INDIAN INSTITUTE OF INFORMATION TECHNOLOGY, ALLAHABAD

(Deemed University)

(A Centre of Excellence in Information Technology Established by Ministry of HRD, Govt. of India)

प्रज्ञानम् ब्रह्म

Project Report

IMPLEMENTATION OF 2-DIMENSIONAL GRAPHIC ENGINE ON FPGA

Project Supervisor:

Dr. Rajat Singh

Submitted by:

Dushyant Singh (IEC2011019)

Manvinder Singh (IEC2011028)

Ayush Dobhal (IEC2011081)

Students Declaration

We, hereby declare that the work presented in this project report entitled "**Implementation of 2-Dimensional graphic engine on FPGA**" submitted towards compilation of the 5th semester project of B.Tech (Electronics and Comm. Engg) at the Indian Institute of Information Technology Allahabad, is an authentic record of our original work carried out under the guidance of Dr. Rajat Singh. The project has been done in full compliance with the requirements and constraints of prescribed curriculum to the best of our knowledge.

Dushyant Singh (IEC2011019)
Manvindar Singh (IEC2011028)
Ayush Dobhal (IEC2011081)

Certificate

This is to certify that the project work entitled "**Implementation of 2-Dimensional graphic engine on FPGA**" has been carried out for the compilation of 5th semester project by B.Tech (Electronics and Comm. Engg.) students namely Dushyant Singh (IEC2011019), Manvinder Singh (IEC2011028) and Ayush Dobhal (IEC2011081) at the Indian Institute of Information Technology Allahabad. The above declaration made by the candidates is correct to the best of my knowledge and belief.

Dr.Rajat Singh
(Assistant Professor)

Acknowledgement

Our project titled as **“Implementation of 2-Dimensional Graphic Engine on FPGA“** could not have been completed if not for the help and encouragement that we got from our mentor **Dr. Rajat Singh**. We would like to sincerely thank you for the support and the proper guidelines that we got regarding this project. This genre was completely new for us but with his intense knowledge and constant pushing us to work hard, we could finally complete the project with contention. We thoroughly gained knowledge through the course of this semester. We are thereby submitting this report to you with full honesty and integrity.

Abstract

Two-Dimensional Graphics Card (GPU) on FPGA

This project realizes a simple graphics processing unit (GPU) on an FPGA (Field Programmable Gate Arrays) which can be used as a GPU to output graphics to a VGA (Video Graphics Array) display. A graphics system consists of several levels of hardware and software working together to make a computer capable of displaying visual information on a monitor. The primary engine behind the graphics system is the graphics controller, which consists of the custom hardware processor that performs all necessary mathematical operations required to generate pixel data (RGB colour), which is then stored in a region of memory. A software driver is also necessary to stimulate the hardware and provide an API for developers to create graphics applications. An Altium FPGA will be used to implement the GPU. Final output of the graphics will be displayed on a VGA compatible monitor. The result of the project is a GPU as well as VGA driver for the host device that allows a user to set up a graphical window to draw into and display on the VGA screen. The GPU will support multiple drawing functions such as text writing, line drawing, and rectangular fills.

Contents

Literature Survey

We were implementing 2-D graphic search engine on FPGA. A graphics system consists of several levels of hardware and software working together to make a computer capable of displaying visual information on a monitor.

1.Graphical displays are increasingly popular for many new applications and industries due to the dramatic increase in the use of graphical user interfaces (GUIs) in mass applications such as PCs, phones, PDAs, and other consumer devices. Many applications now support the addition of a high-resolution color display and GUI where five to ten years ago it would have just been too expensive to be accepted by the market. Thanks to the advent of low-cost FPGAs like the Altera® Cyclone® FPGA series, many developers have eliminated the problem of microcontroller-feature limitations by designing custom systems using a combination of soft microprocessor.[9](www.altera.com/literature/wp/wp-01100-graphic-lcd-display.pdf)

2.The most commonly requested upgrade features for embedded systems with graphical displays are color (or more colors), increased resolution, and touch-screen capability. [9] (www.altera.com/literature/wp/wp-01100-graphic-lcd-display.pdf)

3. The controller is based on an SRAM-based FPGA, which means that it's a fully reprogrammable device. In fact, the design involves an integration of three controllers (SVGA, SDRAM, and FLASH), thereby providing additional functionality to any embedded system. A resolution of 800 x 600 using 16-bit graphics with an 8051 processor, for example, is now possible. Access to 32 MB of on-board memory plus 8 MB of Flash memory provides the ability to build projects that require larger amounts of memory.[10] (www.eetimes.com/document.asp?doc_id=1316433)

4. Silicon Spectrum was founded in 2002 and they licensed and re-implemented an FPGA technology for creating a graphics accelerator. The reason behind this was to provide a binary compatible graphics core for vertical markets: Medical Imaging, Military, Industrial, and Server products. Our 2D/ 3D implementation, has since also been used in Digital picture frames, Avionics equipment and on a range of FPGA based products. Now they are running a Kickstarter to finish the final few modes of operation with the goal of a complete, professional, open source hardware platform. The current aim is to finish the final few modes of operation and release their product as a complete, professional, open source hardware platform. They will provide a complete Verilog implementation of a 2D/3D graphics processor capable of OpenGL and D3D with a full test suite that anyone is free to use and improve upon. They Ultimate stretch goal is to create a complete open source implementation of a modern day graphics accelerator. [11] (www.kickstarter.com/projects/.../open-source-graphics-processor-gpu)

Problem Statement and Motivation

This project stemmed from addressing the problem of providing color graphics for constrained embedded applications, such as a microcontroller. In the situation such as a microcontroller you often need extra hardware to provide color graphics, but then you are still limited to memory constraints of the microcontroller.
The proposed solution to this problem was to have an FPGA act as the GPU co-processor to the embedded application.
This GPU supports an extensible number of drawing modules since a common data-flow scheme is inherent to all modules. Current modules include the functionality for the drawing of lines, circles, rectangles, and polygons. As well as the filling of both circles and rectangles, and that of writing text to the monitor.

The end-user will be able to write graphics by including a graphics module in their program that contains all the code necessary to write to the FPGA. Graphics functions will have a simple English naming convention, such as:

- graphics_fill_circle(canvas, x, y, r, colour)
- graphics_draw_string(canvas, x, y, back, &bitstreamverasans10, colour, FS_NONE)
- graphics_fill_canvas(canvas, colour);
- graphics_fill_rect(canvas, x, y, l, b, colour);

Introduction

A graphics system consists of several levels of hardware and software working together to make a computer capable of displaying visual information on a monitor. The primary engine behind the graphics system is the graphics controller, which consists of the custom hardware processor that performs all necessary mathematical operations required to generate pixel data (RGB colour), which is then stored in a region of memory called the frame buffer. A display controller then reads the frame buffer and converts the RGB data to pixels on-screen. A software driver is also necessary to stimulate the hardware and provide an API (Application Programming Interface) for developers to create graphics applications. Modern graphics controllers are usually implemented as ASIC's (Application Specific Integrated Circuits), primarily due to their high performance requirements, especially for intensive 3D applications. The goal of our project was to implement a graphics system on an FPGA. In particular, we designed and implemented the necessary hardware and software components to perform basic 2D operations.

Desired operations:

o Draw pixel
o Blit (fill a rectangular region on the screen)
o Draw line
o Draw character

Proposed Graphic Engine Framework

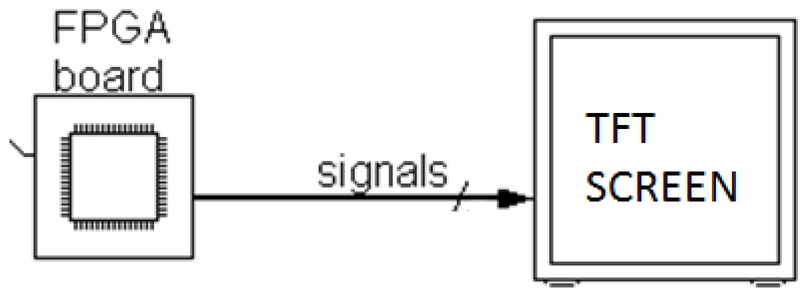


Figure 1

A simple two-dimensional graphics processing unit (GPU) was implemented on an Altium Nanoboard FPGA. The GPU is capable of understanding 80bit instructions sequentially sent to it via RS-232 serial communication. The only requirement being from a user side is that they have installed the proper C files and have a means of sending a byte over serial. As instruction requests are handled, they are appropriately written into SRAM and later displayed on a VGA compatible monitor. This GPU supports an extensible number of drawing modules since a common data-flow scheme is inherent to all modules. Current modules include the functionality for the drawing of lines, circles, rectangles, and polygons. As well as the filling of both circles and rectangles, and that of writing text to the monitor.

