose: Used to draw straight lines between two points in our paint application. (Inactive as performance of C code worked better)

Module: Color_Mapper.sv
Inputs: Clk, Reset, [9:0] BallX, [9:0] BallY, [9:0] BallS, [9:0] DrawX, [9:0] DrawY, [7:0] iR, [7:0] iG, [7:0] iB, VSync, HSync
Outputs: [7:0] VGA_R, [7:0] VGA_G, [7:0] VGA_B
Description: Sets the color of the pixels at the location of the mouse coordinates to display a cursor.
Purpose: Outputs the cursor of the mouse on the screen.

Module: HexDriver.sv
Inputs: [3:0]In0,
Outputs: [6:0]Out0,
Description: Maps the hex display segments with the hex character
Purpose: Used to correctly display the hex character in the hex displays.

Module: hpi_io_intf.sv
Inputs: Clk, Reset, [1:0] from_sw_address, [15:0] from_sw_data_out, from_sw_r, from_sw_w, from_sw_cs,
Outputs: [15:0] from_sw_data_in, [1:0] OTG_ADDR, OTG_RD_N, OTG_WR_N, OTG_CS_N, OTG_RST_N
Inout: [15:0] OTG_DATA
Description: This module acts as the tri-state buffer to load data into the OTG. Loads the values into the OTG signals according to the inputs provided.
Purpose: The input output interface for the Hardware Platform Interface. (Inactive when interfacing with a PS/2 mouse)

Module: lab8.sv
Inputs: CLOCK_50, [3:0] KEY, OTG_INT
Outputs: [6:0] HEX0, [6:0] HEX1, [7:0] VGA_R, [7:0] VGA_G, [7:0] VGA_B, VGA_HS, VGA_VS, VGA_CLK, VGA_BLANK_N, VGA_SYNC_N, [1:0] OTG_ADDR, OTG_CS_N, OTG_RD_N, OTG_WR_N, OTG_RST_N, [12:0] DRAM_ADDR, [1:0] DRAM_BA, DRAM_CAS_N, DRAM_CKE, DRAM_CS_N, [3:0] DRAM_DQM, DRAM_RAS_N, DRAM_WE_N, DRAM_CLK
Inout: [15:0] OTG_DATA, [31:0] DRAM_DQ, [15:0] SRAM_DQ, PS2_DAT, PS2_CLK
Description: It connects all the ports of the processor with the appropriate signals, i.e., the SDRAM connections, VGA connections, SRAM connections, PS/2 connections and OTG-USB connections. It also instantiates the other modules for proper functioning of our application.
Purpose: This top level module is used for making connections to the NIOS II Processor and send output signals to the VGA to ensure proper display

Module: Mem2IO.sv
Inputs: Clk, Reset, [19:0]ADDR, CE, UB, LB, OE, WE, VS, HS, [15:0]Data_from_CPU, [15:0]Data_from_SRAM
Outputs: [15:0]Data_to_CPU, [15:0]Data_to_SRAM
Description: If the output enable (OE) is active(low), it sets the value to be written to the processor to be the Data_from_SRAM. Also sets the Data_to_SRAM to be the Data_from_CPU.
Purpose: Acts as the buffer between the values that need to be read or written between the NIOS II processor and the SRAM.
Module: Mouse.sv
Inputs: CLOCK_50, RESET, KEY, PS2_CLK, PS2_DAT
Outputs: leftButton, middleButton, rightButton, [9:0] cursorX, [9:0] cursorY
Description: This module uses the Altera_UP_PS2 modules provided (reference link given) to get the data packets and determine the x and y coordinates along with the data if the button is pressed or not.
Purpose: Interfaces with the PS/2 mouse to give the locations of the cursor on the screen.