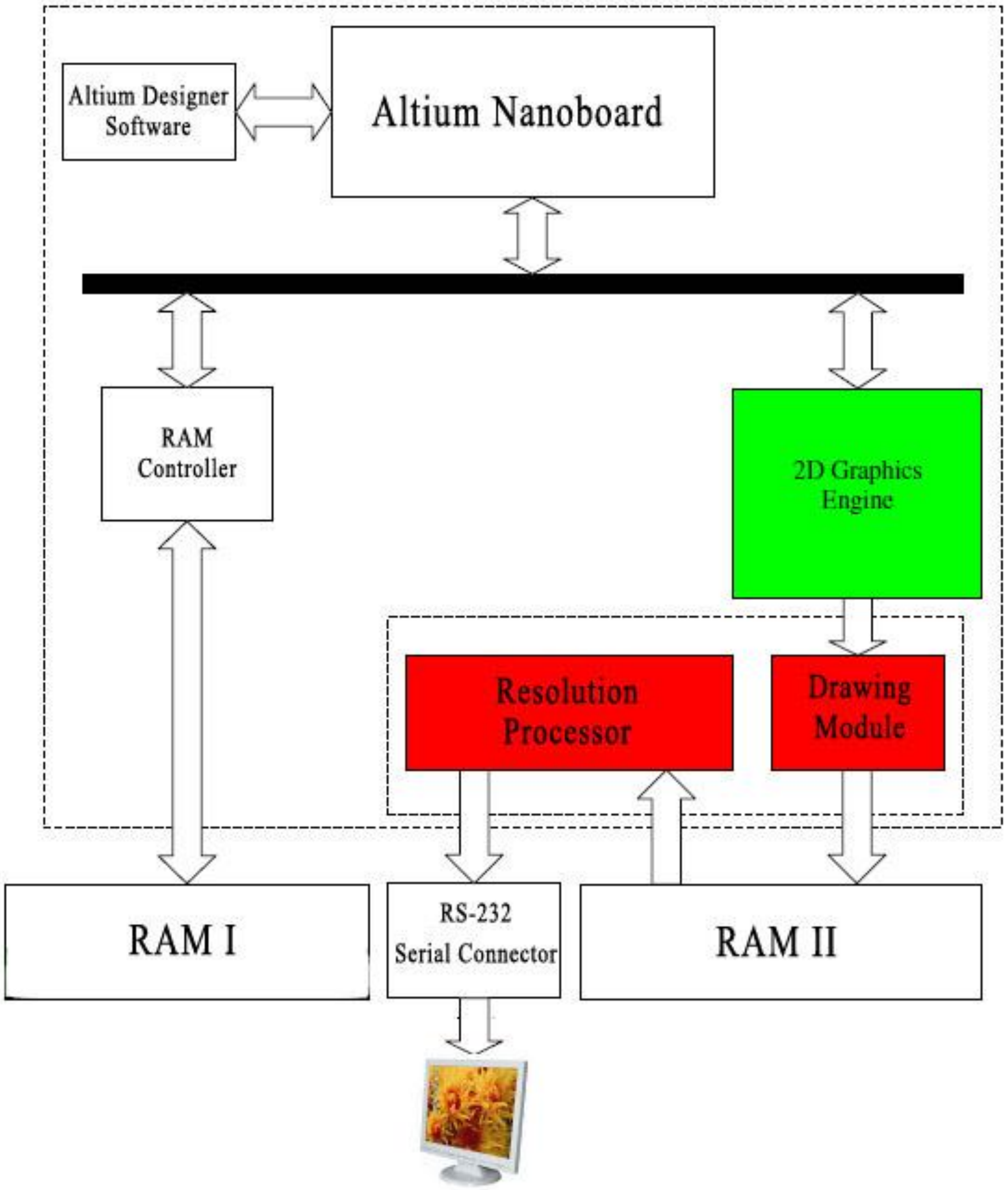


Figure 2

Altium Nano Board : Nano board in the series offers a reprogrammable hardware platform that harnesses the power of a high-capacity, low-cost programmable device to allow rapid and interactive implementation and debugging of digital designs.

<u>Altium Designer Software :</u> Altium Designer is an EDA software package for printed circuit board, FPGA and embedded software design, and associated library and release management automation. It is developed and marketed by Altium Limited of Australia. It consists of :

- Schematic capture
- PCB design
- FPGA and embedded software tools

<u>RS-232 Serial Connector :</u> An RS-232 serial port was once a standard feature of a personal computer, used for connections to modems, printers, mice, data storage, uninterruptible power supplies, and other peripheral devices. However, the low transmission speed, large voltage swing, and large standard connectors motivated development of the Universal Serial Bus, which has displaced RS-232 from most of its peripheral interface roles.

<u>Resolution Processor :</u> Maintains the resolution of the image displayed on the monitor. Here the image resolution is 640x480

<u>Drawing Module :</u> It consists of various modules (functions) used to draw various figures on the VGA.

Rectangle Fills and Draws
The rectangle fill works by starting at the starting coordinate (X,Y) and iterating across X and down Y until the coordinate (X+width,Y+height) is reached. Each pixel is written sequentially. The rectangle draw works by iterating over each wall of the rectangle, north, south, east, and west.

Line Drawing
The line drawing module is implemented based on the well-known Bresenham algorithm.

Block Diagram

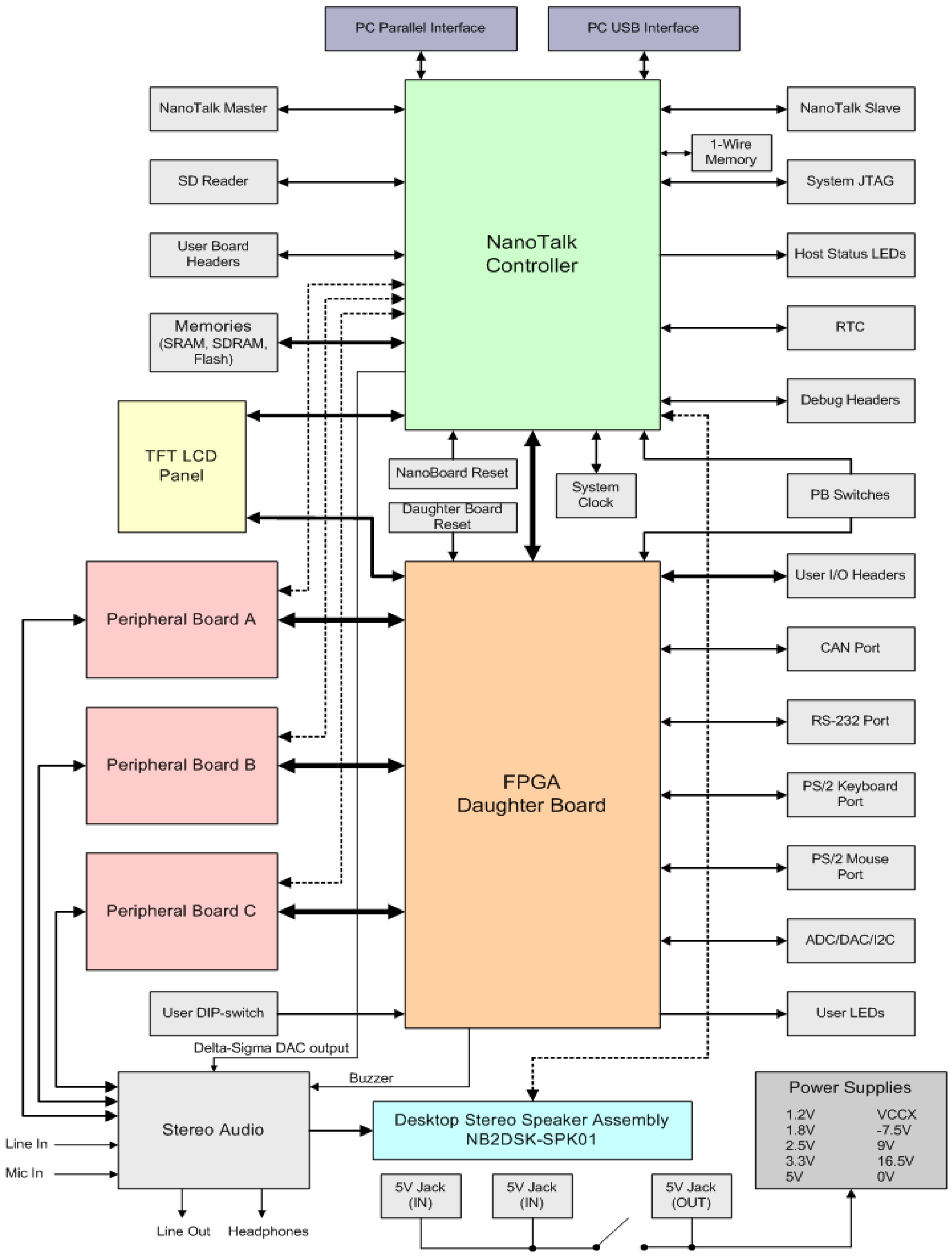


Figure 3

Hardware Specifications

Altium Nano Board:

Nano board in the series offers a reprogrammable hardware platform that harnesses the power of a high-capacity, low-cost programmable device to allow rapid and interactive implementation and debugging of digital designs.

Specifications:

- One 100-way peripheral board connector ('NANOCONNECT' interface)
- 5V DC power connector with power switch
- Altium NanoTalk USB 2.0 PC interface
- 4x Serial SPI Flash memory devices
- One containing Primary boot image for the Host Controller
- One containing Golden boot image for Host Controller
- Two for use by User FPGA (for boot/embedded usage)
- Dedicated System JTAG programming port
- 1-Wire® memory device used to store board ID and related information
- Programmable clock 6 to 200 MHz, accessible by Altium Designer or by a User FPGA design
- Fixed 20MHz reference clock
- SPI Real-Time Clock with 3V battery backup
- Adjustable voltage regulators set to generate 1.2V, 1.8V, 2.5V and 3.3V power
- Power supply test points for all supply levels available on the board, including GND points
- On-board memories accessible by User FPGA
- 256K x 32-bit common-bus SRAM (1MByte)
- 16M x 32-bit common-bus SDRAM (64MByte)
- 8M x 16-bit common-bus 3.0V Page Mode Flash memory (16MByte)
- Dual 256K x 16-bit independent SRAM (512KByte each)
- On-board memory accessible by Host Controller FPGA:
- 256K x 16-bit independent SRAM (512KByte)
- LED array, 8 RGB LEDS
- 5 generic push-button switches
- 4-channel 8-bit ADC, SPI-compatible
- 4-channel 8-bit DAC, SPI-compatible
- 4x isolated IM Relay channels
- 4x PWM power drivers
- Screw terminal headers for ADC/DAC/Relay/PWM interfaces
- S/PDIF interface

- SD (Secure Digital) card readers:
- One for use by the Host Controller FPGA
- One for use by the User FPGA
- Dual 18-way (20 pin) I/O expansion headers with power supply selection links
- User prototyping area
- SVGA interface (24-bit, 80MHz)
- 10/100 Fast Ethernet interface
- USB 2.0 High-Speed interface
- RS-232 Serial Port – DB9M
- RS-485 Serial Port – 'RJ45'
- PS/2 Mini-DIN Mouse Port
- PS/2 Mini-DIN PC Keyboard Port
- 240 x 320 TFT LCD with touch scree
- 37-pin TFT connector
- Resides on separate satellite PCB, attached to motherboard through a 1.8" ATA/IDE connector
- 12-bit SPI touch screen controller
- 8-way DIP-switch
- USB hub: 3x USB A-type connectors
- ISP1760 Hi-speed USB Host Controller
- Stereo 2W audio power amplifier with 3.5mm test input jack and DC volume control
- 24-bit Stereo Audio CODEC with I2S-compatible interface
- Analog audio input: Stereo line in
- Analog audio output: Stereo line out
- Digital audio data: Transfer over I2S bus
- Audio mixer (L and R) for combining audio sources
- Line Out from Host Controller (Delta-Sigma DAC output, driven by FPGA)
- Line Out signal wired from peripheral board socket
- Line Out signal wired from audio CODEC
- Buzzer output from User FPGA design
- Stereo audio jacks (3.5mm):
- Line In / Line Out
- Speaker sockets for direct connection of speakers, resident on a separate (attached) board
- Status LEDs:
- For Host Controller status
- For User FPGA power and program status
- Diagnostics interface – PCI Express (PCIe) edge connector for connection of automated test equipment (ATE)
- 1.8" ATA/IDE connector providing access to user LED and generic switch I/O

- Test/Reset button

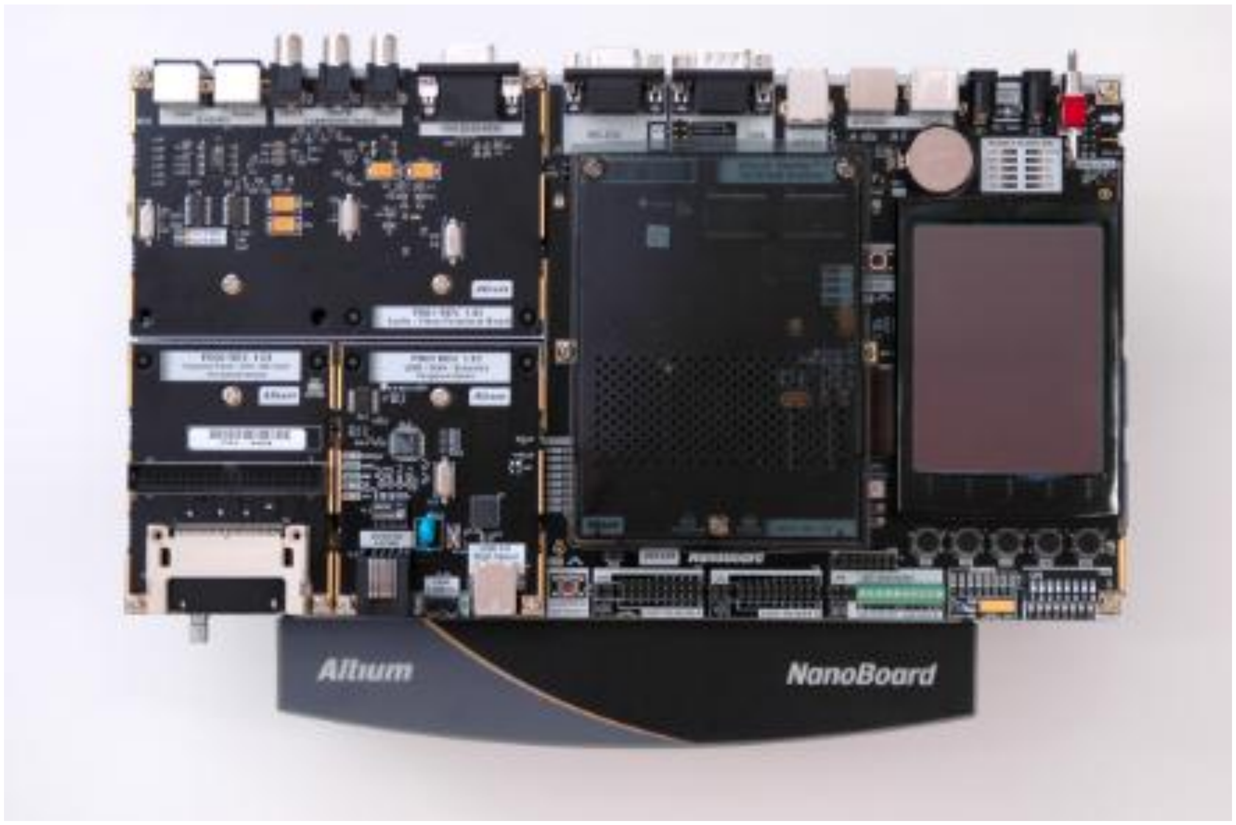


Figure 4

Altium Nanoboard NB2DSK01

Software Specifications

Altium Designer:

Altium Designer is an EDA software package for printed circuit board, FPGA and embedded software design, and associated library and release management automation. It is developed and marketed by Altium Limited of Australia.

Languages Used:

Verilog

Verilog differ from software programming languages because they include ways of describing the propagation of time and signal dependencies (sensitivity). There are two types of assignment operators:

Blocking assignment (=) : The target variable is updated immediately.

Non – blocking assignment (<=): The non-blocking assignment allows designers to describe a state-machine update without needing to declare and use temporary storage variable.

SystemVerilog is a superset of Verilog-2005, with many new features and capabilities to aid design verification and design modeling. As of 2009, the SystemVerilog and Verilog language standards were merged into SystemVerilog 2009 (IEEE Standard 1800-2009).

Simple program to display hello world

```
module main;
initial
   begin
     $display("Hello world!");
     $finish;
   end
endmodule
```

C Programming Language

C is one of the most widely used programming languages of all time, and C compilers are available for the majority of available computer architectures and operating systems.

Design Specifications

In development of the GPU the issues to address can be split into a few categories: performance parameters, functionality, end-user design and required components.

Performance Parameters
Performance Parameters includes what the user can expect to achieve from the GPU. These are things such as frames per second and available resolutions and color depth.