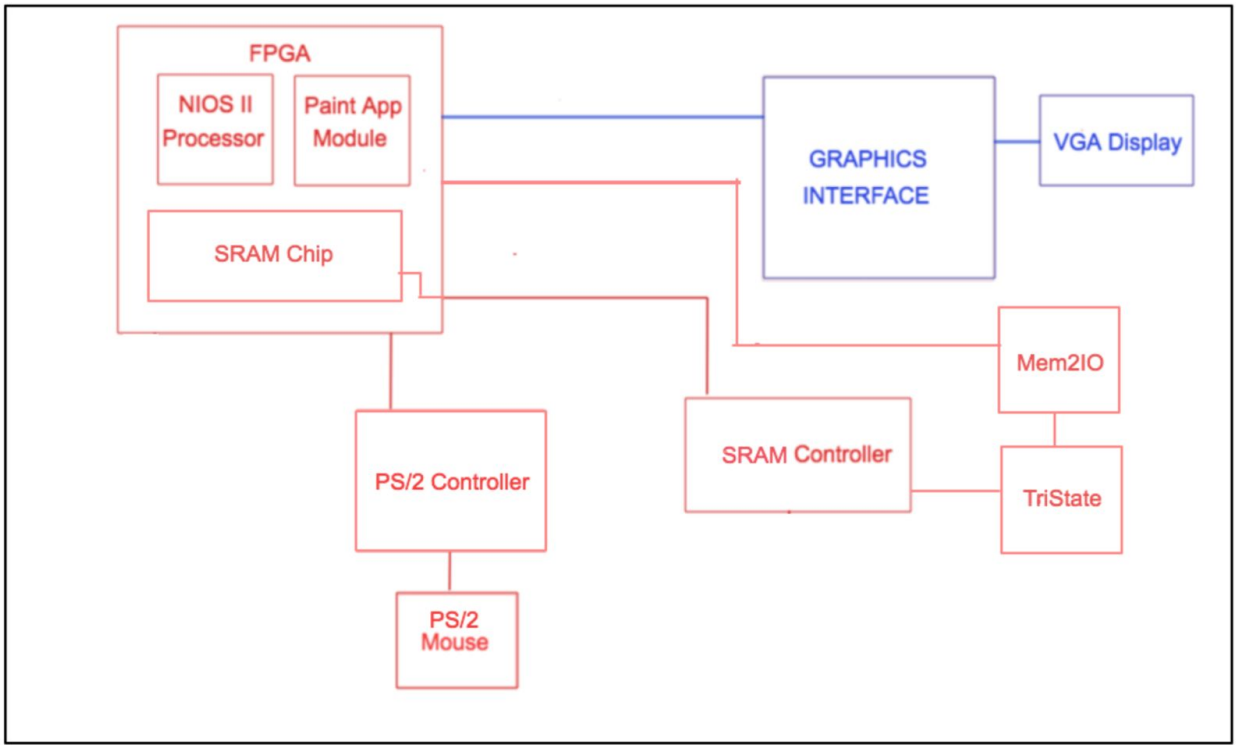
Module: VGA_controller.sv
Inputs: Clk, Reset
Outputs: VGA_HS, VGA_VS, VGA_CLK, VGA_BLANK_N, VGA_SYNC_N, [9:0] DrawX, [9:0] DrawY
Description: Sets the signals required for displaying the content on the VGA display as well as the coordinates for the ball.
Purpose: Acts as the main control unit for the VGA related tasks, that is, setting up the sync pulses and the frequency of the VGA clock.
Module: tristate.sv

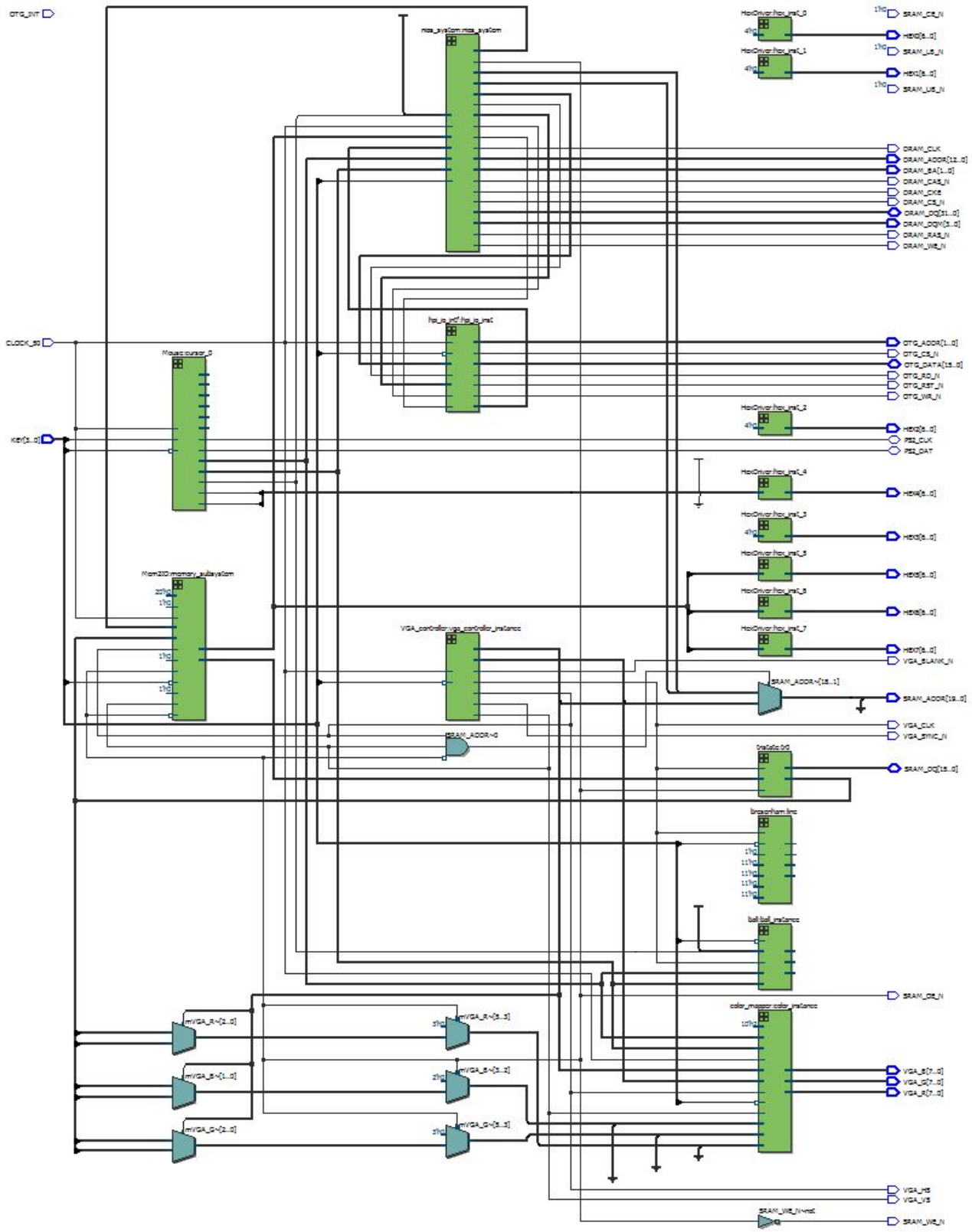
Inputs: Clk, tristate_output_enable, [N-1:0] Data_write
Outputs: [N-1:0]Data_read
Inout wire: [N-1:0]Data
Description: Provided file. Acts as the interface between the CPU and the memory.
Purpose: Acts as the bus from which we can read/write from/to memory.

Module: nios_system.v
Input Wires: clk_clk, reset_reset_n, [15:0] otg_hpi_data_in_port,
Output Wires: [15:0] keycode_export, [1:0] otg_hpi_address_export, otg_hpi_cs_export, [15:0] otg_hpi_data_out_port, otg_hpi_r_export, otg_hpi_w_export, sdram_clk_clk, [12:0] sdram_wire_addr, [1:0] sdram_wire_ba, sdram_wire_cas_n, sdram_wire_cke, sdram_wire_cs_n, [3:0] sdram_wire_dqm, sdram_wire_ras_n, sdram_wire_we_n
Inout Wire: [31:0] sdram_wire_dq
Description: A Qsys generated file, it contains all the ports required by the processor for proper functioning. Even though it wasn't part of the project, the names of the port had to be the same as the one in the top level module for proper synthesis.
Purpose: This module was used for ensuring all the ports required by the NIOS II processor were named and connected properly.