Drawing Functionality

- Shapes – filled and unfilled (Circles, Rectangles, Polygons, and lines)
- Character display – 8x8 pixel characters to be printable on the screen
- Frames – full resolution frames

End-user Design
The end-user will be able to write graphics by including a graphics module in their program that contains all the code necessary to write to the FPGA. Graphics functions will have a simple English naming convention, such as fillRectangle(x1,y1,w,h,red,green,blue).
An API will be available to explain in detail how to write to the FPGA and what methods are available.

Image Memory

One aspect of the project was where to store the image memory. To support 16 colors (4 bits) in two frames at a resolution of 640x480 requires:

0.5bytes * 640 pixels * 480 pixels * 2 frames = 307.2Kbytes

GPU should have enough memory for storing both image frames.
The next criteria is how easy it will be to access each type of memory, since the graphics card will constantly be reading and writing to the memory.

Video Display

Standard graphics modes are:

- 640×480 in 16 colors
- 320×240 in 16 colors
- 320×240 in 256 colors

Activity Time Chart

Work done till Mid-Semester

Mid-August: Right from the very beginning of this semester, we wanted to learn about Verilog and FPGA kit, and this project gives us the required platform. We finalized our work and thus started on this.
The project is divided in 3 phase:

- Study of Verilog (Implementation of basic modules)
- Interfacing with FPGA kit
- Implementation of project

Work done till End Semester

The software API has been developed with a basic of GUI to be capable of running a low-level C function corresponding to each of the 4 basic 2D graphics operations implemented in hardware: draw pixel, blit, draw line, and draw character.
The following low-level functions have been implemented:
• graphics_draw_string: Draw a string on screen with specific colour.
• graphics_fill_circle: Fill an area of circle with a specific colour.
• graphics_draw_line: Draw a line on screen.

Using these basic building blocks we were able to create some simple animations like bouncing balls and moving text.

The output has been shown on the TFT screen connected on the altium kit.

Problems Faced

- Interfacing Altium Nanoboard with the VGA/TFT was quite difficult.
- Defining the delays for smooth operation.
- Understanding of schematic and memory distribution.
- Flickering of the screen due to different refresh rates.

Future Work and Improvement

The following are some proposed features for future work:

- **2D Operations**: The graphics engine was designed to be easily expanded for more custom 2D operations. We could easily implement additional algorithms such as drawing textures.

- **Image Manipulation**: Functionality can be added to skew, stretch, invert, or rotate an image on screen.

- **Effects**: Image effects like shading, alpha blending, or blurring can be added.

- **Media**: Support for image/movie files can be added. For full scale projects like a JPEG or MPEG decoder, our project can be used as a very good starting point.

- **Advanced Software API**: The API can be enhanced to be more object oriented, so that to allow more powerful operations. Games and other animations can be made.

OPEN BUS DIAGRAM

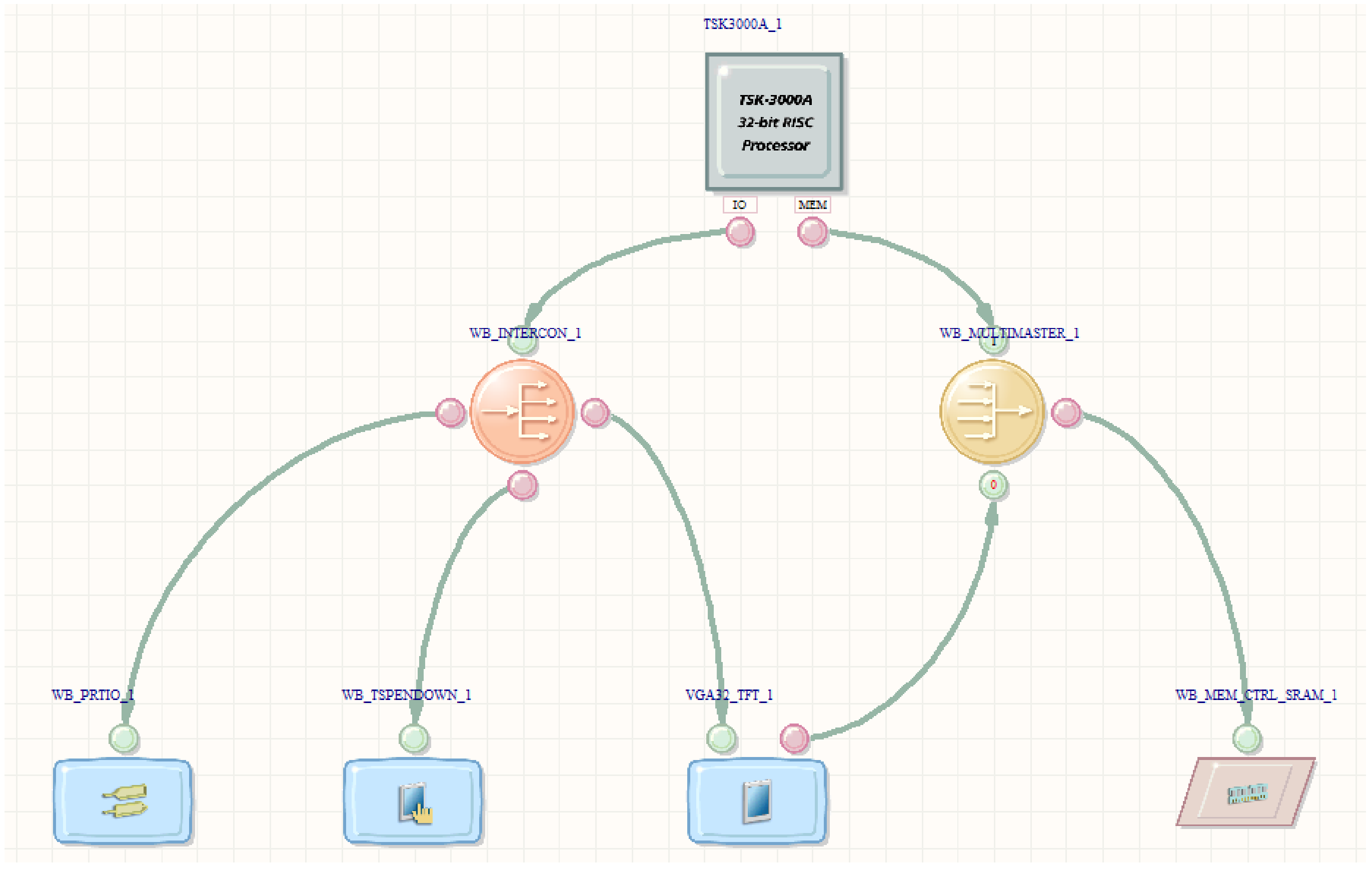


Figure 5

The starting point for any OpenBus System document is the placement of the required devices that will consitute your system. In the schematic world, design components reside in libraries and are placed on the sheet directly from the **Libraries** panel. In the OpenBus Editor, the corresponding OpenBus components are placed from the **OpenBus Palette** panel (Figure 1).