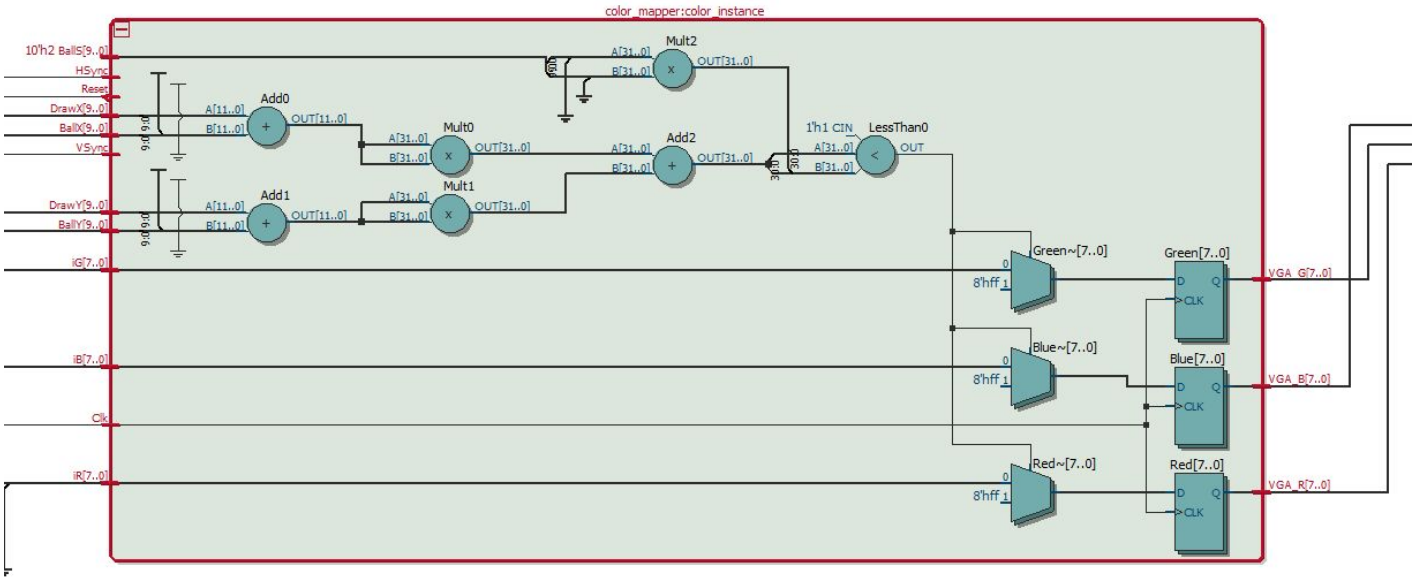
The qsys generates a .qip file, which is added to the project to give Quartus the required information about the system that was created. The qsys contains the NIOS processor, the on chip memory, the DRAM, the JTAG-UART and the PIOs for the proper functioning of the system(described above under the System on Chip section).