SCHEMATIC DIAGRAM

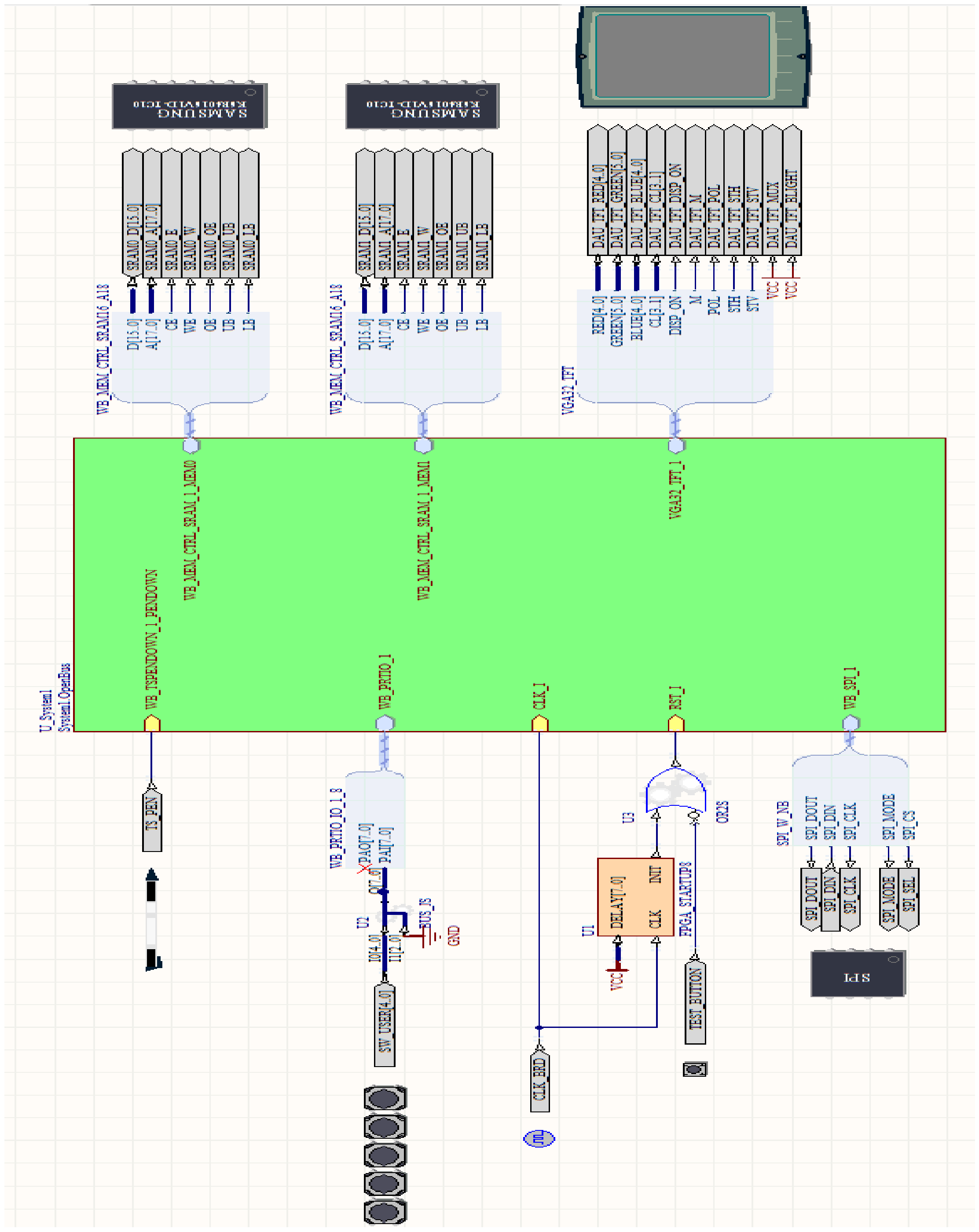


Figure 6

The Schematic Editor, which allows you to check and print the schematic sheets that make up a design project. The tools and utilities needed to perform checks for electrical and drafting violations, generate reports and create presentation quality schematic drawings are available in the editor.

- A main design window in which to view the design.
- Editor-specific menus and toolbars.
- Workspace panels – both global and editor-specific.

<u>DEVICE STACK DIAGRAM</u>

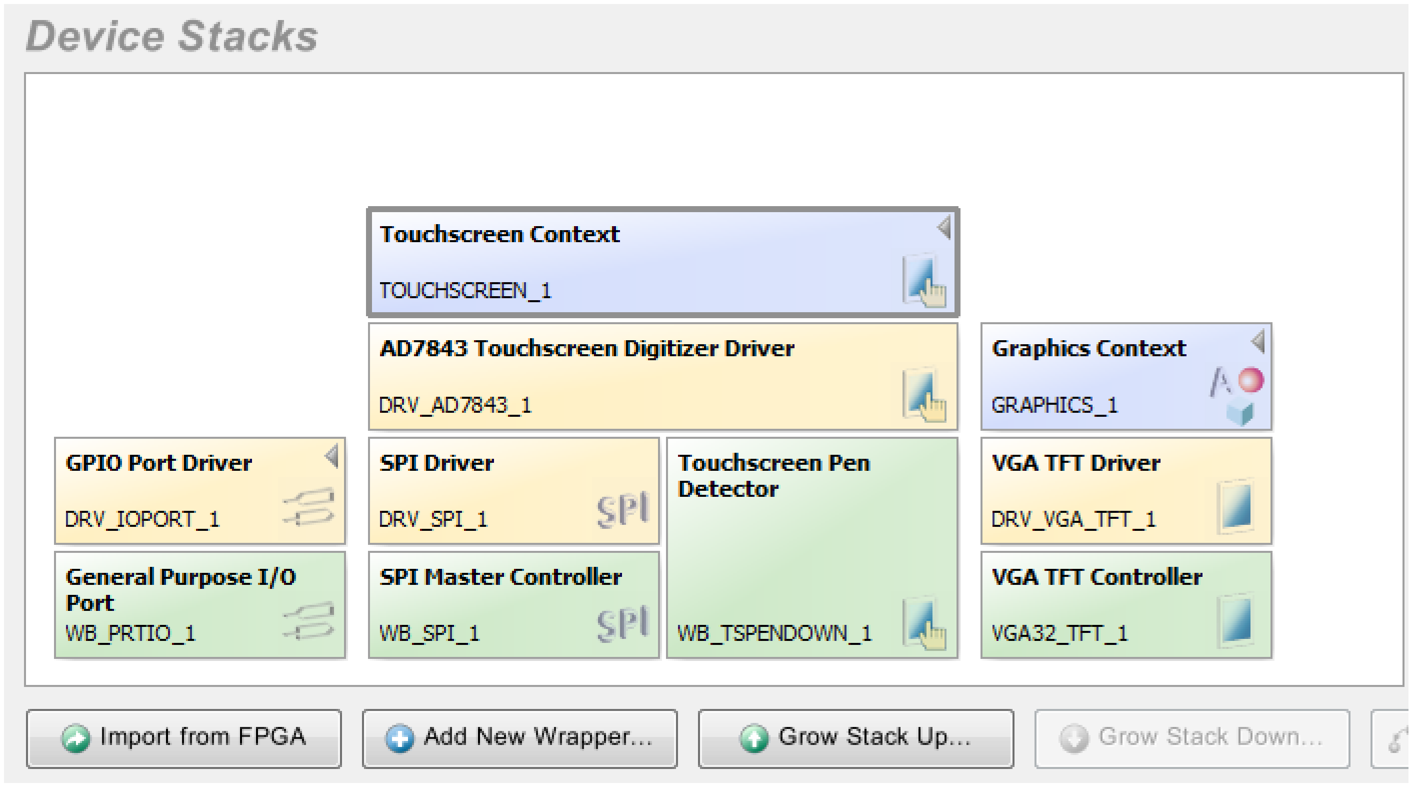


FIGURE 7

The *Software Platform Builder* is the graphical interface that you can use to work with the modules in the Software Platform. The Software Platform Builder enables you to:

- Automatically import modules based on the peripherals devices on the hardware design
- Add or delete hardware wrappers
- Build and edit stacks that are build up hardware wrappers
- Configure modules
- Access help information and inspect public header files

To use the Software Platform Builder you need an Embedded Project. The Software Platform is all about making hardware devices available to application code so it makes sense to associate the Embedded Project with an FPGA design. The Software Platform Builder becomes available when you add the special Software Platform document

The Software Platform Builder consists of two main areas: the **Device Stacks** area and the **Software Services** area. The left part of each section shows which device stacks (wrappers, drivers and/or contexts) or which services are selected to be compiled with your project. The right part of each section shows the configuration of the selected wrapper, driver, context or service.