BLOCK DIAGRAMS

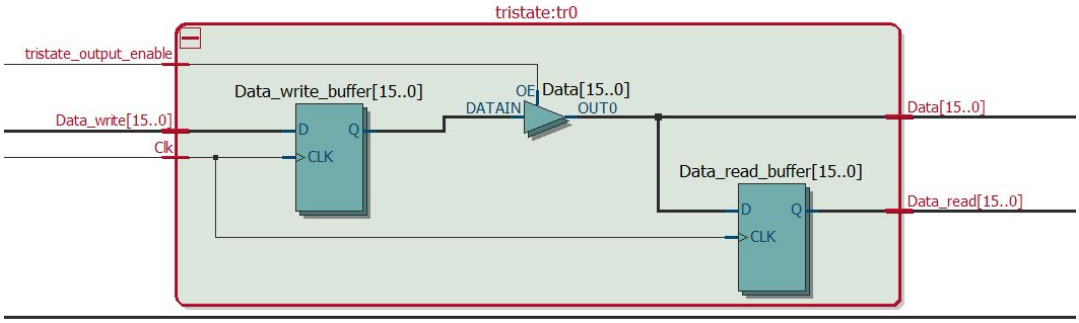




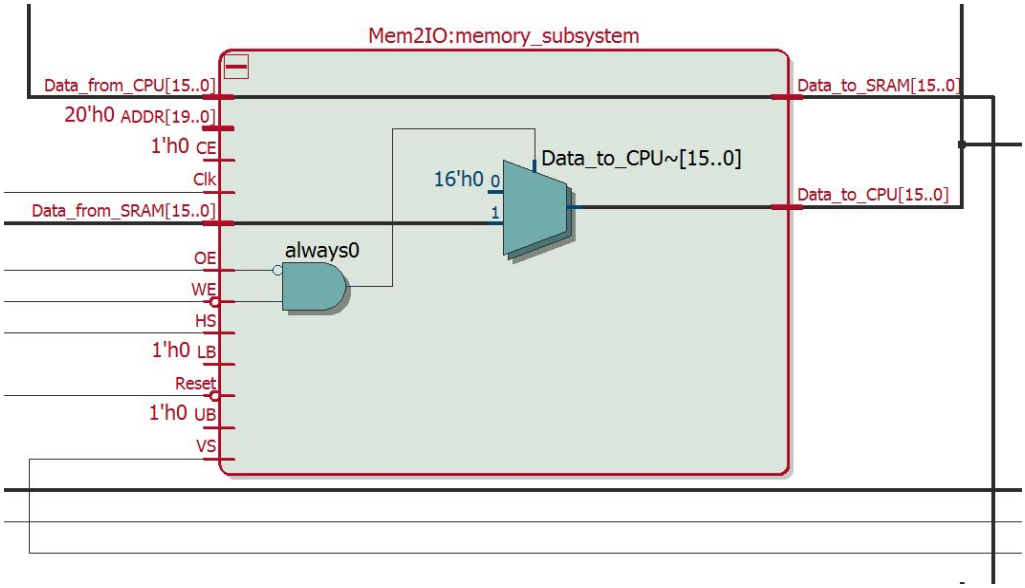
Top - level



color_mapper



Tristate buffer



Mem2IO

5) ALGORITHMS USED

BRESENHAM ALGORITHM

The Bresenham algorithm is an incremental scan conversion algorithm. The big advantage of this algorithm is that, it uses only integer calculations. Moving across the x axis in unit intervals and at each step choose between two different y coordinates.

Bresenham algorithm for slope m < 1

Step 1 – Input the two endpoints of line, storing the left end-point in (x_0, y_0).

Step 2 – Plot the point (x_0, y_0).

Step 3 – Calculate dx and dy, where dy is the change in y for every step dx.

Step 4 - Get the first value for the decision parameter as –

$$p_0 = 2dy - dx$$

Step 5 – At each X_k along the line, starting at k = 0, perform the following test –

If p_k < 0, the next point to plot is $(x_k + 1, y_k)$ and

$$p_{k+1} = p_k + 2dy$$

Otherwise, $(x_k, y_k + 1)$

$$p_{k+1} = p_k + 2dy - 2dx$$

Step 6 – Repeat step 5 for another $dx - 1$ rounds..

For m > 1, find out whether you need to increment x while incrementing y each time.

After solving, the equation for decision parameter p_k will be very similar, just the x and y in the equation gets interchanged.

Pseudocode:

```
function line(x0, y0, x1, y1)
      real deltax := x1 - x0
      real deltay := y1 - y0
      real deltaerr := abs(deltay / deltax)    // Assume deltax != 0
                  // note that this division needs to be done in a way
      that preserves the fractional part
      real error := deltaerr - 0.5
      int y := y0
      for x from x0 to x1
            plot(x,y)
                  error := error + deltaerr
                  if error ≥ 0.5 then
                        y := y + 1
                        error := error - 1.0
```

CIRCLE DRAWING ALGORITHM

Step 1 – Get the coordinates of the center of the circle and radius, and store them in x, y, and R respectively. Set P=0 and Q=R.
Step 2 – Set decision parameter D = 3 – 2R.
Step 3 – Repeat through step-8 while X < Y.
Step 4 – Call Draw Circle (X, Y, P, Q).
Step 5 – Increment the value of P.
Step 6 – If D < 0 then D = D + 4x + 6.
Step 7 – Else Set Y = Y + 1, D = D + 4(X-Y) + 10.
Step 8 – Call Draw Circle (X, Y, P, Q).