OUTPUT

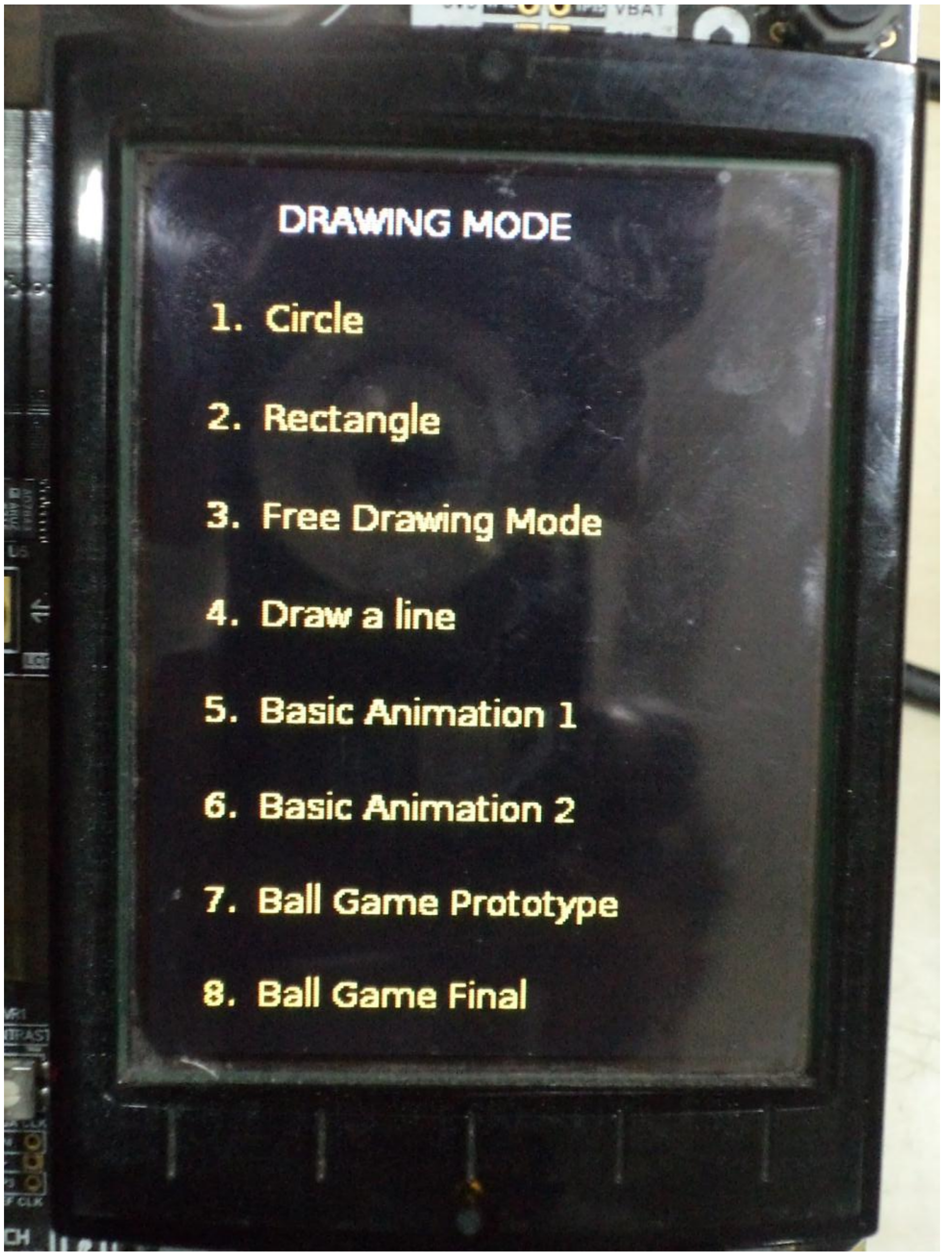


FIGURE 8

MENU DISPLAY ON TFT SCREEN

BOUNCING BALL ANIMATION

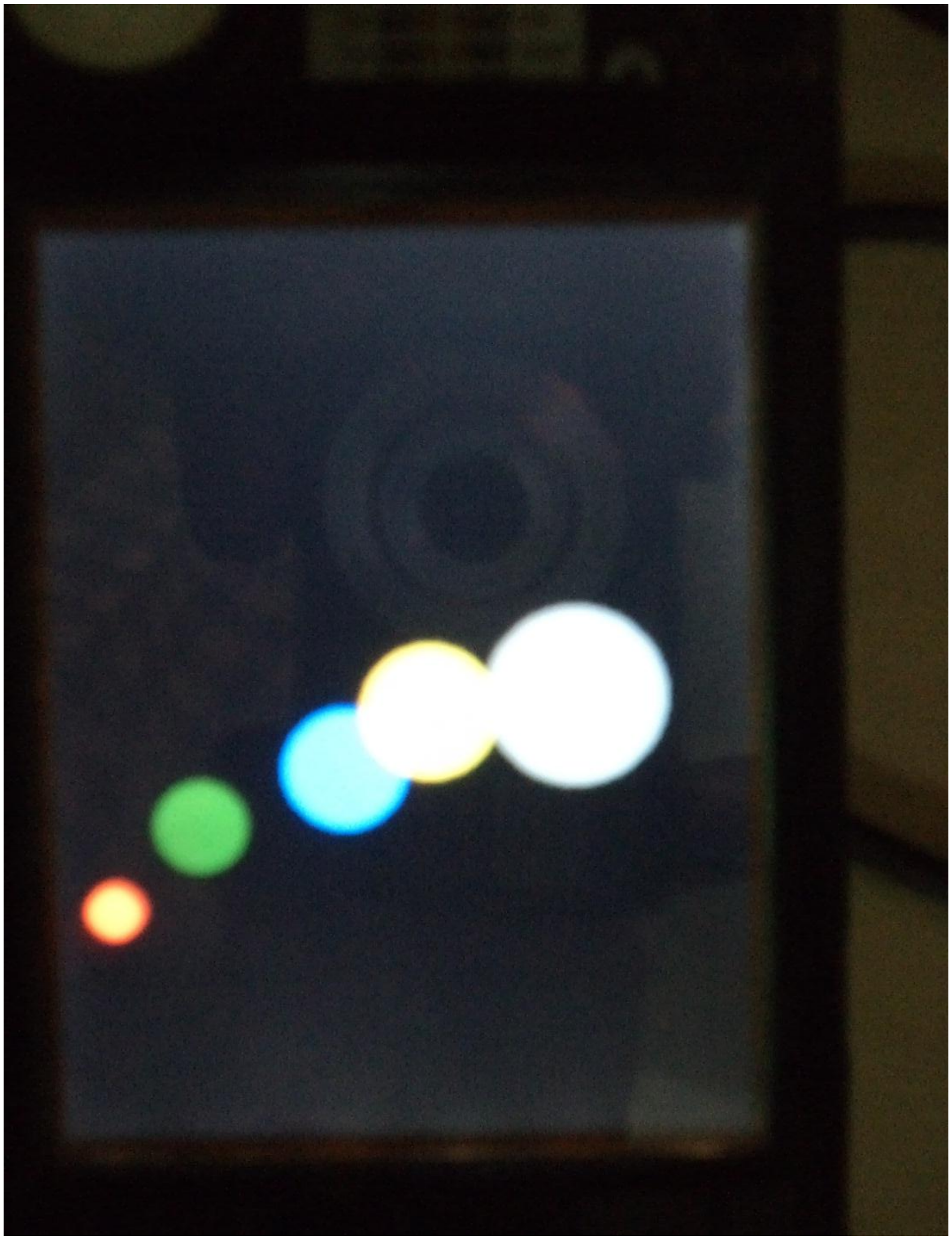
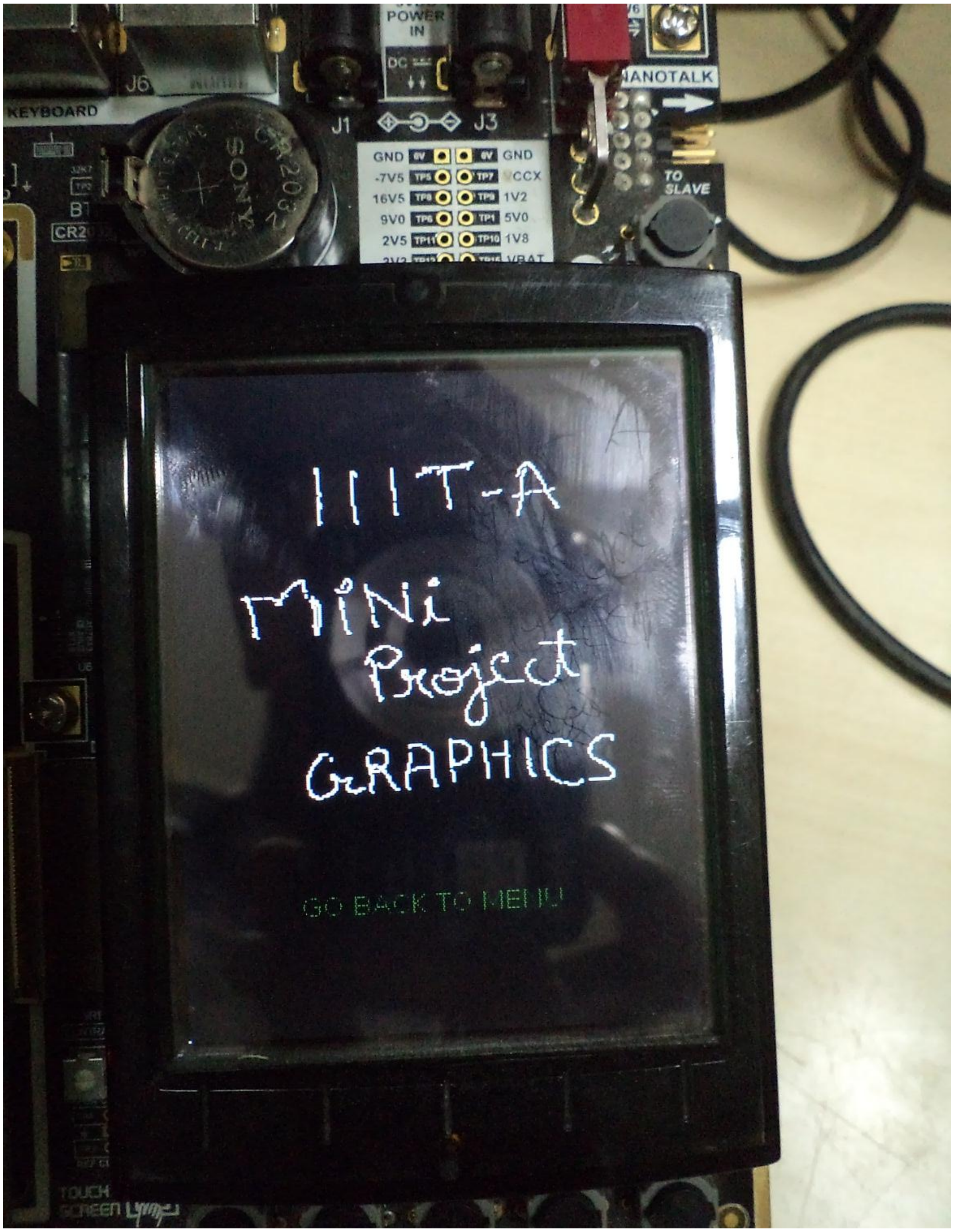


FIGURE 9



FREE HAND DRAWING MODE

Figure 10

CODE

```
#include <stdio.h>
#include <graphics.h>
#include "devices.h"
#include <touchscreen.h>
#include <time.h>

graphics_t *graphics;  /* Global Variables */
canvas_t *canvas;
touchscreen_t *touchscreen;
touchscreen_data_t pos, temp_pos;
int pos_x1, pos_y1, pos_x2, pos_y2;
char *back = " GO BACK TO MENU ";

void Delay (unsigned int Sec)  /*Give delays in seconds */
{
      clock_t ticks1 = clock(), ticks2 = ticks1;
      while ( ( ticks2/CLOCKS_PER_SEC - ticks1/CLOCKS_PER_SEC ) < Sec)
          ticks2 = clock();
}

void loop_delay()  /*Give delays in ms */
{
   int i;
   for (i = 0; i < 100000; i++);
}

void animation_circle()  /* Increasing radius circle */
{
   int r = 0;
   graphics_fill_canvas(canvas, BLACK);

   for (r = 1; r <= 100; r++) {
      graphics_fill_circle(canvas, 120, 150, r, SILVER);
      loop_delay();
   }
   graphics_fill_canvas(canvas, BLACK);
   touchscreen_get_pos(touchscreen, &temp_pos);
   graphics_draw_string(canvas, 50, 250, back, &bitstreamverasans10, GREEN, FS_NONE);
   while(!(temp_pos.y >= 220 && temp_pos.y <= 280)) {
         touchscreen_get_pos(touchscreen, &temp_pos);
   }

   graphics_fill_canvas(canvas, BLACK);
}

void animation_ball() /* Bouncing ball vertical direction */
```

```
{
   int i;
   graphics_fill_canvas(canvas, BLACK);
   int y = 150, inc = 1;
   for (int i = 0; i < 1000; i++) {

      graphics_fill_circle(canvas, 90, y, 50, RED);
      y += inc;
      loop_delay();
      if (y == 240) {
         inc = inc * (-1);
      } else if (y == 140) {
         inc = inc * (-1);
      }
      graphics_fill_canvas(canvas, BLACK);
   }

   touchscreen_get_pos(touchscreen, &temp_pos);
   graphics_draw_string(canvas, 50, 250, back, &bitstreamverasans10, GREEN, FS_NONE);
   while(!(temp_pos.y >= 220 && temp_pos.y <= 280)) {
        touchscreen_get_pos(touchscreen, &temp_pos);
   }

   graphics_fill_canvas(canvas, BLACK);
}

void animation_multiball()  /* Multiple bouncing balls */
{
   int i, j;
   graphics_fill_canvas(canvas, BLACK);
   int y[] = {150, 180, 200, 220, 225} , inc[] = {1, 1, 1, 1, 1}, x[] = {20, 50, 100, 130, 180}, r[] = {8, 15, 18, 15,
20};
   color_t color[] = {RED, GREEN, BLUE, YELLOW, WHITE};
   for (i = 0; i < 1000; i++) {
      for (j = 0; j < 5; j++) {
         graphics_fill_circle(canvas, x[j], y[j], r[j], color[j]);
         y[j] += inc[j];
         loop_delay();
            if (y[j] == 240) {
            inc[j] = inc[j] * (-1);
            } else if (y[j] == 140) {
               inc[j] = inc[j] * (-1);
            }
      }
      graphics_fill_canvas(canvas, BLACK);
   }

   touchscreen_get_pos(touchscreen, &temp_pos);
```

```
    graphics_draw_string(canvas, 50, 250, back, &bitstreamverasans10, GREEN, FS_NONE);
    while(!(temp_pos.y >= 220 && temp_pos.y <= 280)) {
          touchscreen_get_pos(touchscreen, &temp_pos);
    }

    graphics_fill_canvas(canvas, BLACK);
}

void animation_ballgame()  /* Bouncing ball in all directions */
{
    int i, x = 150, y = 150, incx = 1, incy = 1;
    graphics_fill_canvas(canvas, BLACK);

    for (i = 0; i < 1000; i++) {
        graphics_fill_circle(canvas, x, y, 20, RED);
         y += incy;
         x += incx;
         loop_delay();
         if (y == 240) {
            incy = incy * (-1);
         } else if (y== 140) {
            incy = incy * (-1);
         }
         if (x == 220) {
            incx = incx * (-1);
         } else if (x == 50) {
            incx = incx * (-1);
         }
         graphics_fill_canvas(canvas, BLACK);
    }
    touchscreen_get_pos(touchscreen, &temp_pos);
    graphics_draw_string(canvas, 50, 250, back, &bitstreamverasans10, GREEN, FS_NONE);
    while(!(temp_pos.y >= 220 && temp_pos.y <= 280)) {
          touchscreen_get_pos(touchscreen, &temp_pos);
    }

    graphics_fill_canvas(canvas, BLACK);
}

void animation_ballgamefinal() /* Bouncing ball changing direction on touch */
{
    int i, x = 150, y = 150, incx = 1, incy = 1;
    graphics_fill_canvas(canvas, BLACK);

    for (i = 0; i < 1000; i++) {
        graphics_fill_circle(canvas, x, y, 20, RED);
         y += incy;
         x += incx;
         loop_delay();
         if (y == 300) {
            incy = incy * (-1);
         } else if (y== 100) {
            incy = incy * (-1);
         }
         if (x == 240) {
            incx = incx * (-1);
         } else if (x == 40) {
            incx = incx * (-1);
         }
         touchscreen_get_pos(touchscreen, &temp_pos);
         if (temp_pos.x >= x -20 && temp_pos.x <= x - 20 && temp_pos.y >= y - 20 && temp_pos.y <= y +
20) {
            incy = incy * (-1);
            incx = incx * (-1);
         }
         graphics_fill_canvas(canvas, BLACK);
    }
    touchscreen_get_pos(touchscreen, &temp_pos);
    graphics_draw_string(canvas, 50, 250, back, &bitstreamverasans10, GREEN, FS_NONE);
    while(!(temp_pos.y >= 220 && temp_pos.y <= 280)) {
          touchscreen_get_pos(touchscreen, &temp_pos);
    }

    graphics_fill_canvas(canvas, BLACK);
}

int main( void )
{
    bool valid;
    char *cal1 = "Touch screen at marker";
    char *cal2 = "Calibration done";
    char *str1 = "    DRAWING MODE    ";
    char *txt1 = "Give Cordinates";
    char *op1 = "1.  Circle";
    char *op2 = "2.  Rectangle";
    char *op3 = "3.  Free Drawing Mode";
    char *op4 = "4.  Draw a line";
    char *op5 = "5.  Animation Menu ";
    char *op6 = "5.1 Increasing Radius Circle ";
    char *op7 = "5.2 Simple Bouncing Ball ";
    char *op8 = "5.3 Bouncing Ball in All Directions";
    char *op9 = "5.4 Multiple Bouncing Balls";
    char *op10 = "5.5 Ball Touch Game";

    /* open graphics and touchscreen */
    touchscreen = touchscreen_open(TOUCHSCREEN_0);
```

```
   graphics = graphics_open(GRAPHICS_0);
   canvas = graphics_get_visible_canvas(graphics);

   /* set which callback function is called from the calibrate routine */
   touchscreen_set_callback(touchscreen, draw_mark, NULL);

   /* calibrate, repeat until calibrate is ready. Funtion returns true if ready */
   touchscreen_set_rotation(touchscreen, 0);
   graphics_set_rotation(graphics, 0);
   while (!touchscreen_calibrate(touchscreen, 240, 320));

   /* set both graphics and touchscreen to the same rotation */
   touchscreen_set_rotation(touchscreen, 0);
   graphics_set_rotation(graphics, 0);

   /* clear screen */
   canvas_destroy(canvas);
   graphics_fill_canvas(canvas, BLACK);

   while (1)
   {
      valid = 0;

      do {
    graphics_draw_string(canvas, 20, 10, str1, &bitstreamverasans10, WHITE, FS_NONE); /* Print
Drawing Mode */
      graphics_draw_string(canvas, 20, 45, op1, &bitstreamverasans10, YELLOW, FS_NONE);   /* Print 1.
Circle */
      graphics_draw_string(canvas, 20, 80, op2, &bitstreamverasans10, YELLOW, FS_NONE);  /* Print 2.
Rectangle */
      graphics_draw_string(canvas, 20, 115, op3, &bitstreamverasans10, YELLOW, FS_NONE);  /* Print 3.
Free Drawing Mode */
      graphics_draw_string(canvas, 20, 150, op4, &bitstreamverasans10, YELLOW, FS_NONE);  /* Print 4.
Draw line */
      graphics_draw_string(canvas, 20, 185, op5, &bitstreamverasans10, YELLOW, FS_NONE);  /* Print 5.
Animation Menu

            valid = touchscreen_get_pos(touchscreen, &pos);
      } while (valid != 1 && pos.y > 20);

      if(((pos.x) > 20 && (pos.x) < 150) && ((pos.y > 20) && (pos.y < 60))) {  /* Draws Circle */
            graphics_fill_canvas(canvas, BLACK);
            graphics_fill_circle(canvas, 100, 100, 50, SILVER);
            graphics_draw_string(canvas, 50, 250, back, &bitstreamverasans10, GREEN, FS_NONE);
            valid = touchscreen_get_pos(touchscreen, &temp_pos);

            while(!(temp_pos.y >= 220 && temp_pos.y <= 280)) {
                touchscreen_get_pos(touchscreen, &temp_pos);
                }
                graphics_fill_canvas(canvas, BLACK);
                        break;
        } else if(((pos.x) > 20 && (pos.x) < 150) && ((pos.y > 65) && (pos.y < 100))) { /* Draws Rectangle */
                  graphics_fill_canvas(canvas, BLACK);
                  graphics_fill_rect(canvas, 100, 100, 100, 150, FIREBRICK);
                   graphics_draw_string(canvas, 50, 250, back, &bitstreamverasans10, GREEN, FS_NONE);
                   valid = touchscreen_get_pos(touchscreen, &temp_pos);
                   while(!(temp_pos.y >= 220 && temp_pos.y <= 280)) {
                      touchscreen_get_pos(touchscreen, &temp_pos);
                   }
                   graphics_fill_canvas(canvas, BLACK);
                           break;
        } else if (((pos.x) > 20 && (pos.x) < 150) && ((pos.y > 109) && (pos.y < 130))) { /* Free drawing mode
*/
                 graphics_fill_canvas(canvas, BLACK);
                 graphics_draw_string(canvas, 50, 250, back, &bitstreamverasans10, GREEN, FS_NONE);
                  while(!(temp_pos.y >= 220 && temp_pos.y <= 280)) {
                      valid = touchscreen_get_pos(touchscreen, &temp_pos);
                      if ( valid && temp_pos.pendown)      /* if result is valid and pendown is true, draw pixel at
given location */
                      {
                        graphics_draw_pixel(canvas, temp_pos.x, temp_pos.y, WHITE);
                      }
                  }
                  graphics_fill_canvas(canvas, BLACK);
                          break;
        }  else if (((pos.x) > 20 && (pos.x) < 150) && ((pos.y > 135) && (pos.y < 170))) { /* Draw line */
                 graphics_fill_canvas(canvas, BLACK);
                 graphics_draw_string(canvas, 50, 250, back, &bitstreamverasans10, GREEN, FS_NONE);

                 graphics_draw_string(canvas, 20, 150, txt1, &bitstreamverasans10, YELLOW, FS_NONE);
                 Delay(5);
                 graphics_fill_canvas(canvas, BLACK);
                 touchscreen_get_pos(touchscreen, &temp_pos);
                 pos_x1 = temp_pos.x;
                 pos_y1 = temp_pos.y;
                 Delay(3);

                 graphics_draw_string(canvas, 20, 100, txt1, &bitstreamverasans10, YELLOW, FS_NONE);
                 Delay(3);
                 graphics_fill_canvas(canvas, BLACK);
                 touchscreen_get_pos(touchscreen, &temp_pos);
                 pos_x2 = temp_pos.x;
                 pos_y2 = temp_pos.y;

                 graphics_draw_line(canvas, pos_x1, pos_y1, pos_x2+1, pos_y2+1, BLUE);
                  while(!(temp_pos.y >= 220 && temp_pos.y <= 280)) {
                        touchscreen_get_pos(touchscreen, &temp_pos);

                  }
                  graphics_fill_canvas(canvas, BLACK);
                              break;
        } else if (((pos.x) > 20 && (pos.x) < 150) && ((pos.y > 175) && (pos.y < 200))) { /* Enter in Animation
Submenu */
                          valid = 0;
                           do {
                           graphics_fill_canvas(canvas, BLACK);
                           graphics_draw_string(canvas, 50, 250, back, &bitstreamverasans10, GREEN, FS_NONE);
/* Print GO BACK TO MENU */
                           graphics_draw_string(canvas, 80, 10, op5, &bitstreamverasans10, WHITE, FS_NONE);
/* Print Animation Menu
              graphics_draw_string(canvas, 20, 45, op6, &bitstreamverasans10, YELLOW, FS_NONE); /*
Increasing Radius Circle */
              graphics_draw_string(canvas, 20, 80, op7, &bitstreamverasans10, YELLOW, FS_NONE); /*
Simple Bouncing Ball */
              graphics_draw_string(canvas, 20, 115, op8, &bitstreamverasans10, YELLOW, FS_NONE); /*
Bouncing Ball in All Directions*/
                           graphics_draw_string(canvas, 20, 150, op9, &bitstreamverasans10, YELLOW, FS_NONE);
/* Multiple Bouncing Balls */
              graphics_draw_string(canvas, 20, 185, op10, &bitstreamverasans10, YELLOW, FS_NONE); /*
Ball Touch Game */
valid = touchscreen_get_pos(touchscreen, &pos);
 } (valid != 1 && pos.y > 20);

if(((pos.x) > 20 && (pos.x) < 150) && ((pos.y > 20) && (pos.y < 60))) {  /* Increasing Radius Circle */
              animation_circle();
                     break;
        } else if(((pos.x) > 20 && (pos.x) < 150) && ((pos.y > 65) && (pos.y < 100))) { /* Simple Bouncing Ball
*/
animation_ball();
break;
        } else if (((pos.x) > 20 && (pos.x) < 150) && ((pos.y > 109) && (pos.y < 130))) { /* Bouncing Ball in All
Directions*/
animation_multiball();
break;
} else if (((pos.x) > 20 && (pos.x) < 150) && ((pos.y > 135) && (pos.y < 170))) { /* Multiple Bouncing Balls
*/
                     animation_ballgame();
break;
              } else if (((pos.x) > 20 && (pos.x) < 150) && ((pos.y > 175) && (pos.y < 200))) {
                      animation_ballgamefinal();
                   break;
}
}
```

```
    }
}
```