Design Resources and Statistic Table(Arnav)

LUT	3161
DSP	Simple Multipliers (18-bit) : 2 Embedded Multiplier Blocks : 2 Embedded Multiplier 9-bit elements : 4 Signed Embedded Multipliers : 2
Memory(BRAM)	55296 bits
Flip flop	2562
Frequency	115.32 MHz
Static Power	102.09 mW
Dynamic Power	0.91 mW
Total Power	196.21 mW

CHALLENGES and BUGS

Some of the issues that we faced are as follows:

- **Mouse movement** - With our USB mouse implementation, there was high latency in reading the position of the mouse and that resulted in a very rough mouse movement. This provided very poor results on the canvas. As a result we decided to also code in a PS/2 mouse which improved performance and smoothened mouse responses.
- **VGA artifacts** - There was a minor issue in the timing synchronization between printing to VGA and writing to SRAM. This gave rise to some flicker lines every time we tried writing to SRAM on the click of the button. This might have been also due to the low bandwidth of the memory.
- **Drawing shapes -** Getting the shape tools to work was relatively challenging. An issue we faced quite a lot was that one click of the mouse button was being registered as multiple clicks and so connected lines and shapes were getting drawn. This was fixed using a flag and lock mechanism that made sure the next click is registered only when the mouse button is released.
- **Loading our background image -** The raw image data that we had for our background was very large (307200 values , one for each pixel) and when included in eclipse it threw us an error saying that the offset went out of bounds. So we had to write the compression code (Appendix) to be able to compress the size of data and include it in Eclipse and read and write it to SRAM.

CONCLUSION

The final implementation of our project was quite successful. Our application performed all the tasks we expected it to perform quite smoothly and efficiently. We tried several methods to increase the performance of our project and at the end, we got fairly satisfactory results. Our application allowed free draw and spray and was successful in drawing basic shapes and using different colors. It also supported erasing the canvas and clear the canvas functionalities. We managed to complete our base task that we envisioned. We successfully implemented the different tools and their functionalities. The paint application was more challenging than we had imagined and so we couldn't implement the Jezz Ball game as well as the motion draw feature as we spent a lot of time to make the paint application perform well and up to standards. All in all, our final project was a success and we realized our initial goal.

REFERENCES

- Altera DE2-115 User Manual
  ftp://ftp.altera.com/up/pub/Altera_Material/Boards/DE2-115/DE2_115_User_Manual.pdf
- Paint FPGA
  https://people.ece.cornell.edu/land/courses/ece5760/FinalProjects/f2007/rc437_mj288/rc437_mj288/rc437_mj288/index.html
- PS/2 Mouse
  http://www.eecg.toronto.edu/~jayar/ece241_08F/AudioVideoCores/ps2/ps2.html

APPENDIX

- **Background Image compression C code**

```
#include <stdio.h>
#include <stdlib.h>
#include "ImgData.h" // holds the raw image data
#include "Data.h"
int main()
{
        // compressed 2D array
        int a[5132][2];
        long i;
        for(i = 0; i<307200; i++)
        {
            a[i][0] = -1;
            a[i][1] = -1;
        }
        int j;
        j=0;
        a[0][0] = initial_image[0];
        a[0][1] = 1;
        //compression
        for(i = 1; i<307200; i++)
        {
                if(initial_image[i] == a[j][0])
                        a[j][1]++;
                else
                {
                        j ++;
                        a[j][0] = initial_image[i];
                        a[j][1] = 1;
                }
        }
        for(i=0; i<=j; i++)
        {
                        printf("{%d ,%d} , ",a[i][0], a[i][1]); //outputted to a file
        }
        return 0;
}
```