References

[1] S. Brown and J. Rose, "FPGA and CPLD architectures: a tutorial" in *Design & Test of Computers, IEEE* (Volume:13,Issue:2)
[2] Wikipedia: http://www.wikipedia.org. *Field programmable gate array*, 2005.
[3] Altium: http://www.altium.com xilinix platform, 2013.
[4] Fpga4fun: http://www.fpga4fun.com. *Basics of Verilog and implementation of interfaces through FPGA kit*, 1997.
[5] Fawcett, B., "*FPGAs as Reconfigurable Processing Elements*", Circuits and Devices Magazine, March 1996, pp. 8-10
[6] Wiki: http://wiki.altium.com/. *Datasheet of NanoBoard NB2 and communication with kit*, 2012.
[7] G.-G. Mplemenos and I. Papaefstathiou, "*Mplem: An 80-processor fpga based multiprocessor system*," in Field-Programmable Custom Computing Machines, 2008. FCCM '08. 16th International Symposium on, April 2008, pp. 273-274
[8] Wikipedia: http://en.wikipedia.org/wiki/Bresenham%27s_line_algorithm, [Cited: September 10, 2008.]
[9] Altera : *www.altera.com/literature/wp/wp-01100-graphic-lcd-display.pdf*
[10] eetimes: www.eetimes.com/document.asp?doc_id=1316433
[11] kickstarter: www.kickstarter.com/projects/.../open-source-graphics-processor-gpu

Suggestions of the Board members